МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ М.В.ЛОМОНОСОВА
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

Анализ данных, обучение по прецедентам, логические игры, системы WEKA, RapidMiner и MatLab

(ПРАКТИКУМ НА ЭВМ КАФЕДРЫ МАТЕМАТИЧЕСКИХ МЕТОДОВ ПРОГНОЗИРОВАНИЯ)

УЧЕБНОЕ ПОСОБИЕ

Дьяконов А.Г.

Москва, 2010 г.

Печатается по решению редакционно-издательского совета
факультета вычислительной математики и кибернетики
МГУ имени М.В. Ломоносова

Рецензенты:
Ю.И. Журавлёв, д.ф.-м.н., профессор, академик РАН;
К.В. Рудаков, д.ф.-м.н., профессор, чл.-корр. РАН;
В.В. Стрижов, к.ф.-м.н., н.с. ВЦ РАН;
А.А. Докукин, к.ф.-м.н., н.с. ВЦ РАН.

Дьяконов Александр Геннадьевич

Анализ данных, обучение по прецедентам, логические игры, системы WEKA, RapidMiner и MatLab (Практикум на ЭВМ кафедры математических методов прогнозирования): Учебное пособие. – М.: Издательский отдел факультета ВМК МГУ имени М.В. Ломоносова, 2010.

Учебное пособие предназначено для студентов, которые специализируются в области анализа данных (data mining). Описаны основные принципы программирования логических игр, основные модели алгоритмов классификации, кластеризации и восстановления регрессии, а также программное обеспечение для решения задач анализа данных: WEKA, RapidMiner, MatLab.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данном учебном пособии излагается материал для студентов кафедры Математических методов прогнозирования (ММП) факультета ВМК МГУ, который используется на занятиях «Практикум на ЭВМ» в 5 и 6 семестрах. Практические занятия являются важным дополнением к основному курсу «Математические методы распознавания образов» (ММРО), в котором закладываются основы теории обучения по прецедентам. В разное время основной курс читали С.И. Гуров, Л.М. Местецкий (автор первого учебного пособия по курсу [Местецкий, 2002]), А.Г. Дьяконов. С 2007 года курс читает К.В. Воронцов, подготовивший замечательный учебный материал [Воронцов]. К сожалению, в явном виде весь этот материал не удаётся использовать на занятиях практикума по следующим причинам.

1. **На практикуме немного смещены акценты**… Некоторые методы студенты могут легко реализовать, другие лучше брать из готовых библиотек. Некоторые задачи, которые не представляют особой ценности с точки зрения теории, очень важны на практике (например методы «глобальной[1]» или градиентной оптимизации функций). Необходимо **предоставлять студенту работающий код** некоторых алгоритмов и пояснять, как делать с помощью этого кода эксперименты.

2. **Лекции по ММРО не всегда «стыкуются» с занятиями практикума**. Например, студентов хотелось бы «нагрузить» уже в сентябре, но они ещё не знают постановок основных задач и терминологии. Причём не просто «нагрузить», а дать интересное (!) задание, которое «привлечёт» их к предмету (не будучи с ним напрямую связано) и будет полезным в дальнейшем (так появилась глава «Программирование логических игр»).

3. На практикуме гораздо больше приходится объяснять, **что такое «эвристика»**. Вообще, весь практикум посвящён «эвристикам». Надо чётко осознавать, что есть точные методы (например полный перебор), но не всегда они реализуются на практике за приемлемее время. Можно понижать сложность алгоритмов, получая при этом точное решение (например делать альфа-бета отсечения при обходе дерева), но и это не всегда гарантирует решение за приемлемое время. А можно решить «неточно», но за приемлемое время – так появляются эвристики. Причём эвристики могут быть «разного уровня»! Когда мы задаёмся функционалом ошибки при решении задачи регрессии, то уже придумываем эвристику: этот функционал. Когда мы его оптимизируем, то придумываем уже другие эвристики для его оптимизации. Поэтому на практикуме надо **научиться исследовать и правильно интерпретировать результаты** своих экспериментов.

4. **Необходим «упрощённый» материал по теме**. Автору пришлось вести спецсеминары для бакалавров в 2007 – 2009 гг. и он столкнулся с той проблемой, что бакалавры на ВМК не слушают кафедральных курсов, а знать, что такое «задача классификации», «переобучение», «скользящий контроль»

[1] Так в этом пособии названы алгоритмы оптимизации чёрного ящика, см. главу «Методы глобальной оптимизации».

и т.д. должны. Причём эти знания они должны получить достаточно быстро, не забивая голову лишним (поэтому в пособии отсутствуют многие «классические темы», например оценка надёжности алгоритмов, и совсем исключены доказательства). Кроме того, материал пособия использовался в Казахстанском филиале МГУ при чтении потокового курса «Математические основы теории прогнозирования» (2008 – 2010 гг.), на семинарах школы анализа данных «Яндекс» (2009 г.), в основном кафедральном курсе «Алгоритмы, модели, алгебры» (2009 – 2010 гг.) и спецкурсе «Шаманство в анализе данных» (2010 г.).

5. **Необходимо познакомить студентов с языками программирования**, ориентированными на решение рассматриваемых задач, а также **с прикладным программным обеспечением**. Поэтому в данное пособие вошёл материал, посвящённый системе MatLab, на взгляд автора, наиболее удобной для экспериментов по анализу данных. Причём материал, в некотором смысле, уникальный, поскольку существенно отличается от содержания всех справочных и учебных пособий по подобным системам. Здесь на примерах показаны основные функции системы и тонкости их использования. Также описаны популярные программы WEKA и RapidMiner, в которых реализованы различные алгоритмы машинного обучения, MatLab-пакет CLOP/Spider. В настоящее время готовится материал по системе R.

Данное учебное пособие может быть полезным дополнением к различным курсам по машинному обучению и анализу данных (data mining) [Местецкий, 2002], [Воронцов], [Золотых]. В тексте содержатся неточности и опечатки, о которых автору можно сообщить по электронной почте djakonov@mail.ru.

Благодарности

Автор благодарен Юрию Ивановичу Журавлёву и Константину Владимировичу Рудакову за помощь и поддержку, Леониду Моисеевичу Местецкому, Сергею Исаевичу Гурову, Константину Вячеславовичу Воронцову и Дмитрию Петровичу Ветрову за сотрудничество и советы. Отдельное спасибо людям, которые своими ценными замечаниями помогли существенно улучшить учебное пособие: Вадиму Викторовичу Стрижову, Наталье Фёдоровне Дышкант и Алексею Валентиновичу Нефёдову. Данная книга получилась бы гораздо хуже без всех научных и учебно-методических ресурсов, перечисленных в главе «Литература, ссылки». Автор позволил себе не указывать первоисточники по всем рассматриваемым темам, поскольку они легко определяются по ссылкам.

Во время работы над учебным пособием автор был поддержан грантами РФФИ (проекты 10-07-00609, 08-01-00636).

Qu'on ne nous reproche donc plus le manque
de clarté, puisque nous en faisons profession.
Pascal

Общие замечания о проведении экспериментов

В данном учебном пособии для какой-нибудь задачи описываются различные (эвристические) алгоритмы и их модификации, при этом говорится, что «можно применить такую модификацию алгоритма, а можно другую и т.д.» Читатель должен понимать, что когда есть **оптимальный алгоритм** для решения задачи, то использовать, естественно, следует его. В случае, когда такого алгоритма нет, приходится применять различные эвристики (в том числе, придумывать эвристики самим). **Какую эвристику выбрать, решает эксперимент.** В этом, собственно, и заключается типичное задание практикума на ЭВМ (на каф. ММП ВМК МГУ):

1. Изучить «стандартные» эвристики.
2. Разработать свои эвристики.
3. Правильно реализовать «каркас программы» (например, при программировании логических игр – функцию NegaScout, процедуры загрузки и сохранения позиции и т.д.)
4. Провести эксперименты по «навешиванию» эвристик на «каркас».
5. Аккуратно составить отчёт о проделанных экспериментах, сделать выводы, написать «оптимальную» (с вашей точки зрения) программу для решения поставленной задачи / класса задач.

Здесь выполнение п.2 может (и должно) проходить параллельно с выполнением п. 4. Должна чётко соблюдаться схема «гипотеза – эксперимент – выводы». Сначала надо выдвинуть гипотезу: на что может повлиять применение данной эвристики (уменьшится время счёта, увеличится точность решения и т.д.), а потом проверить её в ходе эксперимента и сделать выводы.

Часто приходится исследовать зависимость некоторой величины от параметров. При этом значение величины может получаться в результате работы некоторого алгоритма. Пусть, например, исследуется зависимость качества классификации (конкретного классификатора в конкретной задаче) от длины обучающей выборки[1]. Если просто запустить классификатор на выборках разной длины, построить график (а это необходимо для визуализации закономерности) и сделать вывод (например «качество улучшается при увеличении длины»), то этого будет недостаточно для полноценного исследования.

Во-первых, важно понимать, что если запустить классификатор несколько раз на выборках разной длины (скажем, 10, 50, 100, 250, 500, 1000) и вычислить ошибки классификации, то получится несколько (здесь шесть) **случайных величин**. Для оценки ошибки классификатора на выборке

[1] Терминология объясняется в главе «Задача обучения по прецедентам».

фиксированной длины необходимо сгенерировать несколько выборок такой длины и для каждой построить классификатор, вычислить среднюю ошибку и дисперсию (выборочную) ошибок. Дисперсия позволит судить о том, насколько можно доверять этой средней ошибке. Тогда возможно построение графика-полосы (функция **`errorbar`** в MatLabe) с обозначениями среднеквадратичных отклонений, который позволит делать выводы о свойствах классификатора.

Во-вторых, вывод должен не просто **описывать саму зависимость** («монотонная»), но и **представлять** некоторые её **свойства** (например, график ошибки выпуклый или вогнутый), а также **давать рекомендацию пользователю** (например, какой длины выборки достаточно для достижения такой-то точности).

Глава 1. ПРОГРАММИРОВАНИЕ ЛОГИЧЕСКИХ ИГР

§1.1. Граф игры

Рассмотрим классическую игру **«ним»** (nim). Играют два человека, которые ходят по очереди. Перед каждым ходом на столе лежит несколько кучек спичек. Игрок за один ход должен выбрать одну кучку и разбить её на две непустые кучки разной мощности (содержащие разное число спичек). Например, кучку из 6 спичек можно разбить следующим образом: 5+1, 4+2, а разбиение 3+3 не является допустимым[1]. Проигрывает тот игрок, который не может сделать ход. Начинают игру, как правило, с одной кучки спичек.

Пусть изначально на столе одна кучка из 7 спичек. После хода первого игрока возможны следующие разбиения: 6+1, 5+2, 4+3. Построим граф, у которого на *k*-м уровне будут вершины, которые соответствуют различным **игровым позициям** – разбиениям после *k*-го хода (вершина верхнего нулевого уровня соответствует начальной позиции). Соединим вершину *k*-го уровня с вершиной (*k*+1)-го, если существует допустимый ход, в результате которого возможно соответствующее изменение позиции. Например, если после первого хода позиция была 4+3, то второй игрок может добиться позиции 3+3+1 (разбив первую кучку) или позиции 4+2+1 (разбив вторую кучку). Граф показан на рис. 1.1.1. Такой граф называется **графом игры**.

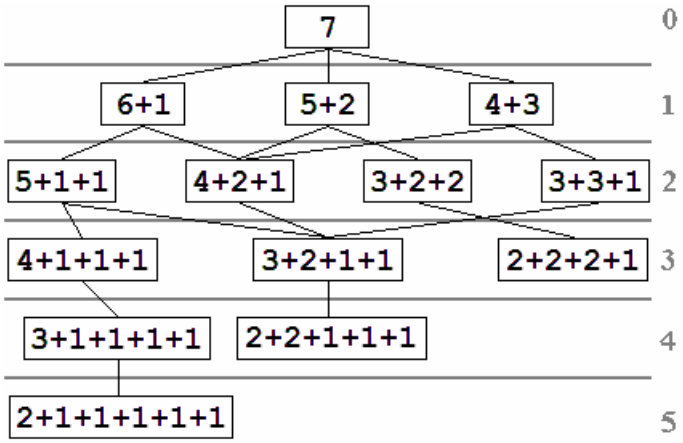


Рис. 1.1.1. Граф игры ним с семью спичками.

Заметим, что в графе есть **терминальные вершины** (2+1+1+1+1+1, 2+2+1+1+1, 2+2+2+1), соответствующие позициям, для которых результат определён правилами игры. Например, в позиции 2+1+1+1+1+1 выигрывает первый игрок, поскольку она «достаётся» второму. Более того, находясь уже в позиции 4+1+1+1, ясно, что победит первый игрок, а находясь в позиции 3+2+1+1, ясно, что победит второй. В позиции 5+1+1 уже нет однозначной определённости: в зависимости от хода может победить первый или второй

[1] Запись «5+1» означает разбиение на две кучки, одна из которых содержит пять спичек, а вторая – одну. Разбиения «*a*+*b*» и «*b*+*a*» отождествляются.

игрок. «К счастью» для первого игрока, ход делает именно он. Ход в позицию 4+1+1+1 обеспечивает ему победу, т.е. **при правильной игре** (первого игрока) в позиции 5+1+1 побеждает первый игрок. Таким образом мы «поднимаемся» по графу, расставляя пометки, которые соответствуют результатам **при правильной игре**. Например, чтобы поставить пометку вершине 6+1 надо знать пометки её **потомков** (вершин следующего уровня, с которыми она соединена рёбрами). Это вершины 5+1+1 (соответствует победе первого игрока) и 4+2+1 (соответствует победе второго игрока). При правильной игре (если второй игрок хочет победить) ход будет в позицию 4+2+1.

Этот процесс можно формализовать следующим образом. Поставим терминальным вершинам метки: метка +1 ставится вершине, если она соответствует победе первого игрока, метка (–1), если она соответствует победе второго. Далее идём по уровням, начиная с уровня, который имеет наибольший номер. Если вершина чётного уровня (соответствует ходу первого игрока) не имеет пометки, то она получает пометку, равную максимуму пометок её потомков. Если вершина нечётного уровня (соответствует ходу второго игрока) не имеет пометки, то она получает пометку, равную минимуму пометок её потомков. Неформально говоря, первый игрок максимизирует на чётных уровнях, а второй игрок минимизирует на нечётных.

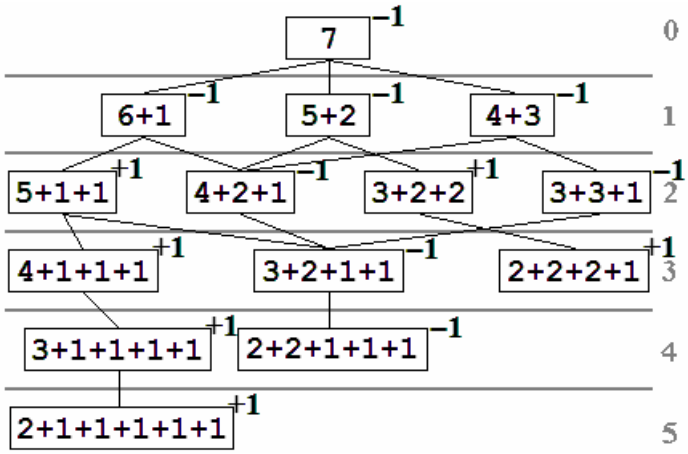


Рис. 1.1.2. Расстановка меток в графе игры.

В итоге получаем, что нулевая вершина имеет пометку –1, то есть **при правильной игре** побеждает второй игрок. Более того, теперь мы даже знаем как. После первого хода первого игрока второй **должен перейти в позицию 4+2+1**. Всё, дальше исход предопределён. Процедура вычисления пометки начальной вершины называется **макс-мин-процедурой** или процедурой **разметки графа игры**.

Нетрудно видеть, что для игр типа шашки, шахматы, уголки, крестики-нолики можно построить аналогичные графы игры. Таким образом, **результат этих игр** тоже, в некотором смысле, **предопределён** (см. дальше). Можно строить дерево (а не произвольный граф): для вершины в *k*-м уровне мы

порождаем в (k+1)-м вершины, соответствующие возможным после допустимого хода позициям, и соединяем её с ними рёбрами (т.е. на рис. 1.1.3 во втором уровне будет 3 вершины, которые соответствуют позиции 4+2+1). Такое дерево называется **деревом игры** или **деревом перебора**. Проблема заключается в том, что это дерево для многих игр имеет гигантские размеры. Например, в шахматах 20 первых допустимых ходов (16 пешками и 4 конями). Договоримся, что **будем также называть их полуходами**, поскольку фраза «в шахматной партии сделано 10 ходов» означает, что 10 раз пошли белыми фигурами и 10 раз чёрными. Ход белыми – одни полуход, ход чёрными – второй полуход, которые вместе образуют ход. Попробуйте подсчитать, сколько вершин в дереве, которое имеет 10 уровней, и каждая вершина которого имеет 20 потомков? Конечно, в реальном дереве число потомков варьируется от вершины к вершине (например, в случае шаха вариантов ответа, как правило, не очень много), но в шахматах для произвольной допустимой позиции в среднем порядка 40 допустимых ходов!

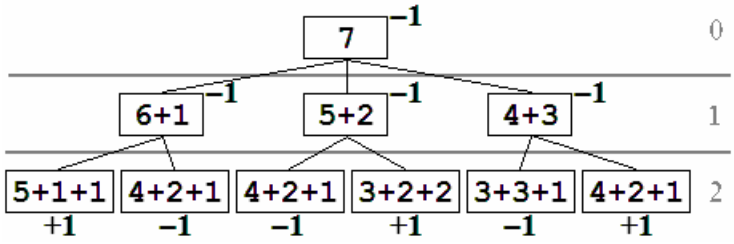


Рис. 1.1.3. Первые уровни дерева для игры ним с семью спичками.

ДЗ *Постройте дерево* (*или граф*) *игры крестики-нолики и проведите для него max-min-процедуру.*

ДЗ *Найдите оптимальные стратегии в игре ним с* 8 *спичками.*

Замечание. Может показаться странным, что от графа игры мы перешли к «избыточному» дереву игры (многие вершины в котором дублируются). Практически все программы работают именно с деревом, поскольку для макс-мин-процедуры его не нужно хранить целиком, а достаточно порождать рекурсивно. Заметим, что так можно достаточно быстро проанализировать дерево игры, которое даже не помещается в оперативной памяти. Естественно, при этом можно учитывать, что некоторые вершины уже проанализированы и если они опять появятся в дереве (при рекурсивном переборе), то их пометку можно не вычислять (об этом см. дальше).

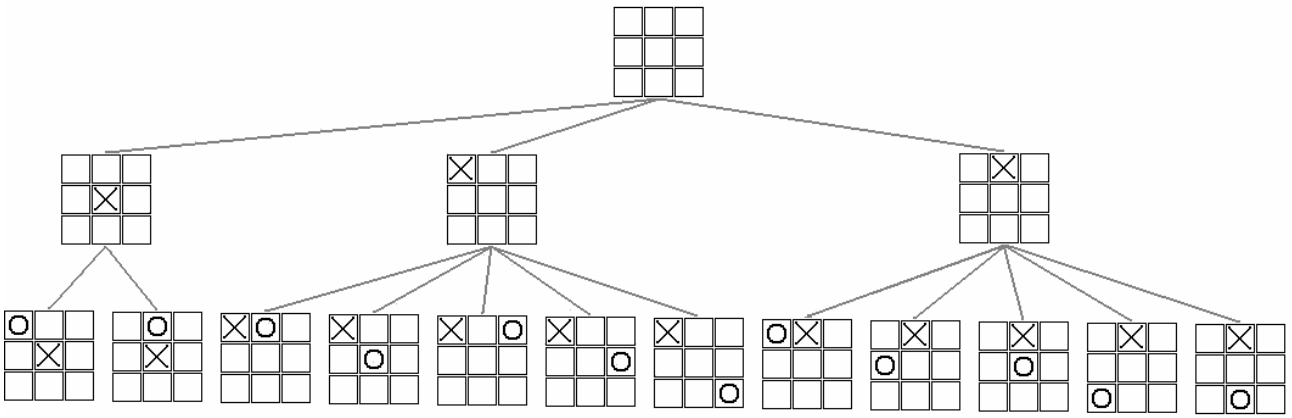


Рис. 1.1.4. Первые уровни дерева игры крестики-нолики[1].

§ 1.2. Альфа-бета алгоритм

Что же делать в том случае, когда построение всего дерева невозможно по причине ограниченности нашего времени (мы не можем думать над ходом очень долго) и памяти (при программировании логических игр на ЭВМ дерево надо ещё где-то хранить)? Мы можем строить не всё дерево, а только несколько его уровней с номерами не выше *r*. Это соответствует тому, что мы обозреваем **все позиции вперёд на *r* полуходов**. Если бы мы знали метки терминальных вершин (а в данном случае мы знаем метки только некоторых, которым соответствуют позиции окончания игры[2]), то мы бы узнали результат игры и оптимальный первый ход. Мы можем попробовать заменить метки терминальных вершин их **приближениями**. Пусть существует некоторая **функция оценки**, которая по любой позиции получает целое число в интервале [–INF, +INF], где INF – достаточно большая константа. **Чем больше значение функции оценки, тем выгоднее позиция для первого игрока.** Нулевое значение соответствует позиции, в которой шансы примерно равны. Значение +INF соответствует победе первого игрока, а значение (–INF) – победе второго.

Во многих играх легко придумать «естественную» функцию оценки. Например, в играх, где побеждает тот игрок, который имеет больше фишек к концу игры (реверси), функция оценки может выдавать разность между числом фишек первого и второго игрока. В игре крестики-нолики это может быть разность между числом линий, которые содержат крестики и не содержат нолики, и числом линий, которые содержат нолики и не содержат крестики (см. рис. 1.2.1).

---

[1] Вообще говоря, из начальной вершины должно следовать 9 рёбер, поскольку первый игрок имеет 9 допустимых ходов (способов поставить крестик). Но получаемые 9 позиций можно рассматривать с точностью до симметрии игрового поля. При разборе игры «на бумаге» такое сокращение сильно экономит время. К сожалению, при программировании логических игр не всегда целесообразно разбирать позиции с точностью до симметрии. **ДЗ** *Проведите эксперименты, чтобы убедиться в целесообразности применения сокращения перебора разбора за счёт отслеживания симметрии в позициях.*

[2] Терминальные вершины последнего слоя других позиций не помечены.

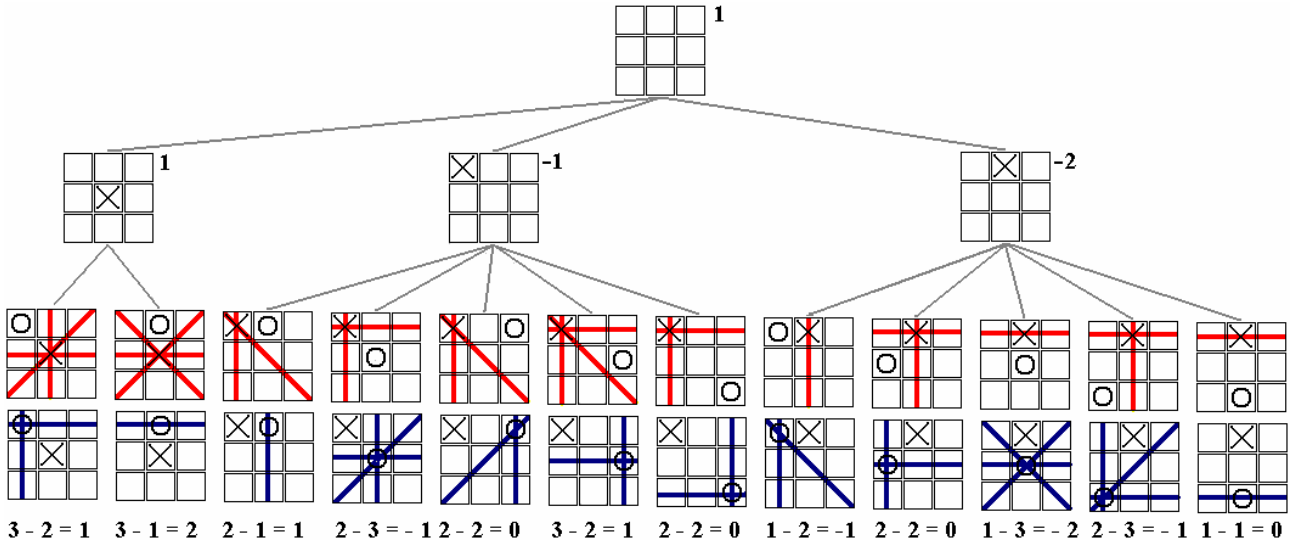


Рис. 1.2.1. Применение простой функции оценки в «крестиках-ноликах».

Ясно также, что для многих игр функция оценки может быть очень нетривиальна. Так, в шахматах надо БЫ учесть:
1) безопасность короля,
2) материальную оценку,
3) «свободу» своих фигур (число клеток, на которые можно сделать ход, какие фигуры можно съесть и т.д.),
4) хорошее расположение пешек (наличие пешечных цепей, контроль центра, близость к последней горизонтали)
5) оценка расклада (например, считается, что иметь двух слонов без коней лучше, чем двух коней без слонов).

ДЗ *Подумайте, как учесть все эти факторы. Чем можно ещё пополнить этот список?*

Под материальной оценкой понимается «перевес в фигурах», например разница между суммами весов фигур соперников. Вес пешки – 1, слона – 3, коня – 3, ладьи – 5, ферзя – 9, короля – (+INF).

Если терминальным вершинам приписать значения функции оценки, то, «поднимаясь» по дереву (как мы это делали раньше), мы вычислим оценку исходной позиции (точнее её приближение). Опять первый игрок на чётных уровнях максимизирует (он делает такой ход, чтобы попасть в позицию с максимальным значением пометки), а второй на нечётных уровнях минимизирует. Функция оценки может быть не совсем корректной. Она эвристическая, и её значение не гарантирует, что позиция действительно хороша или плоха. Поскольку мы использовали её для оценки только терминальных позиций, оценка исходной позиции более адекватна – она определяет, что **мы не получим позицию хуже этой оценки через *r***

полуходов[1]. Естественно, **если мы гарантированно выигрываем или проигрываем меньше чем через *r* полуходов, то мы получим оценку начальной позиции равной соответственно +INF или –INF.**

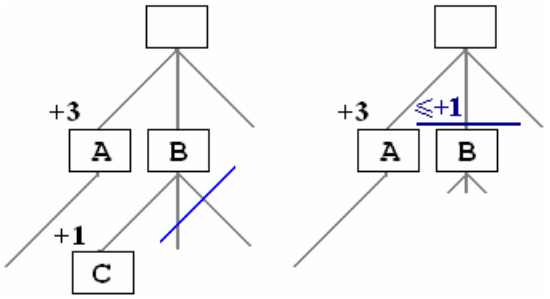


Рис. 1.2.2. Пример подрезки дерева.

Рассмотрим рис. 1.2.2. Пусть после обхода части дерева мы получили, что оценка (обобщённая) позиции A равна +3. Затем, после обхода другой части дерева, получили, что оценка позиции C равна +1. Отсюда сразу следует, что больше оценку позиции B можно не уточнять. Действительно, первый игрок имеет уже гарантированную оценку +3 (ход из начальной позиции, который через *r* полуходов приведёт к позиции, оценка которой не меньше +3). Оценка позиции B не больше +1, поскольку на этом уровне второй игрок минимизирует, а позиция C имеет оценку +1. Поэтому первый игрок при правильной игре не сделает ход в позицию B, и знание её точной оценки нам не интересно. Это называется **подрезкой дерева** или **отсечкой части дерева**. Аналогичная подрезка может возникать где-то в середине дерева, если второй игрок гарантировал себе некоторый результат.

Заметим, что если бы оценка позиции A равнялась нулю, то подрезки не произошло. Таким образом, чем больше оценка позиции A, тем потенциально больше подрезок может произойти (если оценка в позиции A равна +INF, то дальше дерево можно не смотреть, а сразу делать ход в позицию A, ведущий к победе).

Рассмотренная идея (подрезки дерева) реализуется в алгоритме альфа-бета. Псевдокод алгоритма приведён ниже.

Алфа-бета алгоритм

```
int Search (bool Color, int Depth, int alpha, int beta)
{
// если дошли до конца – оценить позицию
if (Depth<=0) return Evaluate(Color);
```

[1] На самом-то деле можем получить и хуже, если при каждом ходе строить дерево из r уровней. Дело в том, что когда мы будем делать второй полуход мы увидим «продолжение» нашего дерева, пометки по всему дереву изменятся. См. дальше «эффект горизонта».

```
// генерация ходов
Pmove move = GenerateAllMoves(Color);

while (move && alpha < beta)
      {
      MakeMove; // сделать ход (определено в define!)

      // должна быть проверка на окончание игры

      int tmp = - Search (!Color, Depth-1, -beta, -alpha);

      UnMakeMove; // вернуть позицию (определено в define!)

      if (tmp > alpha) alpha = tmp; // повышаем гарантию

      move = move->p; // следующий ход
      }

return alpha;
}
```

Чтобы не писать две функции (для минимизации и максимизации на разных уровнях), используется одна, которая вызывается «со знаком минус». При рекурсивном вызове меняется игрок (**Color**), глубина уменьшается на единицу (**Depth-1**). Когда глубина станет равной нулю, достигнута терминальная позиция, для которой вызывается функция оценки **Evaluate**. Код, естественно, нерабочий, поскольку должна быть проверка на терминальность и для вершин средних слоёв. Гарантируемый результат первого игрока – **alpha**, второго – **beta**. При рекурсивном вызове они меняются местами и знаком (функция всегда считает, что ходит первый игрок). Функция возвращает текущий максимальный гарантируемый результат первого игрока (**alpha**), который может повыситься в результате её работы (при вызове рекурсии). Сравнение **alpha < beta** осуществляет подрезку: если неравенство не выполняется, то ходы больше не рассматриваются и функция возвращает значение.

При наилучшем порядке ходов альфа-бета алгоритм просматривает число позиций, равное квадратному корню из числа позиций, которые просматриваются при полном переборе.

§1.3. Практические реализации альфа-бета алгоритма

1.3.1. Амортизация отказов.

На практике чаще используют альфа-бета алгоритм **с амортизацией отказов.** В этой версии алгоритма мы возвращаем не текущее значение **alpha**, а максимальное полученное значение для потомков этой позиции.

Альфа-бета алгоритм с амортизацией отказов

```
int AlphaBeta (bool Color, int Depth, int alpha, int beta)
{
if (Depth<=0) return Evaluate(Color);
int maxtmp = -INF; // пока ничего не получили
Pmove move = GenerateAllMoves(Color);
while (move && alpha < beta)
      {
      MakeMove;
      int tmp = - AlphaBeta (!Color, Depth-1, -beta, -alpha);
      UnMakeMove;
      if (tmp > maxtmp)
         {
         maxtmp = tmp; // текущий максимум
         if (maxtmp > alpha) alpha = maxtmp;
         }
      move = move->p;
      }
return maxtmp;
}
```

1.3.2. Перебор с нулевым окном.

Изначально наш алгоритм запускается вызовом функции **AlphaBeta** со значениями параметров **alpha**, **beta** соответственно **–INF**, **+INF** (**вызов с полным окном**). Если мы вызовем функцию со значениями параметров **a**, **b**, это вызовет «лишние» отсечки, а возвращаемое значение **g** не будет истинным **ist** (как при переборе с полным окном). Но будет выполняться замечательное свойство:

если **g<=alpha**, то **ist<=g**,
если **g>alpha**, то **ist>=g**.

ДЗ *Попробуйте обосновать это свойство. Придумайте, как ещё его можно использовать*[1] (*кроме перебора с нулевым окном, см. ниже*).

Перебор с нулевым окном – это перебор с параметрами **alpha**, **alpha+1**. Ясно, что, поскольку окно **[alpha, alpha+1]** «очень узкое», будет происходить много отсечек и такой перебор производится «очень быстро». Если в результате такого перебора мы получили оценку, которая не превосходит **alpha**, то надобности в полном переборе нет, поскольку оценка полного перебора (перебора с полным окном) также не улучшает гарантированный результат первого игрока **alpha**. Эта идея современного алгоритма **NegaScout**, который в два раза увеличивает скорость перебора дерева игры:

[1] Это свойство использует эвристика «Поиск стремления» (Aspiration search), см. [Корнилов, 2005].

1. Оценку первого позиции-потомка получаем «стандартно»: перебором с полным окном.
2. Прежде чем получить оценку следующего позиции-потомка вызовом функции с полным окном, вызываем функцию с нулевым окном.
3. Если не улучшается **alpha**, то с полным окном не перебираем.
4. Если **alpha** улучшается, т.е. **g>alpha** (где **g** – результат функции с нулевым окном), то перебираем с параметрами (**g, beta**).

Обратите внимание на п.4! Ниже выписан псевдокод алгоритма.

Алгоритм NegaScout

```
int NegaScout (bool Color, int Depth, int alpha, int beta)
{
int tmp;

if (Depth<=0) return Evaluate(Color);

Pmove move = GenerateAllMoves(Color);

if (move && (alpha < beta)) // первый потомок
   {
   MakeMove;
   // здесь должна быть ещё «проверка на мат»

   // полный проход
   tmp = -NegaScout(!color, Depth-1, -beta, -alpha);

   if (tmp > alpha) alpha = tmp;

   UnMakeMove;

   move = move->p;
   }

while (move && alpha < beta)
      {
      MakeMove;

      // здесь должна быть ещё «проверка на мат»

      // проход с нулевым окном
      tmp = -NegaScout(!Color, Depth-1, -(alpha+1), -alpha);
      if (tmp>alpha && tmp<beta)
         {
         // сокращение окна
         tmp = -NegaScout(!Color, Depth-1, -beta, -tmp);
         // см. на аргументы
         }

         }
```

```
      if (tmp > alpha) alpha = tmp;

      UnMakeMove;

      move = move->p;
      }

return alpha;
}
```

Итак, ключевая идея NegaScout – попытаться отсечь часть дерева без выполнения его полного перебора.

1.3.3. Порядок ходов.

Мы видели на рис. 1.2.2, что чем выше значение оценки позиции A, тем больше подрезок может произойти. Таким образом, нам желательно первым рассмотреть ход, который даст максимальную оценку позиции. Вообще-то именно такой ход мы и ищем. Так вот, мы должны его как можно раньше угадать, чтобы потом уже быстро пройтись по дереву с подрезками и убедиться, что он лучший (в алгоритме NegaScout это соответствует проходам с нулевым окном). Естественно, лучший ход мы должны угадывать в каждой вершине дерева (а не только в начальной). Поэтому очень существенно **в каком порядке перебираются ходы**. Считается, что при «умелой» генерации ходов за фиксированное время удаётся считать на глубину, в два раза превышающую глубину просчёта при рандомной генерации.

Во многих играх первыми рассматривают «самые опасные ходы»[1]. Ходы более сильных фигур рассматривают раньше. В шахматах рассматривают сначала взятия и шахи[2], а потом «обычные ходы». Причём взятия упорядочивают: пешка-ферзь, слон-ферзь, и т.д. (т.е. сначала **наиболее ценные жертвы и наименее ценные нападающие** – стратегия **MVV/LVA**). Если король находится «под шахом», то его ходы надо рассматривать первыми (часто это вызывает отсечку). При «спокойных позициях» ходы короля первыми не рассматриваются.

Часто **на одном уровне** в разных позициях **один и тот же ход может оказаться лучшим**, поэтому, перебирая дерево, для каждого уровня хранят ходы и их оценки или, по крайней мере, лучший ход с оценкой. Последняя тактика получила название **эвристика убийцы (killer heuristic)**. Применение хэш-таблиц решает многие проблемы, связанные с генерацией ходов (см. дальше).

---

[1] Часто именно при рассмотрении таких ходов возникает отсечка.

2 Определять в генераторе ходов, является ли ход шахом, очень накладно! Знаком ли Вам термин «вскрытый шах»?

ДЗ *Подумайте, почему при сортировке ходов чаще всего используют пузырьковую сортировку?*

ДЗ *При выполнении практикума на ЭВМ попробуйте сначала пройтись по дереву игры с небольшой глубиной, получить таким образом оценку ходов, а потом использовать её для упорядочивания.*

ДЗ *Подумайте над эвристикой первого хода «съесть последнюю ходившую фигуру соперника»* (*для некоторых игр она существенно повышает* **`alpha`**).

ДЗ *Подумайте над сортировкой простых ходов* (*не являются шахами и взятиями*). *Во многих играх функция* **`Evaluate`** *считается при спуске по дереву: каждый ход что-то прибавляет к текущей оценке позиции. Эту добавку можно использовать при оценке хода. Если в игре важно занять какие-то поля, то можно оценивать приближение «походившей» фишки к этим полям.*

1.3.4. Генерация ходов должна выполняться очень быстро (заметьте, что она выполняется в каждой вершине дерева перебора). Как правило, организуют список фигур. При генерации идут по списку своих фигур[1] и перебирают допустимые ходы каждой фигуры[2].

ДЗ *Подумайте об организации списка фигур. В шахматах желательно иметь отдельно список всех ферзей* (*каждого цвета*), *список всех ладей, и т.д. Это поможет эффективнее осуществлять стратегию MVV/LVA. Подумайте над обработкой «взятия фигуры»* (*можно удалять элемент списка, ставить флаг, использовать не список, а массив с переносов в конец удалённых фигур*).

Замечание. Иногда возникает соблазн не делать процедуру генерации ходов и не создавать список ходов, а генерировать очередной ход непосредственно перед рекурсивным вызовом функции. Это допустимо в играх, где все ходы априорно «примерно одинаковы по силе». Например, в игре ним нет шахов, взятий и т.д. (хотя даже в этой игре можно придумать эвристику сильного хода). В играх, где априорно все ходы «разные по силе», создание списка ходов необходимо, поскольку в процессе его создания ходы упорядочиваются. В профессиональных игровых программах очень быстро генерируется первый ход. Это делается для того, чтобы был шанс получить отсечку до генерации всех ходов.

---

[1] А не перебирают клетки доски…

[2] При съедении фигуры список изменяется (естественно, и на доске фигура удаляется). Можно использовать массивы с флагами присутствия фигур на доске.

1.3.5. Функция оценки позиции.

При прочих равных условиях, если «угаданы» оптимальные структуры данных, программисты разного уровня напишут программы, которые тратят на перебор на фиксированную глубину примерно одно и то же время. Поэтому всё решает функция оценки позиции.

ДЗ *Придумайте функцию оценки позиции в «крестиках-ноликах», которая при просчёте на два полухода позволяет делать оптимальные ходы.*

В шахматах один из простейших способов написания функции оценки позиции состоит в следующем. Материальную оценку, умноженную на некоторую константу, складывают со стратегической оценкой. Стратегическая оценка каждого игрока равна сумме стратегических оценок его фигур. Для каждой фигуры (короля, ферзя и т.д.) хранят таблицу размера 8×8. По сути, это пометки игрового поля. Каждая пометка – стратегическая оценка фигуры, когда она находится на этой клетке. Для короля такая таблица показана на рис. 1.3.1 (взято из [Корнилов, 2005]). Ясно, что в начале и середине партии королю следует находиться на своей первой горизонтали, лучше поближе к углу доски (куда его, как правило, предусмотрительно отправляют рокировкой). Поэтому значения для клеток первой горизонтали самые большие: от –10 до 0. Когда на доске съедены дальнобойные фигуры соперника, король должен быть активным: мешать пешкам соперника, помогать своим проходным пешкам. В конце партии король нередко становится фигурой, с помощью которой ставят мат («прикрывает» матующую фигуру или отрезает пути отступления королю соперника). Поэтому в конце партии (иногда в её середине) следует использовать другую таблицу для стратегической оценки короля. **ДЗ** *Какую?*

```
  0   0  -4 -10 -10  -4   0   0
 -4  -4  -8 -12 -12  -8  -4  -4
-12 -16 -20 -20 -20 -20 -20 -12
-16 -20 -24 -24 -24 -24 -20 -16
-16 -20 -24 -24 -24 -24 -20 -16
-12 -16 -20 -20 -20 -20 -20 -12
 -4  -4  -8 -12 -12  -8  -4  -4
  0   0  -4 -10 -10  -4   0   0
```

Рис. 1.3.1. Стратегическая позиция для короля (в начале партии).

Мы не будем останавливаться на том, как выбрать такие таблицы (их можно задавать экспертно, а можно настраивать по результатам работы программы) и как менять таблицы для фиксированной фигуры в течение партии. Отметим, что в функцию оценки позиции можно добавить ещё много слагаемых (см. §1.2), которые (в шахматах) учитывают сдвоенность ладей, сдвоенность пешек, наличие фигур (а главное, пешек) около короля и т.д.

1.3.6. Быстрые победы

На самом деле, победе первого игрока надо сопоставлять не значение **`+INF`**, а значение **`+INF-i`**, где **`i`** – номер полухода, на котором достигается победа. Аналогично, победе второго игрока – значение **`-INF+i`**. Это связано с тем, что игрок, который делает ход, если выигрывает, то должен стараться выиграть как можно быстрее. Если он проигрывает, то должен стараться оттянуть своё поражение (вдруг у соперника кончится время на обдумывание ходов).

1.3.7. Эффект горизонта

Недостаток перебора на *r* полуходов в том, что он хуже перебора на *r* + 1 полуход... а там, на (*r* + 1)-м ходе, может быть самое интересное. Например, мы можем не заметить, что выигрываем на (*r* + 1)-м ходе, сделать «неверный» ход и уже лишиться возможности выиграть[1]. Или считать, что выигрываем в результате «длинного размена» (съедаем пешкой пешку, её ест конь соперника, которого мы съедаем слоном, его ест слон соперника, мы бьём его ладьёй), а на самом деле проигрываем (следующим ходом съедают нашу ладью). Поэтому для «самых интересных» позиций и «самых интересных» последних ходов считают глубже по дереву – ещё на несколько уровней вперёд. Обычно «продлевают» **форсированные ходы** (в шахматах – взятия фигур, шахи, приближение пешки к последней горизонтали), т.е. ходы, которые существенно изменили положение дел на доске: кто-то получил материальный перевес, значительно ограничил свободу фигур соперника или увеличил свободу своих фигур, сильно изменил оценку позиции (**`Evaluate`**).

Здесь полезно заучить правило программирования подобных игр: **любой размен надо просмотреть до конца!**

Ещё одним способом борьбы с эффектом горизонта является дробная глубина просчёта. Каждый ход уменьшает глубину **`Depth`** на некоторое число в зависимости от его опасности (например, стандартный ход – на 4, взятие – на 2, ход пешки на предпоследнюю горизонталь – на 2, шах – на 1).

1.3.8. Хэш

При программировании логических игр использование хэш-таблиц может существенно ускорить счёт. Например, одна и та же позиция может получаться после разных последовательностей полуходов. Если мы уже один раз в дереве вычислили для неё оценку, то не имеет смысла вычислять ещё раз. Поэтому имеет смысл хранить для каждой позиции получившиеся оценки. Естественно, поскольку позиций очень много (попробуйте посчитать, сколько примерно различных позиций в шахматах), то для этого используются хэш-таблицы.

---

[1] Это и называется «эффектом горизонта». Мы не видим, что за горизонтом, поэтому можем выбрать неверное направление движения.

Здесь есть маленький подвох! Оценку позиции можно не считать, если мы уже вычислили оценку этой позиции **на том же уровне дерева**.

ДЗ *Попробуйте понять, почему нельзя использовать оценку позиции другого уровня.*

1.3.9. Нулевой ход

Во многих играх часто (но не всегда!) если один из игроков ходит два раза подряд, то получает существенное преимущество. Это используется в эвристике «**нулевой ход**» для обрезки дерева. Заметим, что когда в alpha-beta алгоритме мы получаем значение больше или равно beta (гарантированный результат второго игрока), счёт прекращается. Допустим, второй игрок сходил два раза подряд (мы пропустили ход), и оценка позиции стала больше или равна beta. Тогда, если бы мы делали ход, то оценка была бы не меньше (это гипотеза!), поэтому счёт можно прекратить… Это отражено в следующем фрагменте кода.

```
int AlphaBeta (bool Color, int Depth, int alpha, int beta, bool
doNullMove)
{
if (Depth<=0) return Evaluate(Color);

// нулевой ход
// здесь должна быть куча if-ов (не шах, не взятие и т.д.)
if (!doNullMove && (-AlphaBeta (!Color, Depth-1-R, -beta, -alpha,
true)>=beta))
   return beta;

// дальше стандартно... Pmove move = ...
//  с вызовом -AlphaBeta (!Color, Depth-1, -beta, -alpha,
doNullMove);
```

Здесь **R** – коэффициент затухания, который вводится, чтобы при пропуске хода не считать «до конца». Если мы пропустили ход, а позиция «не очень испортилась», то можно не перебирать дерево стандартным полным перебором. Нулевой ход при переборе используют только один раз, т.е. если мы пропустили ход и вызвали рекурсивно alpha-beta-алгоритм, то при рекурсивном переборе больше ходов не пропускаем (см. в коде флаг **doNullMove**). Нулевой ход не используют на маленькой глубине (если значение **Depth** достаточно большое), после шаха, взятия, при расширениях (например при разборе форсированных ходов), если текущая оценка позиции меньше чем **beta**. **ДЗ** *Почему?*

Внимание! Во многих играх эвристика «нулевой ход» не даёт положительного результата (например в шашках).

ДЗ *Попробуйте перебирать дерево игры так, что на двух последних уровнях дерева два раза ходит соперник (иногда программа, которая реализует эту идею, показывает очень занятную тактику).*

Вообще, многие эвристики (Futility pruning, Razoring) используют идею «если текущая позиция достаточно хорошая, то перебор надо закончить». Часто это выражается следующими строками кода:

```
if (Evaluate(Color) >= beta + margin)
   return beta; // или Depth--; для простого сокращения перебора
```

Заметим, что, как правило, такие эвристики применяют на последних уровнях дерева перебора.

1.3.8. Учёт специфики правил

Во многих играх, если серия ходов повторяется три раза подряд, то игра заканчивается вничью, т.е. не всё определяется текущим положением фигур на доске. Поэтому необходимо учитывать «историю ходов»[1].

Как отлаживать программу

Лучше создать несколько разных позиций, которые соответствуют различным этапам игры: начало (позиция симметричная, «ровная»), середина игры (есть «разброс» фигур, кто-то, быть может, имеет преимущество), конец игры (хорошо бы подгадать позицию, в которой игра заканчивается через несколько ходов). Свою программу следует отлаживать на всех этих позициях. Например, вы увеличили глубину перебора. По идее, программа обязана играть лучше из любой позиции (вы запускаете игру «новая версия» – «старая версия»). «Новая версия» может даже начать видеть окончание игры в несколько ходов. Кстати, неплохой метод – найти позицию «выигрыш в N ходов», уменьшить глубину перебора и подстраивать теперь функцию оценки, чтобы программа выигрывала за N ходов, т.е. чтобы, несмотря на эффект горизонта, программа «шла в нужном направлении» (при этом отключают продление форсированных ходов).

Учитывайте также, что процедура, которая даёт хороший порядок ходов в одной позиции, может давать плохой порядок в другой (кстати, в зависимости от позиции можно запускать разные процедуры упорядочивания ходов).

Про программирование логических игр можно почитать в [Корнилов, 2005], [Люгер, 2005]. Много информации есть в Интернете (попробуйте набрать в поисковике запрос «NegaScout»).

---

[1] Студенты кафедры ММП ф-та ВМК МГУ в задании по практикуму должны написать программу, которая по текущей позиции делает ход: считывает позицию из файла и заменяет файл на новый, соответствующий позиции после хода программы. Таким образом, «история ходов» не может быть учтена.

Глава 2. ЗАДАЧА ОБУЧЕНИЯ ПО ПРЕЦЕДЕНТАМ

§2.1. Постановка задачи

Задача обучения по прецедентам (задача машинного обучения) заключается в следующем. Даны два пространства: X (пространство **допустимых объектов**), Y (пространство **ответов** или **меток**) и (**целевая**) функция $y: X \to Y$, которая задана лишь в конечном множестве точек (**обучающей выборке**, **прецедентах**):

$$y(x^1),\ldots,y(x^m),$$

т.е. **известны метки объектов** $x^1,\ldots,x^m$. Требуется построить алгоритм A (**обучить алгоритм**), который по объекту x определяет значение $y(x)$ (или «достаточно близкое» значение, если допускается неточное решение). Таким образом, это **задача экстраполяции функции**, но с условиями существования алгоритмического решения[1], а также с некоторой спецификой задания объектов, которую мы обсудим ниже.

При конечном множестве Y, пусть $Y=\{1,2,\ldots,l\}$, задачу называют **задачей классификацией (на l непересекающихся классов).** В этом случае говорят, что **множество X разбито на классы** $K_1,\ldots,K_l$, где $K_i=\{x\in X \mid y(x)=i\}$ при $i\in\{1,2,\ldots,l\}$:

$$X=\bigcup_{i=1}^{l} K_i\ .$$

При $Y=\{(\alpha_1,\ldots,\alpha_l)\mid \alpha_1,\ldots,\alpha_l\in\{0,1\}\}$ говорят о **задаче классификации на l пересекающихся классов.** Здесь i-й класс –

$$K_i=\{x\in X \mid y(x)=(\alpha_1,\ldots,\alpha_l),\ \alpha_i=1\}.$$

Например, задача классификации электронных писем на спам / не спам – задача классификации с двумя непересекающимися классами. Задача классификации пациентов по историям болезни в группу «здоровы (не нуждаются больше в лечении)», «больны в слабой форме (необходим стационарный режим)» и «серьёзно больны (необходима операция)» – задача классификации с тремя непересекающимися классами. Задача классификации абонентов оператора сотовой сети на класс, представителям которого стоит предложить услугу «Мелодия вместо гудков» (скорее всего они воспользуются услугой и не сочтут предложение спамом), и класс, представителям которого следует предложить поменять тарифный план, является задачей с двумя пересекающимися классами. В ней объекты описываются своими характеристиками (данными анкеты) и протоколами разговоров, пользования дополнительными услугами и изменения счёта.

[1] С учётом прикладных нужд – простого и быстрого алгоритма.

Замечание. Задачу классификации можно свести к нескольким задачам с двумя непересекающимися классами, поэтому такие задачи и будут в основном изучаться (на них ориентировано большинство моделей алгоритмов).

Замечание. Ещё одно важное отличие задачи обучения от задачи экстраполяции состоит в неформальном предположении, что известных данных хватит для приемлемого восстановления функции $y : X \to Y$. Задачу «по имени и фамилии человека восстановить его телефонный номер, обучившись на данных из конкретной записной книжки» на практике не решают (хотя формально её постановка не противоречит постановке задачи обучения).

При континуальном[1] множестве $Y \subseteq \mathbf{R}$ задачу обучения называют **задачей восстановления регрессии**. Например, оценить время окупаемости проекта по его описанию, состоянию рынка и прогнозам специалистов. Очень часто говорят о задаче **прогнозирования**, если обучающая выборка – это значения некоторого процесса в прошлом и необходимо предсказать его значение в будущем. Например, спрогнозировать цену акции на следующей неделе по временному ряду её цены (или многомерному ряду цен группы акций) за предыдущий период.

Часто в задаче определена или может быть легко введена **функция потерь** $L(A(x), y(x))$, которая описывает, на сколько «плох» наш ответ $A(x)$ при верном ответе $y(x)$. В задаче классификации часто

$$L(A(x), y(x)) = \begin{cases} 1, & A(x) \neq y(x), \\ 0, & A(x) = y(x), \end{cases}$$

в задаче регрессии –

$$L(A(x), y(x)) = |A(x) - y(x)|$$

или

$$L(A(x), y(x)) = (A(x) - y(x))^2.$$

Вообще, задача обучения по прецедентам это ещё и **задача оптимизации**, поскольку её решают в виде

$$\frac{1}{m}\sum_{t=1}^{m} L(A(x^t), y(x^t)) \xrightarrow[A]{} \min$$

– **минимизируют эмпирический риск**, где алгоритм A выбирают из некоторого семейства (собственно, в этом учебном пособии речь о таких семействах и пойдёт).

Задачу обучения в «умных книжках» ставят так:

$$\iint_{X \times Y} L(A(x), y)\, p(x, y)\, dxdy \to \min,$$

[1] Ясно, что на практике приходится иметь дело лишь с конечным подмножеством этого множества (более того, с числами, которые представимы в ЭВМ).

интегрируя функцию потерь по всему пространству $X \times Y$ пар «объект-метка», на котором эти пары распределены с плотностью $p(x, y)$. Таким образом, это задача минимизации матожидания функции потерь (**среднего риска**). Конечно, подобный интеграл практически вычислить невозможно (мы ещё поговорим об этом), поэтому его приближают (например, заменяют эмпирическим риском) и решают «похожую» задачу оптимизации[1]. При такой постановке также возникает философский вопрос о существовании плотности $p(x, y)$. Кроме того, наш алгоритм никогда не будет работать на континуальном множестве $X \times Y$. Например, нам нужен спам-фильтр, который будет правильно классифицировать только те письма, которые попадут в наш ящик, а не всевозможные письма на свете[2]. Таким образом, правильнее говорить о том, что наш алгоритм должен минимизировать функционал

$$\frac{1}{q}\sum_{t=1}^{q} L(A(x_t), y(x_t)) \qquad (2.1)$$

на выборке $\{x_t\}_{t=1}^{q}$, которая в момент построения (ещё говорят **настройки, обучения**) алгоритма может быть неизвестна, но на которой он потом будет эксплуатироваться. Это требование называют **обобщающей способностью** алгоритма. Нам не нужен алгоритм, который просто запомнит пары $(x^1, y(x^1)),\ldots,(x^m, y(x^m))$, такой алгоритм называется **переобученным** (термин, возможно, не совсем удачный, но он подчёркивает, что алгоритм «сильно» заучил обучающую выборку и ничего больше не знает).

Для борьбы с переобучением прежде всего организуют **контроль качества алгоритма**[3]. Для этого ошибки алгоритма оценивают одним из следующих способов

1. **Ошибка на контрольной выборке**. Множество объектов с известным значением функции y разбивают на обучающую выборку[4] $\{x^t\}_{t=1}^{m}$, на которой настраивают (обучают) алгоритм A, и **контрольную выборку** $\{x_t\}_{t=1}^{q}$, на которой проверяют качество работы (2.1) алгоритма A, т.е. «моделируют» ту выборку, на которой алгоритму придётся работать.

2. **Ошибка скользящего контроля**. Для некоторого натурального k проводят всевозможные разбиения выборки на две части: с k и $m-k$ объектами. Первую выборку считают контрольной, вторую – обучающей.

---

[1] Некоторые теории посвящены анализу корректности такого приближения (замены), но их результаты в данное учебное пособие не вошли.

[2] Недостаток многих современных теорий в том, что они «достаточно универсальны» и пытаются охватить все возможные случаи, а на практике приходится иметь дело с конкретной задачей и конкретной выборкой.

[3] Правильная организация контроля качества необходима для успешного выполнения заданий практикума на ЭВМ для студентов каф. ММП ф-та ВМК МГУ.

[4] Используем такое же обозначение, которое изначально использовалось для множества всех объектов с известным значением функции y.

Значения (2.1) для контрольной выборки усредняют по всем разбиениям. Чаще всего это делают для $k=1$ (метод **leave-one-out**).

3. **Контроль по блокам (фолдам)**. Выборку разбивают на k блоков (их иногда называют «фолдами»). Проводят k экспериментов, используя в i-м i-й блок в качестве контрольной выборки, а объединение остальных блоков – в качестве обучающей выборки. Значения функции (2.1) усредняют по числу экспериментов.

4. **Контроль на случайной подвыборке.** Случайно выбирают часть выборки в качестве контрольной, а остальную часть назначают обучающей. Проводят серию экспериментов. Усредняют результаты (значения (2.1)) по числу экспериментов. Часто разбиение делают методом **бутстреп**, который проще пояснить на примере.

Пример (бутстреп). Пусть в выборке 10 объектов. Генерируем 10 случайных величин, равномерно распределённых на множестве {1,2,…,10} (в системе MatLab это делается командой **`u = ceil(10*rand([1 10]))`**), пусть сгенерирован вектор значений **`u = [5, 4, 8, 8, 2, 5, 5, 7, 8, 8].`** Объекты с этими номерами (**`unique(u)`**) заносим в обучение, а остальные объекты – в контроль (с номерами **`setdiff(1:10, u)`**, т.е. с номерами из **`[1, 3, 6, 9, 10]`**).

Замечание. При разбиении выборки на **обучение** и **контроль** главное сохранить «пропорцию классов». Если в исходной выборке 900 объектов первого класса и 100 объектов второго класса, то «странно» выделять для контроля 100 объектов, в которых 50 объектов первого класса и 50 второго. Кроме того, в таких задачах[1], где числа представителей классов сильно различаются, по-особому бывает устроена **функция ошибки:**

$$\sum_{k=1}^{2}\frac{1}{|\{t\,|\,y(x_t)=k\}|}\sum_{t:y(x_t)=k}L(A(x_t),y(x_t)).$$

Вообще, очень часто важно правильно классифицировать объекты малых классов! Например, в задачах скоринга по анкете клиента банка надо определить, вернёт ли он кредит. Если банк вёл «достаточно грамотную» работу, то невозвратов кредитов у него мало, но он и не хочет, чтобы они появились. Поэтому надо, чтобы на небольшом классе клиентов, которые не возвращают кредит, ошибка была как можно ниже. Вообще, есть задачи **идентификации**, в которых, в некотором смысле, только один класс. Например, необходимо обучить устройство отличать вашу настоящую подпись от поддельной. Конечно, можно нагенерировать объекты класса «поддельных подписей», но такая выборка не будет **предствительной**, поскольку вряд ли

[1] Если в задаче с двумя непересекающимися классами объекты второго класса встречаются очень редко, то алгоритм, который всё относит к первому классу, будет допускать мало ошибок классификации. Ясно, что не такой алгоритм хотелось бы построить…

такие подписи будут похожи на подписи людей, которые захотят подделать Вашу с какой-то корыстной целью…

Замечание. Кроме организации контроля качества алгоритма есть много других способов борьбы с переобучением. Контроль качества нужен для выбора параметров алгоритма, которые нельзя настроить в процессе обучения (иногда их называют **структурными параметрами**). Например, мы хотим спрогнозировать временной ряд, представив его в виде линейной комбинации базисных функций. Коэффициенты в линейной комбинации выберем так, чтобы минимизировать эмпирический риск, т.е. в процессе обучения (мы потом увидим, что для них есть достаточно простые формулы). А вот как выбрать базисные функции? Если мы хорошо знаем «физику задачи», то можем воспользоваться некоторыми принятыми рекомендациями или «угадать» эти функции. Если нет – можно попробовать разные базисы и выбрать тот, при котором ошибка на контроле минимальна. Есть общая рекомендация, которая следует из «умных теорий», что **модель алгоритмов** (параметризованное семейство, в котором мы ищем наш алгоритм) должна быть достаточно простой. Верен ли этот тезис и как его понимать, лучше узнать в процессе экспериментов на модельных и реальных данных.

§2.2. Задание объектов

До сих пор ничего не было сказано о том, как заданы наши объекты пространства X. Как правило, рассматривают следующие пространства объектов:

1. **Признаковые пространства**. Это пространства вида $F_1 \times \ldots \times F_n$. Каждый объект x описывается вектором $(f_1(x),\ldots, f_n(x))$ из этого пространства (**вектором признаков, признаковым описанием**). Функция $f_t : X \to F_t$ называется t**-м признаком**, $t \in \{1,2,\ldots,n\}$, n – **числом признаков** (**размерностью признакового пространства**). Например, 40-летний пациент-мужчина с температурой 37.2°C и 2-й группой крови может описываться вектором **`[40, 1, 37.2, 2, ...]`**. Здесь значение второго признака равно 1, если пациент мужчина, и равно 0, если женщина. Такой признак называется **2-значным** или **бинарным** (принимает 2 значения). Четвёртый признак называется **4-значным** (принимает 4 значения), третий – **вещественным**, первый – **целочисленным** (все названия описывают область значений признака). k-значные признаки могут быть **упорядоченными** (например «степень ожога»), когда на множестве значений введён порядок, имеющий содержательный смысл, или **номинальными** (например «группа крови», «национальность», «профессия»), когда значение признака имеет смысл принадлежности какой-то группе. Значения некоторых признаков может быть неопределенно (**пропуски данных**). Кроме того, значения некоторого признака могут быть случайны, не коррелировать со значениями функции y и не иметь ничего общего с физикой задачи, такие признаки называются **шумовыми**.

Значения признаков на отдельных объектах могут определяться неверно (температура измеряется с некоторой погрешностью, в графу «пол» по ошибке могли занести неверное значение). Такое явление называется **шумом** (говорят, что значения признака **зашумлены**). В принципе, функция $y(x)$ по своей природе тоже является признаком, поэтому часто её называют **целевым признаком**. Шумы в целевом признаке называются **выбросами**[1] (например, пациенту был неверно поставлен диагноз специалистом, или в задаче фильтрации спама мы по ошибке пометили какое-то письмо как «спам» и оно попало в обучающую выборку). Из-за подобных явлений выборка может стать **противоречивой**: похожие (или вообще одинаковые объекты) классифицированы по-разному (имеют разные значения функции y).

2. **Метрические пространства (и задание с помощью отношений).** Пространство X может быть метрическим, и нам могут быть известны значения метрики $\rho(x, x')$ (т.е. для любой пары объектов x, x' мы можем вычислить $\rho(x, x')$). Часто известна не метрика, а какая-то функция сходства (аксиома треугольника может не выполняться, функция может не быть симметричной и т.д.) или значения нескольких функций $\rho_\Omega(x, x')$, $\Omega \in \Omega^*$. Во многих задачах такие функции легко придумать. Например, если допустимые объекты – сайты Интернета, то можно придумать эвристики оценки сходства заголовков HTML-страниц, похожесть контента, число общих ссылок и т.д. Здесь также есть **шумы**, которые выражаются в ошибках измерения функций $\rho_\Omega(x, x')$ (например, в задаче классификации химических соединений $\rho_\Omega(x, x')$ может быть равно времени химической реакции между соединениями x и x', которое измеряется неточно).

Замечание. Значения $\rho_\Omega(x, x')$ также могут описывать отношение. В частности, принадлежность одному классу некоторой эквивалентности (например вхождение в одну группу по интересам пользователей социальной сети).

3. **Пространство измерений (сигналы).** Рассмотрим задачу классификации сигналов. Каждый объект задаётся вещественным вектором, который описывает показания датчика через равные промежутки времени[2]. Векторы могут быть разной длины. Задача может состоять, например, в отнесении сигнала к классу «нормальная работа механизма» или «есть неполадка» (т.е. это задача технической диагностики по показаниям датчиков). Объекты заданы не признаковым описанием, поскольку снимать показания датчиков начинали в разные моменты времени: после запуска механизма, чуть

[1] Часто объект со значительным изменением какого-то признака (большим зашумлением) называют выбросом.

[2] При описании показания датчика через разные промежутки времени используется вектор пар. Первый элемент пары описывает время показания, а второй – значение.

позже и т.д., т.е. первый элемент вектора не является первым измерением после «подозрения, что произошла поломка». Свойство, которое количественно выражает факт неисправной работы, должно быть инвариантно относительно небольших временных сдвигов (например, относительно удаления первого элемента вектора). В виде измерений хранится бОльшая часть данных о клиентах компаний, предоставляющих какие-то услуги (например время и продолжительность звонков клиента оператора сотовой связи; отметим, что «число совершённых звонков» – это уже значение признака).

Иногда изображения (в простейшем случае, бинарные прямоугольные матрицы) называют **2D-измерениями** (**2D-сигналами**). Здесь тоже есть инвариантность относительно параллельных переносов и небольших искажений (например, Вы можете узнать себя на фотографии независимо от того, находится ли Ваше лицо в центре снимка или нет, и даже при неудачной фокусировке). Есть также 3D-измерения: **сцены**.

Замечание. От одного способа задания объекта можно перейти к другому (см. табл.). Переходят всегда к признаковому или метрическому описанию, поскольку на такие описания ориентированы основные модели алгоритмов. Например, для перехода от метрического пространства к признаковому ищется такая система m точек в $\mathbf{R}^n$ для некоторого n, что значения евклидовой или какой-то другой метрики на парах этих точек совпадает со значениями исходной метрики на соответствующих парах наших объектов.

	Переход к признакам	**Переход к метрике**
Признаки		Введение метрики в признаковом пространстве.
Метрика	Вложение в евклидово пространство при сохранении метрики. Трактовка расстояний до фиксированных точек как значений признаков.	
Измерения	Введение признаков.	Введение метрики на измерениях.

Замечание. В системе MatLab признаковые данные часто задаются в виде двух матриц: **`X`** размера **`m×n`** и **`Y`** размера **`m×1.`** В первой матрице по строкам записаны признаковые описания объектов, она называется **признаковой матрицей** или матрицей **объект-признак**. Неизвестные значения признаков равны **`NaN`**. В векторе **`Y`** записаны метки объектов.

Замечание. Есть и более сложные способы задания объектов: в виде сложной **структуры.** Например, Интернет-страница задаётся заголовком, ключевыми словами, а также своим контентом и его расположением (что и где находится на странице, это может быть описано на специальном формальном языке), описанием «механизмов» такого расположения (HTML, JavaScript и т.д.), ссылками на эту страницу и т.д. Отметим, что в понятие «контент» входят, например, изображения на Интернет-странице. Даже описать одно такое изображение не всегда тривиально. Можно упомянуть также задачу классификации XML-документов.

Подробнее о задачах обучения по прецедентам можно почитать в [Воронцов], см. также [Золотых].

§2.3. Разделяющие поверхности

Каждый классификатор (алгоритм классификации) реализует отображение $A: X \to Y$ в конечное множество Y. Множества классов «с точки зрения классификатора»

$$\{x \in X \mid A(x) = y\},\ y \in Y,$$

как правило, «достаточно просто устроены»: имеют небольшое число компонент связности, часто выпуклы, имеют гладкую границу и т.д. Границу между этими множествами и называют **разделяющей поверхностью**. Если рассмотреть простой случай, когда в задаче с двумя непересекающимися классами объекты заданы в двумерном признаковом пространстве, то это кривая на плоскости, которая отделяет один класс от другого. Ниже мы покажем, как она выглядит для различных алгоритмов классификации.

Замечание. Часто алгоритм классификации в явном виде задаётся параметрической разделяющей поверхностью, а обучение алгоритма сводится к настройке параметров этой поверхности так, чтобы минимизировать некоторый функционал (например, число ошибок на обучающей выборке). Необходимо иметь представление о том, как выглядят разделяющие поверхности для основных алгоритмов классификации, чтобы выработать «геометрическую интуицию» решения задач классификации.

Все приведённые ниже иллюстрации получены в системе MatLab[1] с помощью пакета Clop/Spider (см. [Clop], по сути, будет использован «более старый», но удобный пакет Spider, который входит в состав пакета Clop) и сопровождаются необходимым MatLab-кодом.

[1] Сама система подробно рассмотрена в главе «Среда для вычислений и визуализации MatLab» (в зависимости от версии учебного пособия глава находится в конце книги или в её второй части).

Для начала создадим обучающую выборку. Здесь важно, что $Y = \{+1, -1\}$, поскольку многие алгоритмы (например байесовский классификатор) в рассматриваемом пакете работают лишь при таком кодировании классов, в противном случае они «вылетают с ошибкой». **Совет:** при решении задач в пакетах прикладных программ пробуйте разные способы кодирования классов, часто алгоритмы (без специальных оговорок) ориентированы только на задачи с двумя непересекающимися классами и $Y = \{+1, -1\}$.

```
%% Синтез данных
% Создание обучающей выборки
DataS = rand([40 2]); % случайные точки в ед. квадрате
% 1 класс - внутренность круга
Y = ((DataS(:,1).^2 + DataS(:,2).^2)<0.5);
Y(1:5) = ~Y(1:5); % добавляем выбросы
Y = 2*Y - 1; % переходим к нумерующему вектору с метками -1, +1

% Создание контрольной выборки
[a b] = meshgrid(0:0.01:1, 0:0.01:1);
DataC = [a(:), b(:)]; % мелкая сетка
```

Поскольку ни в самой системе MatLab, ни в пакете Clop/Spider нет средств визуализации разделяющей поверхности, мы сгенерируем точки мелкой сетки и закрасим объекты одного из классов.

Код, который обучает алгоритм (в данном случае метод SVM со специальным выбором параметров), выглядит следующим образом:

```
training_data = data(DataS, Y); % данные для обучения
hyperparameters = {'coef0=1', 'degree=1', 'gamma=1'}; % параметры
untrained_model = svc(hyperparameters); % модель (алгоритмов)
[training_data_output, trained_model] = ...
                  train(untrained_model, training_data); % обучить
test_data = data(DataC); % данные для тестирования
Yc =  test(trained_model, test_data); % тестировать
```

Вывод картинок осуществляется с помощью таких команд:

```
clf;
scatter(DataC(Yc.X>0,1),DataC(Yc.X>0,2),10,'*r'); % раздел. пов-ть
hold on;
% (обучение)
scatter(DataS(Y<0,1),DataS(Y<0,2),25,'ks','filled'); % один класс
scatter(DataS(Y>0,1),DataS(Y>0,2),35,'w','filled'); % второй класс
scatter(DataS(Y>0,1),DataS(Y>0,2),40,'b','LineWidth',2);
```

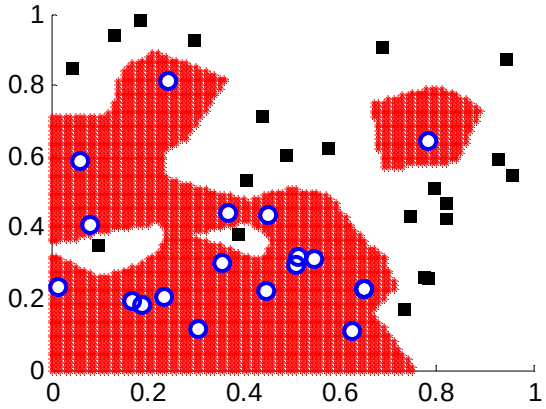


Рис. 2.3.1. Разделяющая поверхность метода ближайшего соседа[1].

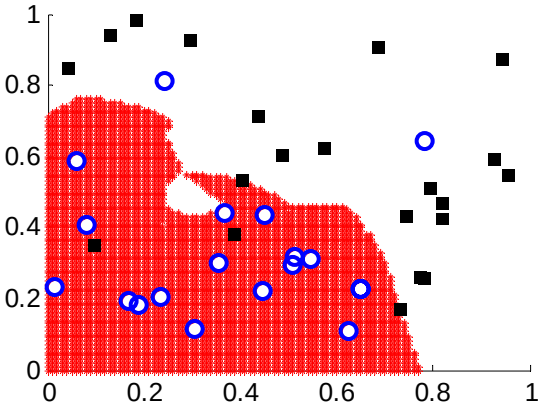


Рис. 2.3.2. Разделяющая поверхность метода 3-х ближайших соседей.

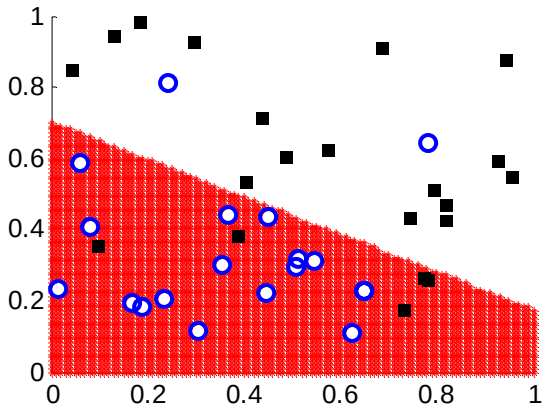


Рис. 2.3.3. Разделяющая поверхность наивного байесовского классификатора[2].

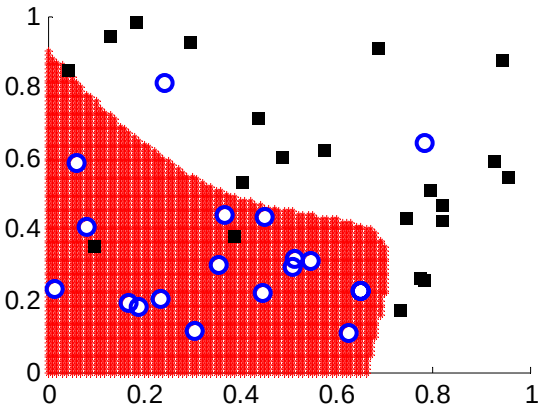


Рис. 2.3.4. Разделяющая поверхность нейросети с 5-ю нейронами.

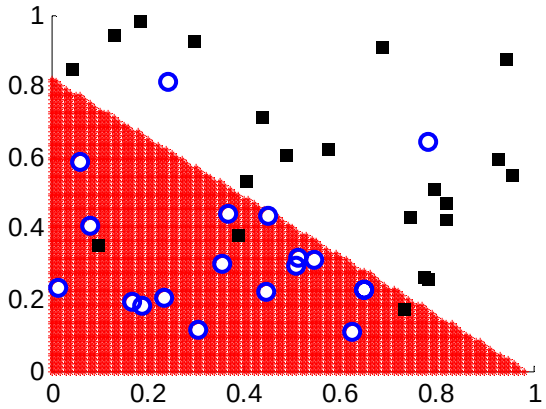


Рис. 2.3.5. Разделяющая поверхность нейросети с 3-я нейронами.

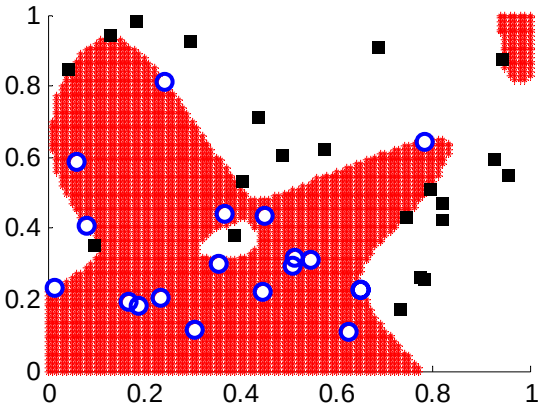


Рис. 2.3.6. Разделяющая поверхность метода SVM с ядром.

---

[1] Об этой и других моделях алгоритмов классификации подробно написано в соответствующих главах.

[2] Для получения адекватной картинки пришлось чуть изменить ответы классификатора. Вывод осуществлялся командой

```
scatter(DataC(Yc.X>1.8, 1), DataC(Yc.X>1.8, 2), 10, '*r');
```

ДЗ *При чтении следующих глав попробуйте поэкспериментировать с алгоритмами, реализованными в пакетах системы MatLab. В частности, сделайте визуализацию разделяющих поверхностей при различных значениях параметров алгоритма. Установите, при каких значениях параметров алгоритмов получены приведённые рисунки* (*например, какая функция активация у нейронов сети на рис.* 2.3.4)?

Рис. 2.3.1 – 2.3.6 представляют популярную иллюстрацию задачи классификации: два класса на плоскости. Обычно представителей одного класса рисуют крестиками, а второго – ноликами. Неформально говоря, задача классификации состоит в том, чтобы отделить крестики от ноликов. Теперь рассмотрим простейшую иллюстрацию задачи восстановления регрессии. Пусть $X = Y = \mathbf{R}$. Таким образом, задача состоит в том, чтобы по конечному набору точек графика функции на плоскости восстановить весь график.

Ниже показан MatLab-код, который генерирует данные, обучает нейронную сеть в пакете Clop/Spider и выводит график, см. рис. 2.3.7, 2.3.8.

```
% Обучающая выборка (синус)
X = 10*rand([100 1]); Y = sin(X);
training_data = data(X, Y);
% Модель - нейросеть
untrained_model = neural({'units=3'}); % 3 нейрона!
[training_data_output, trained_model] = ...
                          train(untrained_model, training_data);
% Контрольная выборка - мелкая сетка
Xc = [-1:0.02:11]';
Yc =  test(trained_model, data(Xc));
% Вывод на экран
plot(Xc,Yc.X,'k')
hold on;
scatter(X,Y,20,'filled')
```

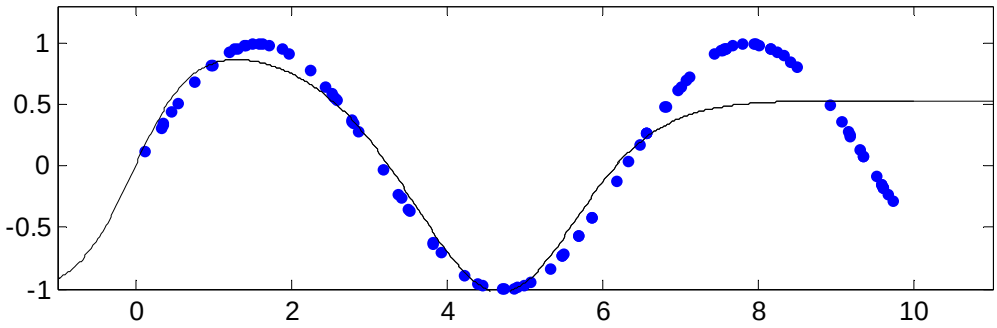


Рис. 2.3.7. Восстановление регрессии нейросетью с тремя нейронами.

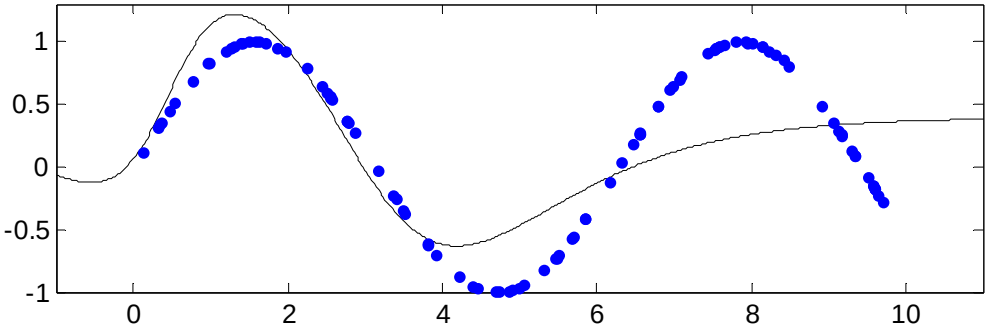


Рис. 2.3.8. Восстановление регрессии нейросетью с пятью нейронами.

§2.4. Оценка семейства алгоритмов: ROC-кривая

В §2.1 были перечислены способы контроля качества алгоритмов. Рассмотрим также один способ оценки качества целого семейства алгоритмов классификации. По сути, это простое усреднение качества по всем алгоритмам семейства, но оно часто встречается на практике, допускает интересную визуализацию и может применяться в некоторых других случаях (например, при оценке качества признаков).

Рассмотрим задачу с двумя непересекающимися классами: $Y = \{+1, -1\}$. Пусть наш алгоритм имеет вид

$$A_c(x) = \begin{cases} +1, & g(x) \geq c, \\ -1, & g(x) < c, \end{cases}$$

где $g(x): X \to \mathbf{R}$ – некоторая функция, а c – параметр алгоритма (у функции могут быть свои настраиваемые параметры). Как будет показано далее, многие современные алгоритмы классификации имеют такой вид. Значение функции g оценивает «степень уверенности» алгоритма: чем оно больше, тем увереннее алгоритм относит объект к первому классу, а чем меньше, тем увереннее относит ко второму. Поэтому значение функции естественно назвать **оценкой принадлежности объекта (первому) классу** [Журавлёв, 1978]. Каждый алгоритм можно оценить количеством ошибок на обучающей выборке. А как оценить семейство алгоритмов, скажем, параметризованное с помощью c: $\{A_c(x)\}_{c \in \mathbf{R}}$? Рассмотрим один из стандартных способов на примере.

Пусть значения $g(x)$ для контрольных объектов равны 3, 4, 7, 8, 9, 13, 16 (мы специально упорядочили их по возрастанию, хотя, как будет видно далее, конкретные значения не важны), а классификации этих объектов соответственно равны –1, +1, –1, –1, +1, +1, +1. Поместим точку в начале декартовой системы координат на плоскости. Будем, просматривая этот ряд классификаций, смещать её вверх, если в ряде стоит –1, и вправо, если в ряде стоит +1. В результате получим смещение, показанное на рис. 2.4.1 (слева). Ясно, что в случае «идеального[1]» классификатора (–1, –1, –1, +1, +1, +1, +1) мы

[1] У которого можно выбрать порог так, чтобы обеспечить 100%-е качество на контроле.

будем двигаться сначала вправо, а затем вверх. В случае «наихудшего» – сначала вверх, потом вправо.

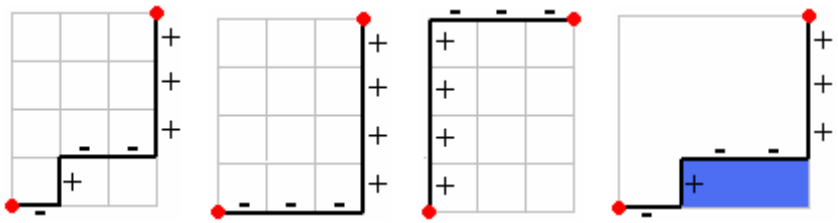


Рис. 2.4.1. Траектории перемещения точки: рассматриваемый случай, идеальный, наихудший и ROC-кривая.

Очень естественно измерять качество нашего семейства классификаторов площадью под получаемой кривой. Эта величина будет показывать качество семейства классификаторов «в среднем[1]». Необходимо только произвести нормировку так, чтобы качество идеального семейства (в котором существует идеальный классификатор) было равно 1. Для этого при смещении вправо перемещаемся на величину $1/|\{x \in U \,|\, A(x) = -1\}|$, а при смещении вверх – на $1/|\{x \in U \,|\, A(x) = +1\}|$ для конечной контрольной выборки U. Полученная кривая (см. рис. 2.4.1 справа) называется **ROC-кривой** (receiver operating characteristic). Площадь под ней называют **AUC-функцией** (area under curve).

Замечание. Отметим, что с помощью AUC-ROC-функции можно измерять и «качество признаков»: качеством j-го признака считать качество рассмотренного выше алгоритма при $g(x) = f_j(c)$ (т.е. алгоритм просто сравнивает значение j-го признака с порогом). Необходимо только учесть, что если говорить о качестве признака, то оно должно быть одинаковым у признаков, которые отличаются на ненулевой множитель (в том числе на –1). Поэтому вместо значения h AUC-ROC-функции надо брать значение $|2h - 1|$.

Рис. 2.4.2. Проекции объектов на «худший» (слева) и «лучший» (справа) признак.

[1] Здесь очень специфическое понятие «в среднем». Попробуйте объяснить, что же показывает эта величина?

Глава 3. БАЙЕСОВСКИЙ ПОДХОД

§3.1. Байесовский классификатор

Байесовский подход (к решению задач классификации) исходит из гипотезы, что данные имеют вероятностную природу и известны (или могут быть получены с приемлемой точностью) вероятности $P(\Omega_i)$ наблюдения объектов из i-го класса и функции распределения[1] $P(x|\Omega_i)$, $i \in \{1,2,\ldots,l\}$, где l – число классов. При таких предположениях удаётся построить **оптимальный классификатор**. Оптимальность понимается в смысле минимизации **функционала среднего риска**

$$\sum_i \sum_s \lambda_{is} \cdot P(\Omega_i) \cdot P(a(x) = s \,|\, \Omega_i)$$

здесь λ_{is} – штраф за отнесение алгоритмом объекта из i-го класса к s-му классу, а $P(\Omega_i) \cdot P(a(x) = s \,|\, \Omega_i)$ – вероятность этого события (точнее, вероятность появления объекта, который будет так неверно классифицирован). **Оптимальный классификатор** относит объект x к s-му классу:

$$s = \arg\min_r \sum_i \lambda_{ir} \cdot P(\Omega_i) \cdot P(x|\Omega_i).$$

Замечание. Запись $P(a(x) = s \,|\, \Omega_i)$ не совсем корректна. Например, в случае дискретного распределения это мера на множестве описаний объектов x.

Замечание. Мы сами часто пользуемся байесовским классификатором. Например, пусть мы играем в «орёл-решка», причём «монетой», которая падает орлом вверх с вероятностью 0.9, а решкой вверх – с вероятностью 0.1. При угадывании исхода мы получаем рубль, в противном случае его теряем. Естественно, наша ставка всегда будет «орёл». Теперь изменим правила. Пусть за угадывание орла мы получаем рубль, за неугадывание орла мы теряем рубль, за угадывание решки мы получаем 100 рублей, а за неугадывание решки мы теряем 100 рублей. Какой теперь будет наша ставка? Обратите внимание, что мы поменяли штрафы! Это примеры для дискретного случая (объекты являются элементами конечного множества), **в дальнейшем будем рассматривать непрерывный случай**.

---

[1] Здесь это вероятность наблюдения объекта x, при условии, что это объект i-го класса (в случае дискретного распределения) или значение плотности распределения описания объекта x из i-го класса (в случае непрерывного распределения). В этой главе символ Ω_i используем по традиции изложения байесовского подхода (в некотором смысле, этот символ означает «принадлежность объекта i-му классу»).

Пример. Разберём подробно задачу с двумя классами на прямой[1] (см. также [Местецкий, 2002]). Если классификатор относит объект к первому классу, то его средняя ошибка (средний штраф) –

$$\lambda_{11} \cdot P(\Omega_1) \cdot P(x|\Omega_1) + \lambda_{21} \cdot P(\Omega_2) \cdot P(x|\Omega_2),$$

а если ко второму, то

$$\lambda_{12} \cdot P(\Omega_1) \cdot P(x|\Omega_1) + \lambda_{22} \cdot P(\Omega_2) \cdot P(x|\Omega_2).$$

Естественно, надо выбрать из этих величин наименьшую, т.е. сравнить

$$(\lambda_{11} - \lambda_{12}) \cdot P(\Omega_1) \cdot P(x|\Omega_1) <> (\lambda_{22} - \lambda_{21}) \cdot P(\Omega_2) \cdot P(x|\Omega_2).$$

Считая, что не происходит деления на ноль (а это вполне естественное предположение), получаем

$$\frac{P(x|\Omega_1)}{P(x|\Omega_2)} <> \frac{(\lambda_{22} - \lambda_{21}) \cdot P(\Omega_2)}{(\lambda_{11} - \lambda_{12}) \cdot P(\Omega_1)}.$$

Если объекты классов распределены по нормальным законам (с параметрами (a_1, σ_1) и (a_2, σ_2)), тогда всё сводится к сравнению

$$\frac{(x-a_2)^2}{\sigma_2^2} - \frac{(x-a_1)^2}{\sigma_1^2} <> \ln\left(\frac{(\lambda_{22} - \lambda_{21}) \cdot P(\Omega_2) \cdot \sigma_1}{(\lambda_{11} - \lambda_{12}) \cdot P(\Omega_1) \cdot \sigma_2}\right).$$

Если дисперсии σ_1, σ_2 различны, то классификатор внутренность некоторого отрезка относит к одному классу (в вырожденном случае – одну точку), а все остальные точки прямой – ко второму классу. Если же дисперсии одинаковы, то классификатор всё, что левее некоторой точки на прямой, относит к одному классу, а всё, что правее, – к другому (кроме вырожденного случая, когда распределения двух классов совпадают). **ДЗ** *Попробуйте дать интерпретацию этому факту. Докажите, что в случае нормальных распределений двух классов на плоскости разделяющая поверхность является поверхностью порядка не выше двух.*

ДЗ *Рассмотрите задачу с двумя стрелками, которые стреляют по одной мишени / по разным мишеням. Необходимо по отметке пулевого отверстия определить, кто делал выстрел. Считайте, что распределение пулевых отметок подчинено нормальному закону с центром в середине мишени. Дисперсии (а здесь даже матрицы ковариаций) характеризуют точность стрелка. Перед тем, как получить аналитические формулы для принятия решения, ответьте на вопрос, как бы Вы осуществили классификацию в такой задаче? Например, когда Вы видите отметки пуль двух стрелков, сделавших по 5 выстрелов в одну мишень, один из которых стреляет хуже другого. Что изменится, если вы знаете, что один сделал 3 выстрела, а второй – 7 (изменение* $P(\Omega_i)$*)? Придумайте ситуацию в этой задаче, когда значения штрафов* λ_{11}, λ_{22} *«неразумно» положить равным нулю.*

[1] Тем более что при выполнении заданий практикума каф. ММП ф-та ВМК МГУ необходимо в явном виде находить байесовский классификатор в двумерных и одномерных пространствах.

Таким образом, необходимо уметь оценивать величины $P(\Omega_i)$ и $P(x|\Omega_i)$. В первом случае всё очевидно: значение $P(\Omega_i)$ следует положить равным частоте встречаемости представителей i-го класса в выборке. Во втором **необходимо восстанавливать плотность по выборке** (причём плотность распределения каждого из классов). Есть три группы методов восстановления плотности: **параметрическое восстановление** (знаем плотность с точностью до параметров и оцениваем эти параметры), **непараметрическое** (пытаемся восстановить плотность, не задаваясь явной формулой; как правило, с помощью локального оценивания), **восстановление смеси распределений** (предполагаем, что плотность представима в виде суммы относительно простых функций).

ДЗ *Подумайте, как оценивать вероятность* $P(\Omega_i)$, *если у нас есть некоторые априорные сведения о её значении? Например, есть две разновидности одной болезни, которые* (*в повседневной жизни*) *встречаются примерно с одной частотой, но в нашей выборке у* 13*-ти пациентов первая разновидность, а у* 17*-ти – вторая. Как в этом случае оценить* $P(\Omega_i)$*? Подумайте, как оценивать плотности* $P(x|\Omega_i)$, *если есть априорные сведения о распределении? Приведите примеры реальных задач, в которых такие сведения есть.*

Зачем нужен байесовский классификатор?

1. Он действительно хорошо работает на практике, **когда данные имеют вероятностную природу** и есть «достаточно большая» выборка (или известны параметры распределения).
2. Он просто реализуется, его можно попробовать применить в задаче «для первой пробы», он быстро классифицирует, поскольку не является **ленивым классификатором** (см. ниже).
3. Его следует исследовать для овладения основами классификации. Поскольку его оптимальность доказана, то другие алгоритмы следует сравнивать с идеальным байесовским классификатором. Кроме того, при проведении экспериментов **теоретическую оценку средней потери следует сравнить с фактической**. Напомним, что на практике параметры распределения восстанавливаются по ограниченной выборке, поэтому реальный байесовский классификатор не оптимален, в отличие от идеального теоретического. **На этом классификаторе следует понаблюдать эффекты выбросов, шумовых признаков и т.д.**
4. Он разделяет объекты **достаточно простыми**, но нетривиальными разделяющими поверхностями (в случае нормальных распределений).

Тестирование байесовского классификатора на модельной задаче (постановка задачи).

1. Сгенерируйте выборку с заданным распределением (при заданных $P(\Omega_i)$ и $P(x|\Omega_i)$, лучше ограничиться случаями одномерных и двумерных выборок, чтобы эксперименты допускали визуализацию).
2. Вычислите аналитически среднюю ошибку оптимального классификатора (следует ещё определить оптимальную разделяющую поверхность и изобразить её при визуализации[1]). При невозможности аналитического вычисления часто удаётся оценить её методами Монте-Карло (см. [Воронцов]).
3. Вычислите ошибку байесовского классификатора (с восстановленными по модельным данным функциями распределения)

$$\sum_i \sum_s \lambda_{is} \cdot |\{a(x) = s \,|\, x \in \Omega_i \cap \tilde{X}\}| / |\tilde{X}|$$

на контрольной выборке $\tilde{X}$.

4. Найдите ответы на следующие вопросы:

- Как зависит качество классификации от параметров выборки (длины, разнесённости центров классов, дисперсий и т.д.)?
- Как зависит качество классификации от качества восстановления плотности (оценки распределения)? Придумайте несколько функционалов качества оценки распределения (например близость параметров функций плотности распределений[2], супремум модуля разности функций плотности распределений, интеграл от модуля разности и т.д.). Какой функционал качества позволяет предсказать ошибку классификации?
- Насколько **наивный байесовский классификатор (naïve Bayes)**, который строится в предположении, что признаки являются независимыми случайными величинами, уступает обычному на модельных данных?

§3.2. Восстановление функций плотности

Непараметрические методы восстановления плотности. Здесь и далее считаем, что задана выборка $x^1,\ldots,x^m$ в пространстве $\mathbf{R}^d$. В **методе окон Парзена** плотность оценивается по формуле

$$\frac{1}{mh}\sum_{i=1}^{m} K\left(\frac{x - x^i}{h}\right),$$

где $K(x)$ – функция окна. В точке (0,0,…,0) она принимает максимальное значение, а при удалении от этой точки монотонно невозрастает, интеграл по всему пространству от неё равен 1. Таким образом, она оценивает плотность

[1] Кстати, подумайте, как изобразить разделяющую поверхность алгоритма (тут могут быть небольшие технические сложности).

[2] Имеется в виду получившейся (восстановленной) и истинной.

распределения выборки, состоящей всего из одной точки. Параметр h определяет «размазанность» этой локальной плотности. Термин «окно» проистекает из классического вида функции

$$K\left((z_1,\ldots,z_d)\right)=\begin{cases}1, & \forall j\in\{1,2,\ldots,d\}\,|\,|z_i|\le 0.5\\ 0, & \text{иначе.}\end{cases}$$

Ясно, что при таком выборе функции окна плотность будет ступенчатой, поэтому на практике используют более гладкие функции, например

$$K(\tilde{z})=\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\tilde{z}^{\mathrm{T}}\tilde{z}}{2}\right).$$

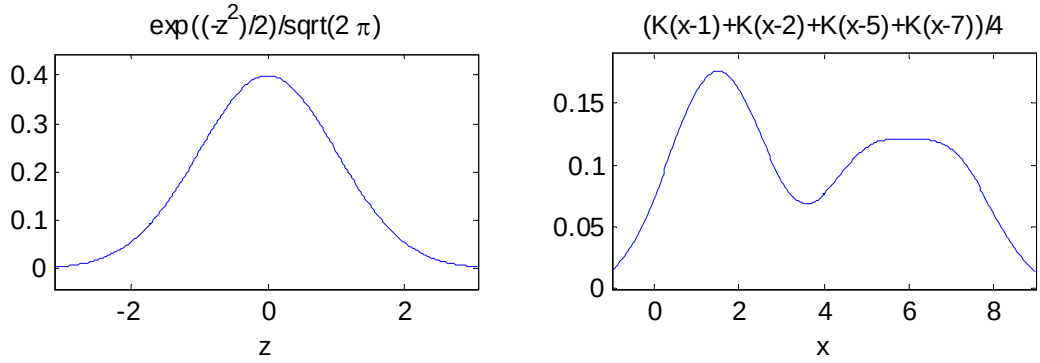


Рис. 3.2.1. График функции окна (слева) и восстановленная по точкам 1, 2, 5, 7 с помощью такой функции плотность (справа).

MatLab-код для получения рис. 3.2.1

```
K = @(z) exp((-z.^2)/2)./sqrt(2*pi);
f = @(x) (K(x-1)+K(x-2)+K(x-5)+K(x-7))/4;
clf; subplot(1,2,1); ezplot(K);
subplot(1,2,2); ezplot(f,[-1 9]);
```

ДЗ ***Ширина окна*** h *существенно влияет как на качество восстановления плотности, так и на качество классификации в байесовском алгоритме* (*при оценивании плотности методом Парзена*). *Исследуйте эти влияния* (*это два задания!*). *Доведите исследования до конкретных рекомендаций пользователю по выбору параметра. Постройте графики плотности при различных значениях* h, *постройте разделяющие поверхности классификатора при различных значениях. Примените скользящий контроль для выбора оптимальной ширины окна* (*оцените при этом эффективность такого подхода*).

ДЗ *Проведите исследования, как выбор функции окна влияет на качество восстановления плотности и качество байесовского алгоритма. Попробуйте, как минимум,* 5 *различных видов функции окна* (*см.* [Воронцов].). *Посмотрите, как меняются разделяющие поверхности алгоритма классификации.*

Параметрические методы восстановления плотности исходят из гипотезы, что плотность $p(x;\theta)$ задана с точностью до параметра θ, который, как правило, определяется методом максимального правдоподобия. Максимизируют функцию

$$\prod_{i=1}^{m} p(x^i;\theta)$$

или (что бывает проще) –

$$\sum_{i=1}^{m} \ln(p(x^i;\theta)).$$

ДЗ *Докажите, что для многомерного нормального распределения*

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}\,|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)\right)$$

оценка по методу максимального правдоподобия даёт следующие значения

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^i\,,\ \Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^i-\mu)(x^i-\mu)^{\mathrm{T}}\,.$$

Являются ли эти оценки несмещёнными? При каком числе объектов матрица Σ *заведомо будет вырожденной? Какие вы знаете способы борьбы с этой вырожденностью? Попробуйте применить их на практике для малых выборок. Оцените их эффективность. См.* [Воронцов].

ДЗ *Проверьте экспериментально, как выбросы и шумы влияют на качество параметрического восстановления плотности. Под шумами здесь понимается сдвиг каждого объекта* (*или объектов какой-то части выборки*) *в его* ε*-окрестность.*

Восстановление смеси распределений. Предполагается, что плотность записана в виде

$$p(x) = \sum_{j=1}^{k} w_j p_j(x;\theta_j)\,,\qquad \sum_{j=1}^{k} w_j = 1,\ \ \forall j \in \{1,2,\ldots,k\}\ \ w_j \ge 0\,,$$

а функции $p_j(x;\theta_j)$ также имеют смысл плотностей (**плотности компонент смеси**). Часто задачу восстановления такой плотности очень сложно решить аналитически (например методом максимального правдоподобия), поэтому применяют итерационные алгоритмы. Стандартный алгоритм оценки параметров w_j, θ_j – **EM-алгоритм (expectation – maximization)**. В нём последовательно повторяются два этапа. На первом (E-шаг) – вычисляются значения

$$g_j^i = \frac{w_j p_j(x^i;\theta_j)}{\sum_{s=1}^{k} w_s p_s(x^i;\theta_s)},$$

$i \in \{1,2,\ldots,m\}$, $j \in \{1,2,\ldots,k\}$. На втором (М-шаг) – вычисляются значения

$$\theta_j = \arg\max_{\theta} \sum_{i=1}^{m} g_j^i \ln(p_j(x^i;\theta)), \quad w_j = \frac{1}{m}\sum_{i=1}^{m} g_j^i,$$

$i \in \{1,2,\ldots,m\}$, $j \in \{1,2,\ldots,k\}$.

Шаги повторяются, пока вычисляемые параметры не стабилизируются. В случае, если компоненты имеют многомерное нормальное распределение с параметрами μ_j, Σ, оптимизационная задача М-шага имеет аналитическое решение

$$\mu = \frac{1}{mw_j}\sum_{i=1}^{m} g_j^i x^i, \quad \Sigma = \frac{1}{mw_j}\sum_{i=1}^{m} g_j^i (x^i - \mu_j)(x^i - \mu_j)^{\mathrm{T}}.$$

ДЗ *При реализации EM-алгоритма попробуйте несколько различных эвристик начального выбора параметров. Сравните их.*

ДЗ *Решите проблему выбора числа компонент. Попробуйте последовательное добавление/удаление компонент.*

ДЗ *Попробуйте применить EM-алгоритм при «наивном» предположении о некоррелированности признаков в каждой компоненте смеси. Сравните наивный подход со стандартным.*

См. также [Воронцов], [Ветров, Кропотов].

§3.3. Оценка среднего значения

При оценивании параметров в параметрических методах восстановления плотности (и вообще везде, где требуется оценка параметров, например в EM-алгоритме) надо понимать, что способы оценивания «заточены» под идеальную выборку (без шумов и выбросов), которая не реализуется на практике. Поэтому часто матожидание нормального распределения оценивают не по формуле

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^i, \tag{3.3.1}$$

а в несколько этапов. После получения оценки (3.3.1) удаляют из выборки несколько точек, которые максимально удалены от μ, а затем для новой выборки пересчитывают (3.3.1). В [Шурыгин, 2009] предлагается следующий алгоритм (для одномерного случая)

Алгоритм А.М. Шурыгина

1. Если $m \le 2$, то пользуемся формулой (3.3.1). Выход.
2. Пусть $x^1 \le \ldots \le x^m$ (без ограничения общности).

3. Если $\frac{x^1 + x^m}{2} \le x^2$, то удаляем из выборки x^1. Переходим к п.1
(с соответствующей перенумерацией объектов).
4. Если $\frac{x^1 + x^m}{2} \ge x^{m-1}$, то удаляем из выборки x^m. Переходим к п.1
(с соответствующей перенумерацией объектов).
5. Исключаем из выборки x^1, x^m, но добавляем в неё $\frac{x^1 + x^m}{2}$.

ДЗ *Ясно, что представленный алгоритм эвристический и легко может быть модифицирован. Предложите свои методы борьбы с выбросами. Проверьте их эффективность на модельных выборках.*

Полезно понимать, что такое «среднее». Выражение (3.3.1) минимизирует функцию

$$\sum_{i=1}^{m}(x^i - \mu)^2, \qquad (3.3.2)$$

если её заменить функцией

$$\sum_{i=1}^{m}|x^i - \mu|, \qquad (3.3.3)$$

то ответом будет медиана (**ДЗ** *докажите это*!). Кстати, на практике очень часто замена среднего арифметического медианой повышает качество работы алгоритма (видимо, это связано с тем, что медиана более устойчива к выбросам). В [Шурыгин, 2009] отмечено, что с таким же успехом для минимизации можно выбрать функцию вида

$$\sum_{i=1}^{m} f(x^i, \mu). \qquad (3.3.4)$$

Полученная в результате решения задачи минимизации оценка μ называется **оценкой минимального контраста.** Если после дифференцирования (здесь рассматриваем одномерный случай) получается

$$\sum_{i=1}^{m}\psi(x^i - \mu) = \sum_{i=1}^{m}(x^i - \mu)\xi(x^i - \mu) = 0,$$

для некоторых функций ψ (оценочная функция) и ξ (весовая функция), то часто успешно применяется итеративный способ вычисления параметра μ по формуле

$$\mu = \frac{\sum_{i=1}^{m} x^i \xi(x^i - \mu)}{\sum_{i=1}^{m} \xi(x^i - \mu)}.$$

Заметим, что $\xi(x^i - \mu)$ – это вес, с которым учитывается точка x^i. Если весовая функция принимает маленькие значения при больших по модулю аргументах, значит помогает «не учитывать выбросы». Мешалкин Л.Д. (1977) предлагал[1] использовать $\psi(y) = ye^{-\lambda y^2/2}$, т.е. $\xi(y) = e^{-\lambda y^2/2}$.

ДЗ *Предложите свои оценочные функции. Проверьте их эффективность на практике* (*например в задаче оценки среднего нормального распределения для зашумлённой выборки*). *Тем самым вы повторите Принстонский эксперимент* 1972 *года. См.* [Шурыгин, 2009].

Приведём оценки среднего и матрицы ковариации Мешалкина Л.Д. для многомерного распределения, а точнее систему уравнений для их поиска:

$$\begin{cases} \sum_{i=1}^{m}(x^i - \mu)e^{-\lambda \cdot q_i/2} = 0, \\ \sum_{i=1}^{m}\left((x^i - \mu)(x^i - \mu)^{\mathrm{T}} - \frac{1}{1+\lambda}C\right)\cdot e^{-\lambda \cdot q_i/2} = 0, \end{cases}$$

где $q_i = (x^i - \mu)^{\mathrm{T}} C^{-1}(x^i - \mu)$.

ДЗ *Пользуясь тем, что медиана минимизирует функцию* (3.3.3) *в одномерном случае, обоснуйте, что одно из «естественных» обобщений понятия медиана на многомерный случай может быть получено итерационным процессом по следующей формуле:*

$$\mu = \frac{\sum_{i=1}^{m}\frac{x^i}{\sqrt{q_i}}}{\sum_{i=1}^{m}\frac{1}{\sqrt{q_i}}}.$$

Попробуйте использовать это понятие среднего в EM-алгоритме на «обычных» и зашумлённых данных.

Пример. Отметим, что необходимость минимизации функций типа (3.3.4) и нахождение «среднего» возникает в анализе данных очень часто. Например, при прогнозировании некоторой величины, когда есть ответы нескольких алгоритмов (или экспертов). Естественно выдавать их усреднённое значение, но какое? На рис. 3.3.1 – 3.3.2 представлены графики функций (3.3.3), (3.3.2) для системы точек {1, 3, 4, 6, 8} вместе с графиками ошибок для каждой точки в отдельности. При прогнозировании финансовых временных рядов для оценки эффективности прогнозирования часто используют SMAPE-функцию[2]:

[1] Вся необходимая библиография приведена в [Шурыгин, 2009].
[2] SMAPE = Symmetric Mean Absolute Percent Error.

$$\mu = \frac{2}{q}\sum_{i=1}^{q}\frac{|\,y(x^i) - A(x^i)\,|}{y(x^i) + A(x^i)},$$

т.е. абсолютную разницу нашего прогноза и истинного значения делят на их среднее арифметическое (это вполне логично: одно дело, когда мы предсказываем курс валюты 31.00 вместо 30.00, а другое – 2.00 вместо 1.00). На рис. 3.3.3. представлен график функции

$$2\cdot\sum_{i=1}^{m}\frac{|\,x^i - \mu\,|}{x^i + \mu}. \qquad (3.3.5)$$

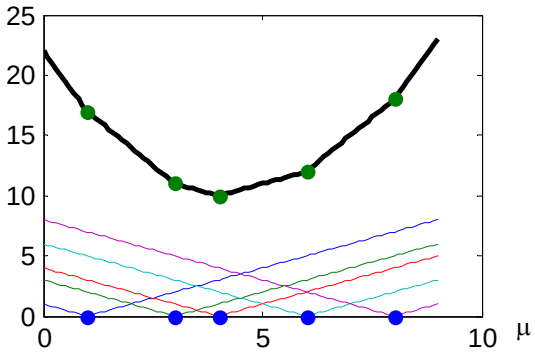


Рис 3.3.1. График функции (3.3.3).

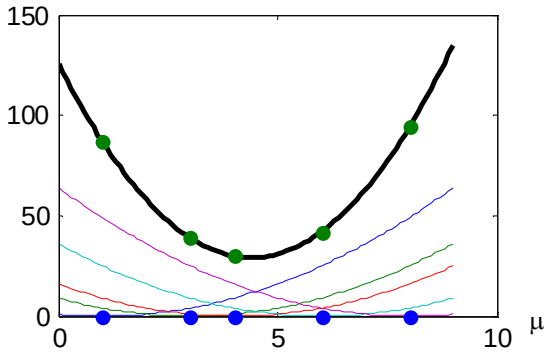


Рис 3.3.2. График функции (3.3.2).

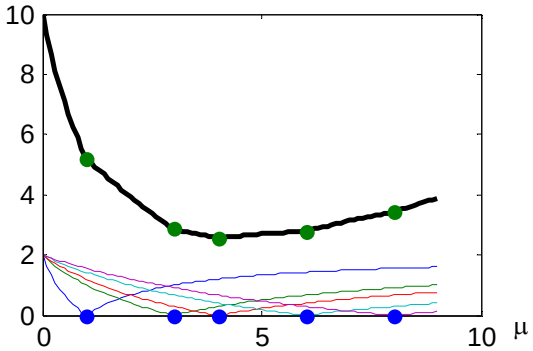


Рис 3.3.3. График функции (3.3.5).

MatLab-код для получения рис. 3.3.1.

```
X = [1 3 4 6 8];
x = 0:0.1:9;
[rX rx] = meshgrid(X,x);
f = abs(rX-rx);
F = sum(f,2);
plot(x,f)
hold on;
plot(x,F,'k','LineWidth',2);
scatter(X,zeros(size(X)),35,'filled')
scatter(X,F(ismember(x,X)),25,'filled')
```

ДЗ *Напишите в среде MatLab функцию минимизации выражения* (3.3.4). *Проиллюстрируйте её работу на графиках.*

Моделирование многомерных нормальных распределений

Если уметь моделировать стандартное одномерное нормальное распределение с плотностью $p(x;0,1)$, то легко генерировать n-мерные нормальные векторы x с плотностью распределения

$$\frac{1}{(2\pi)^{n/2}}\exp\left(-\frac{x^{\mathrm{T}}x}{2}\right)$$

(каждую компоненту независимо генерируем с плотностью $p(x;0,1)$). Нетрудно видеть (**ДЗ** *докажите это*), что для случайного вектора $z = m + Ax$, где A – невырожденная $n \times n$-матрица, плотность распределения равна $p(z; m, AA^{\mathrm{T}})$, где

$$p(z;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}\,|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(z-\mu)^{\mathrm{T}}\Sigma^{-1}(z-\mu)\right).$$

С плотностью $p(x;0,1)$ распределены (причём независимо) случайные величины

$$\sqrt{-2\ln(\xi_1)}\sin(2\pi\xi_2),$$
$$\sqrt{-2\ln(\xi_1)}\cos(2\pi\xi_2),$$

где ξ_1, ξ_2 –две независимые случайные равномерно распределённые на отрезке $[0,1]$ величины.

Глава 4. МЕТОД БЛИЖАЙШЕГО СОСЕДА

§4.1. Описание метода ближайшего соседа

Метод ближайшего соседа (БС) исходит из очень простой и «самой естественной» гипотезы: объект надо классифицировать в тот класс, на представителей которого он похож. «**Что такое похож**» формализуется некоторой функцией сходства, как правило, метрикой. Она может

1) определяться постановкой задачи (объекты уже заданы как элементы метрического пространства),

2) задаваться экспертом[1] (объекты, например, заданы в признаковом пространстве, но в этом пространстве «очевидно», как надо ввести метрику для приемлемого решения),

3) получаться в результате решения оптимизационной задачи (перебираем метрики из некоторого класса и максимизируем функционал качества решения)[2].

После выбора функции сходства необходимо определиться, как **учитывать** это **сходство.**

Пусть $\{x^t\}_{t=1}^m$ – обучающая выборка, ρ – метрика на множестве объектов, причём (без ограничения общности)

$$\rho(x,x^1) \le \rho(x,x^2) \le \ldots \le \rho(x,x^m), \qquad (4.1.1)$$

тогда ответ **алгоритма одного ближайшего соседа** на объекте x – классификация объекта x^1:

$$A(x) = y(x^1).$$

Ответ **алгоритма *k* ближайших соседей** – классификация, которая чаще всего встречается среди объектов $x^1, x^2, \ldots, x^k$:

$$A(x) = \arg\max_j |\{t \in \{1,2,\ldots,k\} \,|\, y(x^t) = j\}|.$$

В различных обобщениях ответ записывается в виде

$$A(x) = \arg\max_j \Gamma_j(x, \{x^t\}_{t=1}^k).$$

В [Воронцов] описан **метод взвешенных ближайших соседей**:

$$\Gamma_j(x, \{x^t\}_{t=1}^k) = \sum_{t\in\{1,2,\ldots,k\}:\, y(x^t)=j} q^t,$$

метод парзеновского окна:

$$\Gamma_j(x, \{x^t\}_{t=1}^k) = \sum_{t\in\{1,2,\ldots,k\}:\, y(x^t)=j} K\left(\frac{\rho(x^t,x)}{h}\right),$$

---

[1] Вы сами можете быть этим экспертом.

[2] Выбор метрики – это отдельная проблема, и метод БС вообще является самым лучшим методом… **после удачного выбора функции сходства.**

метод парзеновского окна переменной ширины:

$$\Gamma_j(x,\{x^t\}_{t=1}^k) = \sum_{t\in\{1,2,\ldots,k\}:\, y(x^t)=j} K\left(\frac{\rho(x^t,x)}{h(x,\{x^t\}_{t=1}^k)}\right),$$

где $h(x,\{x^t\}_{t=1}^k)$ – некоторая функция, например $h(x,\{x^t\}_{t=1}^k)=\rho(x^k,x)$ или $h(x,\{x^t\}_{t=1}^k)=\rho(x^1,x)+\ldots+\rho(x^k,x)$ (в [Воронцов] предложен другой вариант). Здесь $K:[0,+\infty)\to[0,+\infty)$ – функция, которая принимает в нуле максимальное значение и является невозрастающей. Значение $\Gamma_j(x,\{x^t\}_{t=1}^k)$ называется **оценкой принадлежности объекта** x **к** j**-му классу** [Журавлёв, 1978]. Функция Γ_j может учитывать также положения объектов в окрестности. Например, на рис. 4.1.1 показана окрестность объекта x, в которой с точки зрения расстояний до соседних объектов нет предпочтения для отнесения объекта x к одному из классов, но в этой окрестности объекты одного класса «сильно» разбросаны, а другого – наоборот.

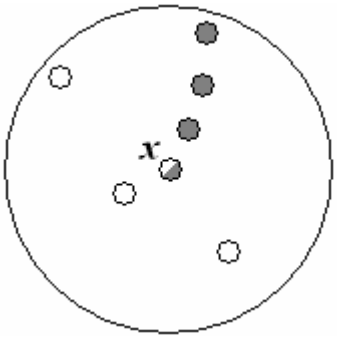


Рис. 4.1.1. Расположение объектов (соседей) в окрестности.

Замечание. k ближайших соседей не всегда однозначно определены, поскольку в (4.1.1) все неравенства нестрогие (вырожденный случай – расстояния до всех объектов из обучения равны).

Замечание. В задаче с двумя непересекающимися классами k обычно выбирают нечётным.

Есть алгоритмы, которые анализирую близость, но основаны на других формулах оценки принадлежности, например на формуле

$$A(x)=\arg\max_j \Gamma_j(x,\{x^t \mid \rho(x,x^t)\le\varepsilon\cdot k\}),$$

где k выбирается минимальным натуральным числом, для которого множество

$$\{t\in\{1,2,\ldots,k\} \mid \rho(x,x^t)\le\varepsilon\cdot k\},\ \varepsilon>0,$$

непустое (или содержит заранее заданное число объектов), см. рис. 4.1.2.

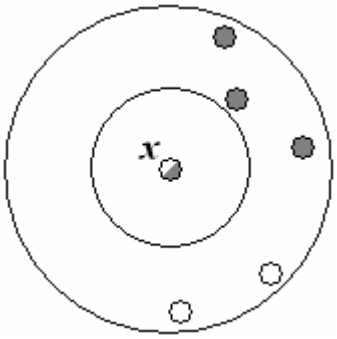


Рис. 4.1.2. Кольца соседства вокруг объекта.

Есть результаты (см., например, [Дуда, Харт, 1976]) теоретического обоснования метода БС. Для достаточно больших выборок ($m \to \infty$) вероятность ошибки $P_{\text{БС}}$ метода БС удовлетворяет неравенствам

$$P_{\text{Байес}} \le P_{\text{БС}} \le P_{\text{Байес}}\left(2 - \frac{l}{l-1} P_{\text{Байес}}\right),$$

где $P_{\text{Байес}}$ – вероятность ошибки байесовского классификатора, l – число классов в задаче.

НЕДОСТАТКИ МЕТОДА БС

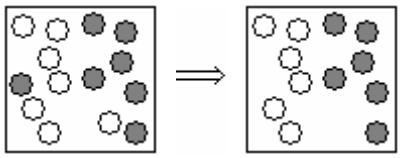
Недостаток	Метод устранения
Это «ленивый» алгоритм, поэтому процедура обучения отсутствует и приходится хранить всю выборку.	Хранят только часть выборки (**эталонные объекты**), которая выбирается так, чтобы обеспечить такую же функциональность алгоритма, как и при полной выборке.
	Есть методы быстрого нахождения ближайших соседей (например, пространство разбивается на области, сосед ищется в области объекта, а затем в ближайших соседних областях).
Неустойчивость к шумам (неверным значениям функции сходства, например, из-за ошибок в описании объектов) и выбросам (объектам, которые лежат в области «чужих» классов)	В методе используют «достаточно большое» количество соседей (следуя гипотезе, что не может быть много выбросов рядом).
	Применяют процедуры устранения выбросов из выборок.
	Применяют методы обобщённых эталонов, когда сравнение происходит не с реальными объектами выборки, а с «виртуальными». Описание эталона может получаться усреднением описаний нескольких объектов, расположенных рядом.

Неоднозначность выбора метрики	Оптимизация метрики; применение «универсальных» метрик с весами, которые вычисляются специальными эвристиками (фильтрами).

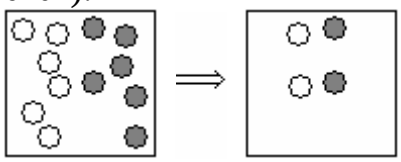
Ленивые алгоритмы (lazy)	**Нетерпеливые алгоритмы (eager)**
Процедура обучения отсутствует (могут настраиваться лишь некоторые «метапараметры» методом скользящего контроля).	В результате процедуры обучения данные переводятся в некоторое сжатое описание (модель).
Данные обрабатываются в процедуре классификации. Используются сами данные.	После создания модели данные не нужны (в процедуре классификации).
Медленная процедура классификации.	Быстрая процедура классификации.
Простейший пример: kNN.	Простейший пример: линейный классификатор (см. дальше).

Пример метода отбора эталонов можно найти в [Воронцов]. Отметим, что подобные методы являются эвристическими (теория отбора эталонов вообще не сильно разработана), но основные этапы подобных алгоритмов следующие.

1. **Устранение выбросов** (удаление объектов, которые «окружены» объектами чужих классов).

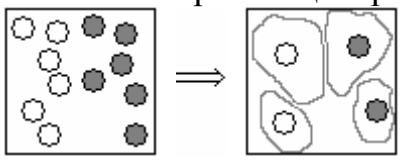


2. **Устранение объектов, если при этом не меняется функциональность алгоритма** (или слабо меняется).



Во многих алгоритмах присутствует этап

3. **Введение новых эталонов** (например, выделение некоторого кластера[1] объектов и замена всех объектов кластера его центром).



Последовательность этапов в алгоритме может быть различной.

[1] Кластер – группа похожих объектов (см. подробнее главу «Кластеризация»).

§4.2. Пример проведения эксперимента в среде MatLab

Продемонстрируем, насколько легко производить вычислительные эксперименты в системе MatLab. Исследуем следующую проблему: **как влияет на качество классификации методом ближайшего соседа в задаче с двумя непересекающимися классами соотношение мощностей классов**? Во многих задачах случайно выбранный объект может с вероятностью близкой к 0.5 относится к первому классу (и ко второму классу), но есть задачи, в которых объекты второго класса встречаются значительно реже (например, в задаче обнаружения цели с помощью радиолокации самолёты неприятеля могут редко залетать на нашу территорию). Попробуем выяснить, какие задачи «хуже»? Сразу отметим, что решить такую глобальную проблему удастся лишь в очень частном (и модельном) случае.

Будем исследовать простейший **алгоритм** – метод ближайшего соседа **с евклидовой метрикой**, объекты – точки в единичном квадрате на плоскости (это наше **признаковое пространство**). Стороны квадрата параллельны осям координат прямоугольной системы координат. Точки квадрата, лежащие левее прямой, параллельной оси Oy и пересекающей квадрат, образуют первый класс; точки, лежащие правее – второй класс (это **описание классов**). Различные положения этой прямой (однозначно определяются точкой пересечения с осью Ox) соответствуют различным соотношениям мощностей классов (положения этой прямой мы и **будем менять в процессе экспериментов**). **Обучающую выборку** можно взять «достаточно представительной» и зафиксировать. Для этого мы используем совокупность равномерно распределённых точек на квадрате. **Контрольная выборка** – точки равномерной сетки на нашем квадрате. **Функционал качества** (который будем исследовать) – процент верных ответов нашего алгоритма.

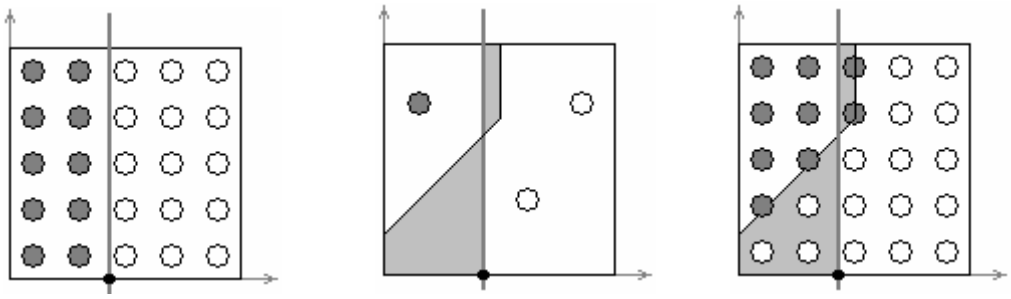


Рис 4.2.1. Классификация (истинная) контрольных объектов, определяемая прямой. Классификация, определяемая обучающими объектами. Классификация контрольных объектов, определяемая обучающими объектами.

В предыдущем абзаце выделены основные понятия, которые необходимо формализовать перед экспериментом:

1. Объект исследования (функционал качества).
2. Описание классов и признакового пространства.
3. Способ формирования обучающей и контрольной выборки.

4. Алгоритм классификации (или модель алгоритмов), а также методы обучения (для алгоритма ближайшего соседа методов обучения нет).
5. Схема проведения экспериментов.

Ниже представлен листинг MatLab-программы, которая проводит такой эксперимент[1].

```
%% Контрольные объекты
%  - это сетка mx*my объектов
mx = 20;
my = 20;
[temp1 temp2] = meshgrid(linspace(0,1,mx),linspace(0,1,my));
Control = [temp1(:), temp2(:)];

%% Обучающая выборка
%  - случайная выборка из mt объектов
mt = 20;
Train = rand([mt 2]);

%% Матрица попарных расстояний
%  Запомните этот трюк...
D = sqrt(sum(( repmat(permute(Control, [1 3 2]), ...
[1 size(Train,1) 1]) - repmat(permute(Train, [3 1 2]), ...
[size(Control,1) 1 1]) ).^2, 3));
% Попробуйте imagesc(D); figure(gcf);

%% Нахождение ближайших соседей
%  Будем менять только классификацию обучения
%  Сами объекты всё время фиксированы, поэтому и соседи
фиксированы
[m i] = min(D');

%% "Правильность" классификации
ne = 5; % число различных положений прямой
Quality = zeros ([1 ne]); % инициализация (можно Quality = [])
j = 1;
Z = linspace(1,0.5,ne); % Это множество пробегает порог (обратный
порядок - для Publish)
% В данной задаче лучше пробегать другое множество... какое?

for z = Z;

    YTrain = Train(:,1)<z; % Классификация обучения
    YControl = Control(:,1)<z; % Классификация контроля

    A = YTrain(i); % Ответ метода ближайшего соседа
```

[1] Из-за специфики оформления данного пособия в коде имеются лишние переносы строк (в длинных командах **scatter**). «Работающим» код становится после устранения переносов или добавления в переносимые строки MatLab-команды многоточие: «...». Это замечание справедливо и для некоторых других листингов.

```
    Quality(j) = mean(A==YControl); % Качество классификации

    % Визуализация
    clf;  hold on;
    % контроль
    scatter(Control(YControl>0,1), Control(YControl>0,2), 20,
A(YControl>0), 's', 'r');
    scatter(Control(YControl<=0,1), Control(YControl<=0,2), 20,
A(YControl<=0), 'o', 'b', 'filled');
    nonright = (A~=YControl); % обвести ошибки
    scatter(Control(nonright,1), Control(nonright,2), 40,
A(nonright), 'k');
    % обучение
    scatter(Train(YTrain>0,1), Train(YTrain>0,2), 50,
YTrain(YTrain>0), 's', 'r', 'LineWidth', 2);
    scatter(Train(YTrain<=0,1), Train(YTrain<=0,2), 50,
YTrain(YTrain<=0), 'o', 'b', 'LineWidth', 2);
    % граница классов (идеальная)
    line ([z z], [1 0], 'LineWidth', 2, 'Color', [.4 .9 .4]);
    title('Classification');

    % вывод результатов
    fprintf('z = %2.2f Quality = %3.2f\n',z,Quality(j)*100);
    pause; % ждать нажатие клавиши

    j = j + 1;
end;

%% Визуализация кривой верных ответов
clf; plot(Z,Quality); title('Quality');
```

Если выбрать опцию редактора MatLaba **File/Publish**, то будет сгенерирован следующий отчёт о работе программы (его анализ показывает, как использовать код для генерации отчёта). Обратите внимание, что не все рисунки, которые выводились в процессе выполнения программы вошли в отчёт. Сгенерировать отчёт можно также командой **publish('filename', 'html')**. При генерации происходит запуск скрипта (т.е. выполнение всех описанных команд, поэтому меняются переменные рабочей области и создаются окна для вывода графики).

Contents

Контрольные объекты

- это сетка mx*my объектов

```
mx = 20;
my = 20;
[temp1 temp2] = meshgrid(linspace(0,1,mx),linspace(0,1,my));
Control = [temp1(:), temp2(:)];
```

Обучающая выборка

- случайная выборка из mt объектов

```
mt = 20;
Train = rand([mt 2]);
```

Матрица попарных расстояний

Запомните этот трюк...

```
D = sqrt(sum(( repmat(permute(Control, [1 3 2]), ...
[1 size(Train,1) 1]) - repmat(permute(Train, [3 1 2]), ...
[size(Control,1) 1 1]) ).^2, 3));
% Попробуйте imagesc(D); figure(gcf);
```

Нахождение ближайших соседей

Будем менять только классификацию обучения

Сами объекты всё время фиксированы, поэтому и соседи фиксированы

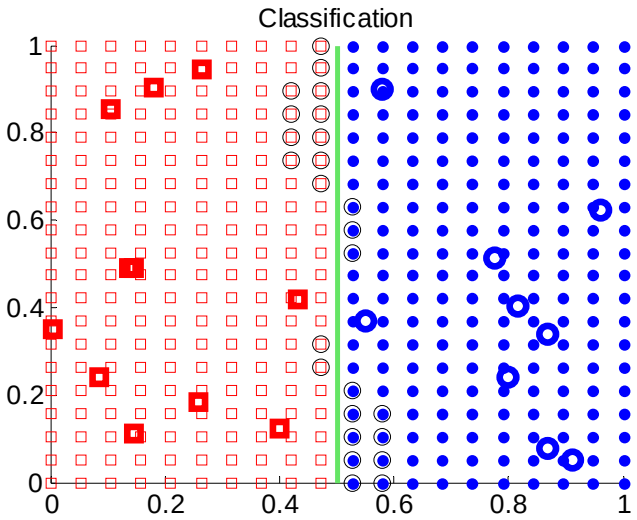
```
[m i] = min(D');
```

"Правильность" классификации

```
ne = 5; % число различных положений прямой
Quality = zeros ([1 ne]); % инициализация (можно Quality = [])
j = 1;
Z = linspace(1,0.5,ne); % Это множество пробегает порог (обратный
порядок - для Publish)
% В данной задаче лучше пробегать другое множество... какое?

for z = Z;

    YTrain = Train(:,1)<z; % Классификация обучения
    YControl = Control(:,1)<z; % Классификация контроля

    A = YTrain(i); % Ответ метода ближайшего соседа

    Quality(j) = mean(A==YControl); % Качество классификации

    % Визуализация
    clf; hold on;
    % контроль
    scatter(Control(YControl>0,1), Control(YControl>0,2), 20,
A(YControl>0), 's', 'r');
    scatter(Control(YControl<=0,1), Control(YControl<=0,2), 20,
A(YControl<=0), 'o', 'b', 'filled');
    nonright = (A~=YControl); % обвести ошибки
    scatter(Control(nonright,1),
Control(nonright,2),40,A(nonright),'k');
    % обучение
    scatter(Train(YTrain>0,1), Train(YTrain>0,2), 50,
YTrain(YTrain>0), 's', 'r', 'LineWidth', 2);
    scatter(Train(YTrain<=0,1), Train(YTrain<=0,2), 50,
YTrain(YTrain<=0), 'o', 'b', 'LineWidth', 2);
    % граница классов (идеальная)
```

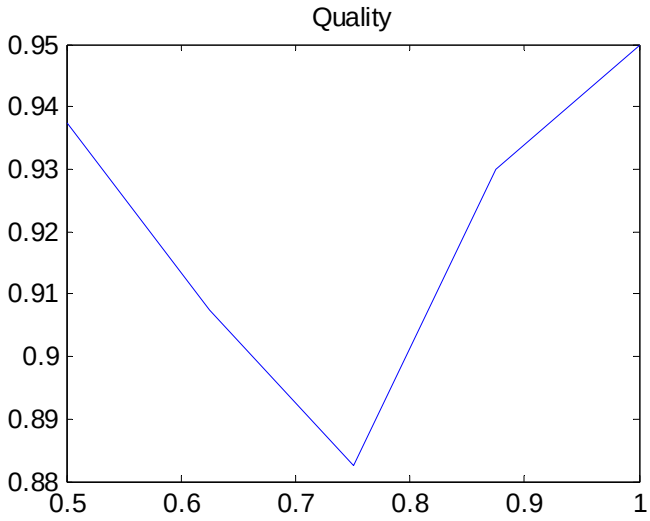
```
    line ([z z], [1 0], 'LineWidth', 2, 'Color', [.4 .9 .4]);
    title('Classification');

    % вывод результатов
    fprintf('z = %2.2f Quality = %3.2f\n',z,Quality(j)*100);
    pause; % ждать нажатие клавиши

    j = j + 1;
end;
```

```
z = 1.00 Quality = 95.00
z = 0.88 Quality = 93.00
z = 0.75 Quality = 88.25
z = 0.63 Quality = 90.75
z = 0.50 Quality = 93.75
```



Визуализация кривой верных ответов

```
clf; plot(Z,Quality); title('Quality');
```



Замечание. В скрипте два знака комментария подряд **%%** начинают новый блок. В MatLabe есть возможность запускать только отдельные блоки кода. Название блока автоматически заносится в меню отчёта. Если название пустое (после знаков **%%** идёт переход строки), то в меню ничего не заносится. Есть возможность вставлять в отчёт формулы в стиле TeXa: **$$e^{\pi i} + 1 = 0$$**. Также есть возможности разметки текста: текст между двумя символами «*» (звёздочка) будет напечатан жирным шрифтом, между «_» (подчёркивание) – курсивом, между «|» (вертикальная черта) – текстом, в котором все символы имеют одну ширину (типа Courier). Строка, которая начинается с «**% ***», будет элементом списка.

Замечание. Блок визуализация можно написать намного проще (здесь сделана визуализация, которая будет «хорошо смотреться» в градациях серого при распечатке отчёта). Например, как показано ниже.

```
    % Визуализация
    clf;
    scatter(Control(:,1), Control(:,2),10,A,'filled');
    hold on;
    nonright = (A~=YControl); % обвести ошибки
    scatter(Control(nonright,1), Control(nonright,2), 30,
A(nonright));
    % обучение
    scatter(Train(:,1), Train(:,2),50,YTrain,'*')
    % граница классов (идеальная)
    line ([z z], [1 0],'LineWidth',2,'Color',[.4 .9 .4]);
    title('Classification');
```

Отметим, что наши эксперименты имели достаточно много методических недостатков и просчётов:

1. При уменьшении мощности одного из классов длина границы оставалась постоянной. Это не кажется «реалистичным». Например, в другой модельной задаче: первый класс – шар радиуса r, второй – разность шара радиуса R и шара радиуса r, $R > r$ (центры всех шаров совпадают), это уже не выполняется при уменьшении r.
2. В нашей задаче есть фиктивный второй признак (вторая координата каждой точки). Следует провести эксперименты без фиктивных признаков и с бОльшим числом фиктивных признаков. Вообще, размерность может сильно влиять на результаты эксперимента (необходимо провести эксперименты с различным числом нефиктивных признаков).
3. Выбранный функционал качества не совсем адекватен. Ясно, что при уменьшении мощности второго класса он стремится к 1, поскольку задача с одним классом решается с точностью 100%. *Какие можно предложить другие функционалы в этой задаче?* Обратите внимание, что на практике верная классификация объектов «малого класса» очень важна:

обнаружение самолётов противника, выявление спама, диагностика редкой болезни и т.д.

4. На практике чаще всего соотношение объектов первого и второго классов в обучении не совпадает с отношением мощностей этих классов (которое, тем более, не известно). *Почему? Как сделать эксперименты более реалистичными?*
5. Пока рассмотрен одиночный эксперимент, но экспериментов надо сделать «достаточно много» при разных значениях параметров.

В экспериментах напрашивается сравнение метода ближайшего соседа с алгоритмами, «заточенными под модельную задачу». В задаче объекты можно разделить гиперплоскостью (в данном случае – прямой), поэтому интересно оценить качество работы линейного классификатора (естественно, при различных способах его настройки). *Как для этой задачи «выглядит»* (*оптимальный!*) *байесовский классификатор*?

В заключение приведём несколько графиков. Это функционалы качества (жирная линия – результат их усреднения), построенные с помощью программы, описанной выше, с другими параметрами (числом объектов в обучении, контроле, числом различных значений порога). *Попробуйте по графикам восстановить значения этих параметров. Какие выводы можно сделать из наших экспериментов? Обратите внимание, что вид жирной линии можно предсказать до проведения экспериментов. Более того, можно теоретически вычислить некоторые её характеристики. Какие? Попробуйте это сделать!*

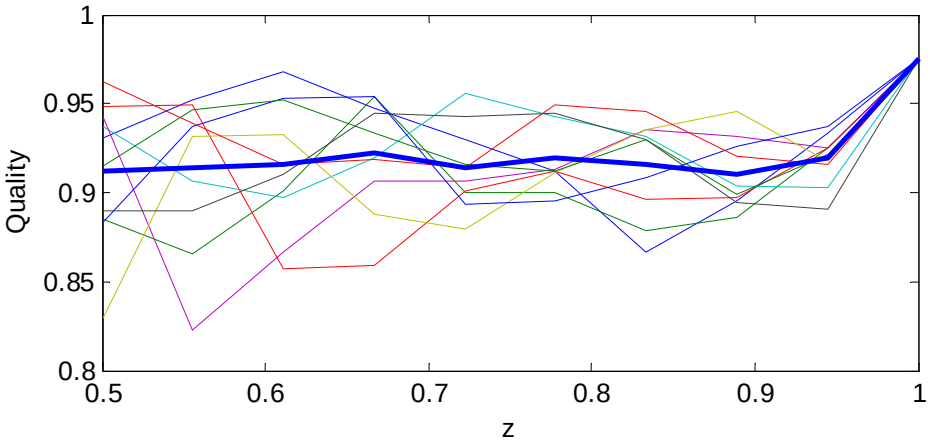


Рис. 4.2.2. Качество классификации в первой серии экспериментов.

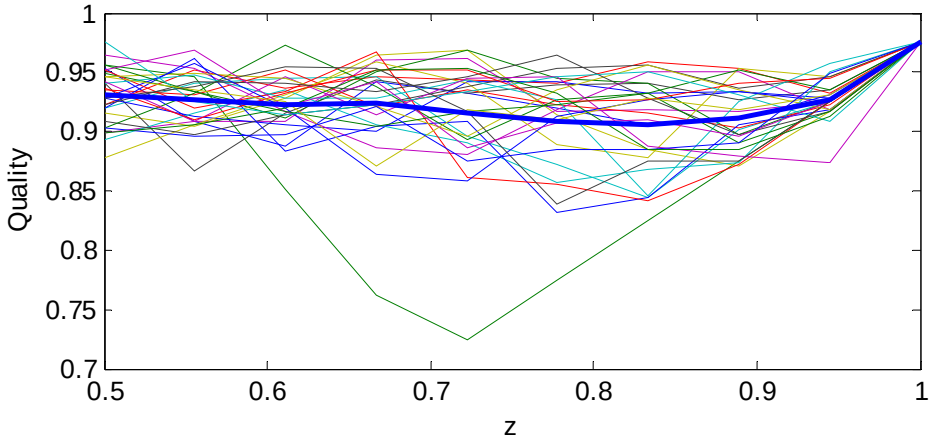


Рис. 4.2.3. Качество классификации во второй серии экспериментов.

Глава 5. ЛИНЕЙНЫЙ КЛАССИФИКАТОР

§5.1. Алгоритм персептрона

Пусть объект x описывается своим признаковым описанием

$$\tilde{x} = (f_0(x), f_1(x), \ldots, f_n(x))^{\mathrm{T}} \in \mathbf{R}^{n+1}.$$

Будем предполагать, что

$$\forall x \quad f_0(x) = 1$$

(всегда можно пополнить признаковое пространство таким фиктивным признаком). Это позволяет записать уравнение гиперплоскости в (исходном) признаковом пространстве в виде

$$\tilde{w}^{\mathrm{T}} \cdot \tilde{x} = w_0 f_0(x) + w_1 f_1(x) + \ldots + w_n f_n(x) = 0,$$

где $\tilde{w}^{\mathrm{T}} = (w_0, w_1, \ldots, w_n)$. **Линейный классификатор** в задаче с двумя непересекающимися классами K_1, K_2 **определяется вектором** $\tilde{w}$ и работает по следующему принципу

$$A(x) = K_1 \text{ при } \tilde{w}^{\mathrm{T}} \cdot \tilde{x} > 0,$$
$$A(x) = K_2 \text{ при } \tilde{w}^{\mathrm{T}} \cdot \tilde{x} < 0$$ [1]

(при $\tilde{w}^{\mathrm{T}} \cdot \tilde{x} = 0$ классификатор отказывается от классификации или относит объект в первый класс). Простейший алгоритм обучения линейного классификатора (построения разделяющей гиперплоскости) – **персептронный**.

Алгоритм персептрона (стохастический режим)

0. Выбрать начальное значение вектора весов $\tilde{w}$ (можно равным нулевому вектору).

1. Перебрать все объекты x обучения.

2. Если объект x классифицирован неверно

при $\tilde{w}^{\mathrm{T}} \cdot \tilde{x} \geq 0$ и $y(x) = K_2$ изменить вектор весов $\tilde{w} := \tilde{w} - \tilde{x}$,

при $\tilde{w}^{\mathrm{T}} \cdot \tilde{x} \leq 0$ и $y(x) = K_1$ изменить вектор весов $\tilde{w} := \tilde{w} + \tilde{x}$,

(если объект $\tilde{x}$ классифицирован верно, то вектор весов не меняется).

3. Если все объекты обучения были классифицированы верно, то линейный классификатор построен, иначе (применялся п.2) перейти к п.1.

Замечание. Термин «персептронный» происходит от другого названия линейного классификатора, принятого в теории нейронных сетей: **персептрон** (или перцептрон).

Обучение линейного классификатора в случае, если существует гиперплоскость, разделяющая объекты двух классов, гарантируется теоремой

[1] Записи «$A(x) = K_1$», «$A(x) = K_2$» используются вместо «$A(x) = 1$», «$A(x) = 2$» для наглядности.

Новикова [Воронцов]. Если такой гиперплоскости не существует, то алгоритм зацикливается (при фиксированном порядке предъявления объектов).

ДЗ *При выполнении задания по практикуму установите, в чём выражается «зацикливание». Предложите методы распознавания зацикливания* (*т.е. невозможности построить гиперплоскость*).

ДЗ *Число исправлений вектора* $\tilde{w}$ *в персептронном алгоритме* (*до полного обучения*) *оценивается в теореме Новикова, их не больше*

$$\left(\frac{\max\limits_{t} \| x^t \|}{\delta} \right)^2$$

при нулевом начальном приближении вектора весов $\tilde{w}$ *и существовании* $\tilde{w}^*$*:* $(\tilde{w}^*)^{\mathrm{T}} \tilde{x} > \delta > 0$ (*существовании линейной разделимости*). *Попробуйте экспериментально выяснить, насколько эта оценка завышена.*

Этот же алгоритм можно записать в виде алгоритма с пакетным режимом обучения:

Алгоритм персептрона (пакетный режим)

0. Выбрать начальное значение вектора весов $\tilde{w}$ (можно равным нулевому вектору).

1. Если множества

$$X_1 = \{x \mid \tilde{w}^{\mathrm{T}} \cdot \tilde{x} \ge 0,\ y(x) = K_2\},\ X_2 = \{x \mid \tilde{w}^{\mathrm{T}} \cdot \tilde{x} \le 0,\ y(x) = K_1\}$$

(здесь x пробегает всю обучающую выборку) пустые, то выйти.

2. Иначе

$$\tilde{w} := \tilde{w} - \sum_{x \in X_1} \tilde{x} + \sum_{x \in X_2} \tilde{x},$$

перейти к п.1.

Режимы обучения

Стохастический[1]	**Пакетный**
Учитывается ошибка на конкретном объекте, которая вызывает коррекцию параметров классификатора. Объекты предъявляются в случайном порядке.	Учитываются ошибки сразу на всех объектах обучения (или на достаточно большом множестве – «пакете»). Суммарная ошибка вызывает коррекцию параметров классификатора.

ДЗ *Сравните экспериментально пакетный и стохастический режимы обучения.*

[1] Если объекты предъявляются в определённом порядке, то режим называют **последовательным**.

Алгоритм персептрона является алгоритмом типа **поощрения-наказания**. «Наказывается» алгоритм за неверную классификацию (п.2 в описаниях алгоритма). Поощрение здесь отсутствует (вообще, можно было бы даже при правильной классификации объекта изменять вектор $\tilde{w}$ так, чтобы алгоритм более уверенно классифицировал объект).

ДЗ *Придумайте процедуры поощрения для линейного классификатора. Проведите их экспериментальные исследования.*

ДЗ *Придумайте несколько эвристик для выбора начального значения вектора весов* $\tilde{w}$. *Сравните их на модельных задачах. См. рекомендации в* [Воронцов].

ДЗ *Работа алгоритма в стохастическом режиме зависит от того, в каком порядке предъявляются объекты* (*в зависимости от порядка предъявления и начального приближения* $\tilde{w}$ *будут построены разные гиперплоскости*). *Попробуйте поэкспериментировать с различными порядками. Например, можно в первую очередь предъявлять классификатору объекты, на которых он ошибался на предыдущем проходе. Верно ли, что если в выборке присутствуют выбросы (которые мешают линейной разделимости классов), то при обучении линейного классификатора он чаще всего будет неверно классифицировать именно эти выбросы?*

§5.2. Минимизация функций ошибки

Алгоритм персептрона является также примером **алгоритма градиентной оптимизации**. Если умножить описания $\tilde{x}$ объектов второго класса на (-1), то задача построения линейного классификатора сведётся к нахождению такого вектора $\tilde{w}$, что $\tilde{w}^{\mathrm{T}} \cdot \tilde{x} \geq 0$ для всех объектов x из обучения. Рассмотрим функцию

$$J(\tilde{w}, \tilde{x}) = \frac{1}{2}(|\tilde{w}^{\mathrm{T}} \cdot \tilde{x}| - \tilde{w}^{\mathrm{T}} \cdot \tilde{x}),$$

которая обращается в ноль при выполнении нужного неравенства, а в противном случае «пропорциональна ошибке». Минимизация этой функции по параметру $\tilde{w}$ эквивалентна нахождению $\tilde{w}$, для которого $\tilde{w}^{\mathrm{T}} \cdot \tilde{x} \geq 0$. Очевидно, что

$$\frac{\partial J}{\partial \tilde{w}} = \frac{1}{2}(\tilde{x}\,\mathrm{sgn}(\tilde{w}^{\mathrm{T}} \cdot \tilde{x}) - \tilde{x}),$$

где

$$\mathrm{sgn}(\tilde{w}^{\mathrm{T}} \cdot \tilde{x}) = \begin{cases} +1, & \tilde{w}^{\mathrm{T}} \cdot \tilde{x} > 0, \\ 0, & \tilde{w}^{\mathrm{T}} \cdot \tilde{x} = 0, \\ -1, & \tilde{w}^{\mathrm{T}} \cdot \tilde{x} < 0. \end{cases}$$

Замечание о правиле дифференцирования:

$$\frac{\partial J}{\partial \tilde{w}} = \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial w_n} \end{pmatrix}$$

при $\tilde{w} = (w_1, \ldots, w_n)^{\mathrm{T}}$. При $\tilde{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$

$$\frac{\partial(\tilde{w}^{\mathrm{T}} \cdot \tilde{x})}{\partial \tilde{w}} = \frac{\partial(w_1 x_1 + + w_n x_n)}{\partial \tilde{w}} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \tilde{x}\,.$$

При обучении алгоритма желательно минимизировать функцию J. Применение метода градиентного спуска [Васильев, 2002] приводит к следующему выражению для коррекции параметров линейного классификатора:

$$\tilde{w} := \tilde{w} - c \frac{\partial J}{\partial \tilde{w}}\bigg|_{\tilde{w}}$$

(значение производной берётся в текущей точке) или

$$\tilde{w} := \tilde{w} - \frac{c}{2}(\tilde{x}\,\mathrm{sgn}(\tilde{w}^{\mathrm{T}} \cdot \tilde{x}) - \tilde{x}),$$

что фактически совпадает с коррекцией в персептроном алгоритме.

В **алгоритме дробной коррекции** [Ту, Гонсалес, 1978] правило изменения весов

$$\tilde{w} := \tilde{w} + \frac{c\,|\,\tilde{w}^{\mathrm{T}} \cdot \tilde{x}\,|}{\tilde{x}^{\mathrm{T}} \cdot \tilde{x}} \tilde{x} \;\text{ при }\; \tilde{w}^{\mathrm{T}} \cdot \tilde{x} \le 0,$$

которое получается при выборе

$$J(\tilde{w}, \tilde{x}) = \frac{1}{4\tilde{x}^{\mathrm{T}} \cdot \tilde{x}}(|\,\tilde{w}^{\mathrm{T}} \cdot \tilde{x}\,|^2 - |\,\tilde{w}^{\mathrm{T}} \cdot \tilde{x}\,| \cdot \tilde{w}^{\mathrm{T}} \cdot \tilde{x})$$

в качестве функции для минимизации (**ДЗ** *докажите это*!). Алгоритм сходится при $0 < c < 2$. При $c > 1$ после коррекции весов линейный классификатор начинает правильно классифицировать объект, на котором была ошибка (**ДЗ** *докажите это*!). Заметим, что в персепронном алгоритме это не так.

В задаче оптимизации

$$J(\tilde{w}) = -\sum_{\tilde{x}\,:\,\tilde{w}^{\mathrm{T}} \cdot \tilde{x} \le 0} \tilde{w}^{\mathrm{T}} \cdot \tilde{x} \to \min$$

(x пробегает все объекты обучения, для которых $\tilde{w}^{\mathrm{T}} \cdot \tilde{x} \le 0$) получаем коррекцию пакетного режима обучения (с точностью до константы c):

$$\tilde{w} := \tilde{w} - c \frac{\partial J}{\partial \tilde{w}}\bigg|_{\tilde{w}} = \tilde{w} + c \sum_{\tilde{x}\,:\,\tilde{w}^{\mathrm{T}} \cdot \tilde{x} \le 0} \tilde{x}\,.$$

Заметим, что функция J зависит от всей выборки (а не от работы алгоритма на отдельном объекте) и тоже очень «естественная» для применения при построении линейного классификатора: мы учитываем величины ошибок на всех объектах.

ДЗ *Придумайте ещё несколько «естественных» функций, минимизация которых приводит к обучению линейного классификатора. Сравните различные способы обучения на модельных задачах.*

Замечание. На практике часто коэффициент c не полагают равным константе, а делают зависимым от номера j исправления параметров, например $c(j)=1/j$.

Пример. Проиллюстрируем геометрический смысл алгоритма персептрона. Пусть

$$\left.\begin{matrix}\tilde{x}^1=(1,0)^{\mathrm{T}}\\ \tilde{x}^2=(1,1)^{\mathrm{T}}\end{matrix}\right\}\in K_1,\ \left.\begin{matrix}\tilde{x}^3=(1,2)^{\mathrm{T}}\\ \tilde{x}^4=(1,3)^{\mathrm{T}}\end{matrix}\right\}\in K_2.$$

Здесь описания дополнены фиктивным единичным признаком. Домножим описания объектов второго класса на (-1), наша задача свелась к решению системы линейных неравенств

$$(w_1,w_2)\cdot\begin{bmatrix}1 & 1 & -1 & -1\\ 0 & 1 & -2 & -3\end{bmatrix}>(0,0,0,0)$$

(собственно, именно такие системы и решают описанные выше алгоритмы). Персептронный метод решения системы – идти по столбцам, если $\tilde{w}^{\mathrm{T}}\cdot\tilde{x}\le 0$ для столбца $\tilde{x}$, то менять вектор: $\tilde{w}:=\tilde{w}+\tilde{x}$. На рис. 5.2.1 показано, что при $\tilde{w}=(3,3)^{\mathrm{T}}$ неверно классифицирована третья точка (она лежит «не в той» полуплоскости, относительно прямой с положительно-ориентированным вектором нормали $\tilde{w}$), поэтому вектор $\tilde{w}^{\mathrm{T}}$ меняется на $(3,3)+(-1,-2)=(2,1)$ (на рис. 5.2.1 показано сложение этих векторов по правилу параллелограмма). В итоге прямая с положительно-ориентированным вектором нормали $\tilde{w}$ занимает положение так, что все точки, записанные в матрице будут в одной (положительной) полуплоскости. Таким образом, прямая (точнее, нормаль) поворачивается в сторону неверно классифицированной точки и так, качаясь, стремится занять «нужное» положение.

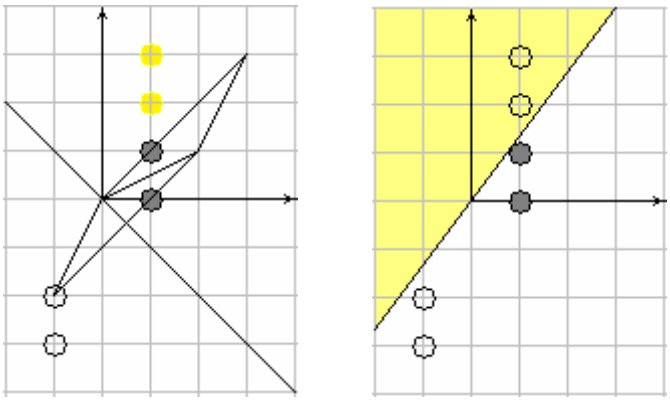


Рис. 5.2.1. Изменение положения прямой (слева). Разделяющая прямая (справа).

Замечание. Предложить алгоритм, который настраивает линейный классификатор (решает нужную систему линейных неравенств) за конечное число шагов достаточно просто. Можно считать, что все коэффициенты гиперплоскости целые (**ДЗ** *обоснуйте*!). Таким образом, различных гиперплоскостей счётное число, поэтому их можно перебирать, пока не встретится «нужная». **ДЗ** *Подумайте об эффективности этого алгоритма. Попробуйте реализовать его на ЭВМ и сравнить с перечисленными выше.*

Замечание. Теорема Новикова даёт оценку числа шагов персептронного алгоритма, которой нельзя воспользоваться на практике (поскольку в неё входит величина δ).

§5.3. Алгоритм потенциальных функций

На процедуре поощрения-наказания основан один из первых методов обучения **алгоритма потенциальных функций**. В этом алгоритме строится потенциал $\Gamma(x)$, который положителен на объектах первого класса и отрицателен на объектах второго класса[1].

Обучение алгоритма потенциальных функций

0. Положить потенциал равным нулю во всех точках пространства: $\Gamma(x) := 0$.

1. Перебрать все объекты x^t обучения.

2. Если объект x^t классифицирован неверно,

при $\Gamma(x^t) \geq 0$ и $y(x) = K_2$ изменить потенциал:

$$\Gamma(x) := \Gamma(x) - K\left(\frac{\rho(x^t, x)}{h}\right),$$

при $\Gamma(x^t) \leq 0$ и $y(x) = K_1$ изменить потенциал:

[1] Формально алгоритм потенциальных функций не является линейным классификатором, но существует (новое) признаковое пространство, где он будет таковым.

$$\Gamma(x) := \Gamma(x) + K\left(\frac{\rho(x^t, x)}{h}\right).$$

(если объект x^t классифицирован верно, то потенциал не меняется).

3. Если все объекты обучения были классифицированы верно, то алгоритм настроен, иначе (применялся п.2) перейти к п.1.

Здесь по-прежнему $K:[0,+\infty)\to[0,+\infty)$ – функция, которая принимает в нуле максимальное значение и является невозрастающей. В методе потенциальных функций такую функцию принято называть **функцией ядра**.

ДЗ *В алгоритме параметр h считается фиксированным. Предложите метод обучения алгоритма потенциальных функций, в котором потенциал ищется в виде*

$$\sum_{x^t\in K_1} K\left(\frac{\rho(x^t, x)}{h_t}\right) - \sum_{x^t\in K_2} K\left(\frac{\rho(x^t, x)}{h_t}\right).$$

Проведите его сравнение с «классическим» потенциальным алгоритмом на модельных данных.

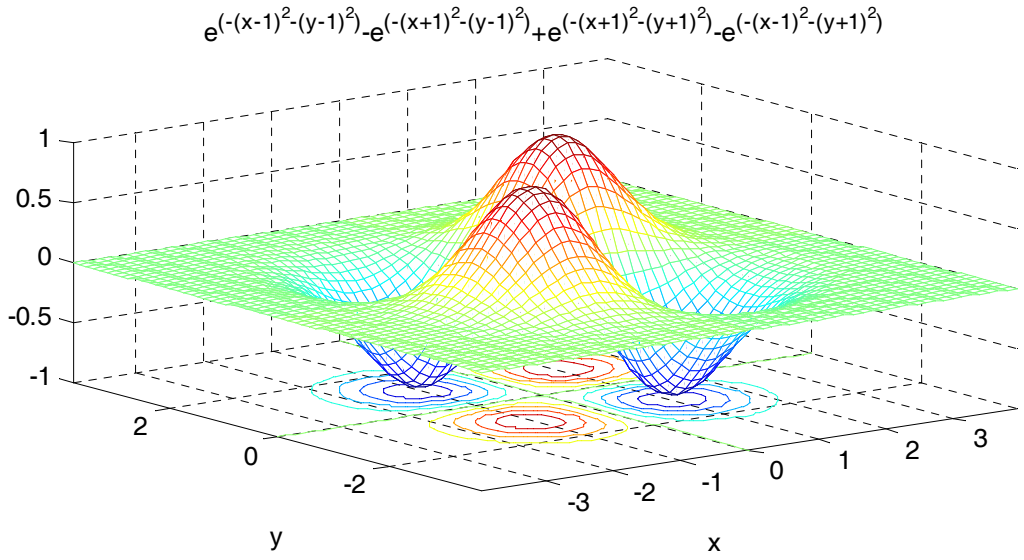


Рис 5.3.1. Окончательный вид потенциала.

Пример. Рассмотрим известную задачу XOR, которая не разрешима линейным классификатором со 100%-й точностью:

$$\tilde{x}^1 = (+1,+1)^{\mathrm{T}} \in K_1,\ \tilde{x}^2 = (-1,+1)^{\mathrm{T}} \in K_2,\ \tilde{x}^3 = (-1,-1)^{\mathrm{T}} \in K_1,\ \tilde{x}^4 = (+1,-1)^{\mathrm{T}} \in K_1.$$

Идём последовательно по объектам, пусть $K(x) = e^{-x^2}$, тогда при евклидовой метрике потенциал изменяется следующим образом: 0,

$$e^{-(f_1(x)-1)^2-(f_2(x)-1)^2} \text{ (1-я коррекция)}$$

$$e^{-(f_1(x)-1)^2-(f_2(x)-1)^2} - e^{-(f_1(x)+1)^2-(f_2(x)-1)^2} \text{ (2-я коррекция)}$$

$$e^{-(f_1(x)-1)^2-(f_2(x)-1)^2} - e^{-(f_1(x)+1)^2-(f_2(x)-1)^2} + e^{-(f_1(x)+1)^2-(f_2(x)+1)^2} \text{ (3-я коррекция)}$$

$$e^{-(f_1(x)-1)^2-(f_2(x)-1)^2} - e^{-(f_1(x)+1)^2-(f_2(x)-1)^2} + e^{-(f_1(x)+1)^2-(f_2(x)+1)^2} - e^{-(f_1(x)-1)^2-(f_2(x)+1)^2}$$

(4-я коррекция).

Последний потенциал верно классифицирует обучающую выборку (см. рис. 5.3.1).

Замечание. Все описанные выше методы обучения линейного классификатора (и алгоритма потенциальных функций) сейчас считаются устаревшими. Причины понятны…

1. Методы работают при условии существования линейного классификатора (гиперплоскости, разделяющей точки в пространстве).

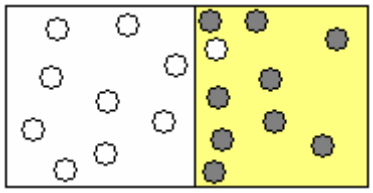


Рис. 5.3.2. Несуществование разделяющей гиперплоскости из-за шумов/выбросов.

2. Методы строят **какой-то** линейный классификатор. Понятно, что не все классификаторы одинаково хороши. Считается, что разделяющая поверхность должна максимизировать **отступ**[1] между классами (расстояние между поверхностью и ближайшей точкой выборки).

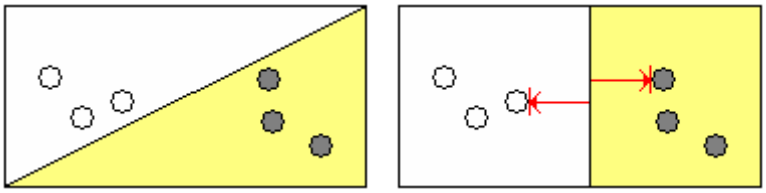


Рис. 5.3.3. Неоптимальная (слева) и оптимальная (справа) разделяющие гиперплоскости.

Все эти недостатки устранены в методе SVM и различных его модификациях. Отметим, что, несмотря на недостатки, сами схемы обучения «поощрение-наказание» и градиентной оптимизации (в конечно счёте, это одна и та же схема) очень широко применяются, особенно при обучении «сложных» классификаторов. Кроме того, построив несколько (пусть даже неоптимальных) линейных классификаторов $\tilde{w}_1,\ldots,\tilde{w}_s$, можно попробовать получить общий на их основе методом усреднения ($\tilde{w}_1+\ldots+\tilde{w}_s$) или голосования:

$$A(x)=\begin{cases} K_1, & \operatorname{sgn}(\tilde{w}_1^{\mathrm{T}}\tilde{x})+\ldots+\operatorname{sgn}(\tilde{w}_s^{\mathrm{T}}\tilde{x})>0, \\ K_2, & \text{иначе.} \end{cases}$$

[1] Здесь употребляется одно из возможных определений термина «отступ» (не все существующие определения эквивалентны).

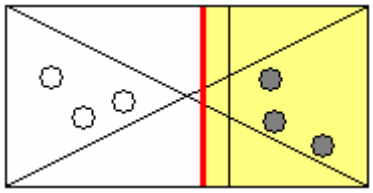


Рис. 5.3.4. Результат усреднения линейных классификаторов.

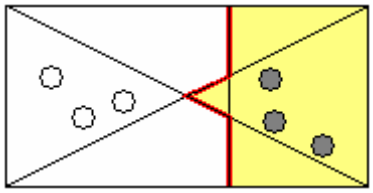


Рис. 5.3.5. Результат голосования линейных классификаторов.

ДЗ *Сравните голосование и усреднение линейных классификаторов на модельных задачах. Какие ещё существуют способы учёта результатов работы нескольких классификаторов?*

§5.4. Алгоритм, основанный на минимизации среднеквадратичной ошибки

Рассмотрим **НСКО-алгоритм**[1] [Ту, Гонсалес, 1978], который часто называют **алгоритмом Хо-Кашьяпа**. Основное его преимущество – наличие критерия линейной разделимости. Как было показано выше, построение линейного классификатора эквивалентно решению системы линейных неравенств

$$X \cdot \tilde{w} > 0,$$

где в матрице X[2] записаны описания объектов, дополненные фиктивным признаком (везде равным единице), причём описания объектов второго класса берутся со знаком минус (неравенство «больше нуля» поэлементное). Заменим её формально на систему линейных равенств и неравенств

$$X \cdot \tilde{w} = \tilde{b} > 0$$

(неравенство $\tilde{b} > 0$ понимаем как поэлементное: все элементы вектора $\tilde{b}$ положительны). Первое равенство выполнено, когда минимума достигает функция

$$J = \frac{1}{2} \| X \cdot \tilde{w} - \tilde{b} \|^2.$$

Заметим, что нельзя решить эту систему равенств, написав $\tilde{w} = X^{-1} \cdot \tilde{b}$, поскольку матрица X может не быть квадратной. Поэтому будем минимизировать функцию J. Нетрудно видеть (**ДЗ** *докажите это*), что

---

[1] НСКО – наименьшая среднеквадратичная ошибка.

[2] Здесь обозначение матрицы совпадает с обозначением пространства допустимых объектов, но это не должно вызвать путаницы. По-сути, для программиста нет абстрактного пространства допустимых объектов, а есть только перечень этих объектов. Кстати, именем «**X**» часто называют матрицу «объект-признак» в различных репозиториях данных; возможны также имена «**xtrain**», «**train**», «**training_set**» и т.д.

$$\frac{\partial J}{\partial \tilde{w}} = X^{\mathrm{T}} \cdot (X \cdot \tilde{w} - \tilde{b}),$$
$$\frac{\partial J}{\partial \tilde{b}} = -(X \cdot \tilde{w} - \tilde{b}).$$

Первую производную можно приравнять к нулю, получим обобщённое решение уравнения $X \cdot \tilde{w} = \tilde{b}$:

$$\tilde{w} = (X^{\mathrm{T}} \cdot X)^{-1} \cdot X^{\mathrm{T}} \cdot \tilde{b} \qquad (5.4.1)$$

(матрицу $(X^{\mathrm{T}} \cdot X)^{-1} \cdot X^{\mathrm{T}}$ называют **обобщённым обращением** матрицы X, в системе MatLab её реализует функция **pinv**). Вторую производную нельзя приравнять к нулю, поскольку минимизация производится при условии $\tilde{b} > 0$, поэтому применяем изменённый шаг градиентного алгоритма

$$\tilde{b} := \tilde{b} + \max[c \cdot (X \cdot \tilde{w} - \tilde{b}), 0], \qquad (5.4.2)$$

где операция max заменяет все отрицательные компоненты левого векторного аргумента на нули. Реализация алгоритма Хо-Кашьяпа заключается в чередовании этапов (5.4.1) и (5.4.2). При наличии линейной разделимости алгоритм сходится при $0 < c \le 2$. Разделяющая гиперплоскость построена, если $X \cdot \tilde{w} > 0$ (это происходит также при $X \cdot \tilde{w} - \tilde{b} = 0$, см. листинг ниже). Если на каком-то шаге выполняется $0 \ne (X \cdot \tilde{w} - \tilde{b}) \le 0$, то линейной разделимости не существует.

Программа для системы MatLab, реализующая НСКО

```
%% Обучающая выборка
X = rand([20, 2]); % объекты
Y = (X(:,1)>X(:,2)); % классы (есть линейная разделимость)

%% Подготовка для визуализации
clf; hold on;

%% Подготовка к экспериментам
b = ones ([size(X,1) 1]); % нач. значения b
X(:,end+1) = b; % фиктивный признак
X(~Y,:) = -X(~Y,:); % инверсия объектов 2го класса

%% Основной цикл алгоритма
w = 0; % для проверки первого условия
while ~all(X*w > 0) % пока не найдено решение
    w = (X'*X)\(X'*b); % шаг (5.4.1)
    e = X*w - b;
    if all(e==0) % для страховки: all(abs(e)<eps)
        % найдено решение
        break;
    end;
    if all(e<=0) % для страховки: if all(e<=0)
        % нет решения
        disp('-');
        break;
```

```
    end;
    b = b + max(e,0); % шаг (5.4.2)
    % рисуем ГП
    plot ([0 1], [-w(3)/w(2) -(w(3)+w(1))/w(2)], 'g');
end;

%% Визуализация обучения и ответа
plot ([0 1], [-w(3)/w(2) -(w(3)+w(1))/w(2)], 'k', 'LineWidth', 2);
scatter(X(Y,1), X(Y,2),30, 's', 'r'); % 1й класс
scatter(-X(~Y,1), -X(~Y,2),30, 'o', 'b', 'filled'); % 2й класс
```

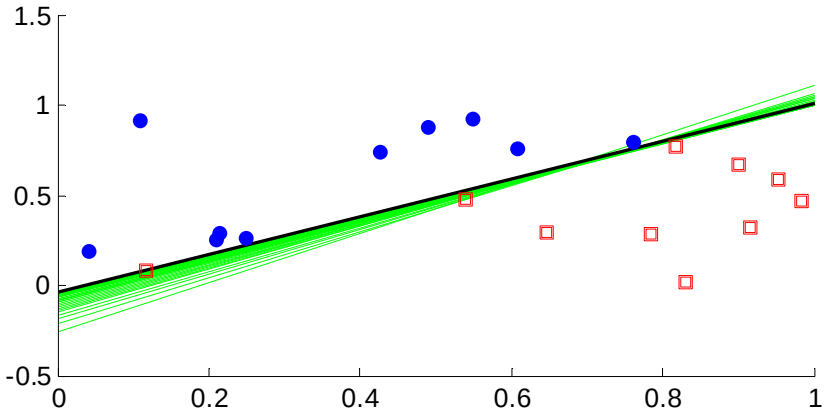


Рис. 5.4.1. Картинка, сгенерированная программой.

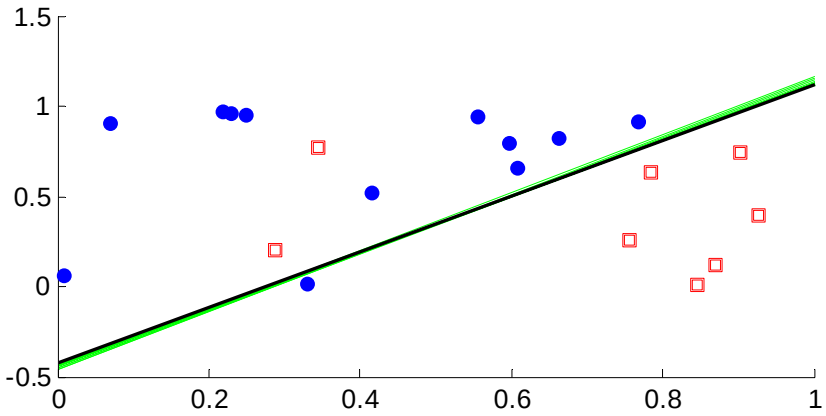


Рис 5.4.2. Случай линейно неразделимых классов.

ДЗ *При каких условиях существует обратная к* $X^{\mathrm{T}} \cdot X$ *матрица?*

Замечание. Обратите внимание, что мы научились решать уравнения вида $X \cdot \tilde{w} = \tilde{b}$. Часто упрощённо (и неправильно) их «учат» решать следующим образом: домножим обе части на X^{T}, получим уравнение $(X^{\mathrm{T}} \cdot X) \cdot \tilde{w} = X^{\mathrm{T}} \cdot \tilde{b}$ с квадратной матрицей, дальше домножаем на обратную к ней. В «умных книжках» рассмотренную нами функцию J называют **квадратом нормы невязки**.

ДЗ *Проведите следующий очень познавательный эксперимент. Решите НСКО-алгоритмом задачу XOR, в которой одна пара противоположных вершин квадрата на плоскости принадлежит первому классу, а вторая пара – второму классу. Вот нужное задание классов в системе MatLab:* `X = [1,1; 1,0; 0,1; 0,0]; Y = X(:,1)==X(:,2);`. *Теперь добавьте шум* (`X = X + rand([4 2])/100;`) *и повторите эксперимент. Почему построенные гиперплоскости так сильно отличаются?*

Обратите внимание, что во многих модельных задачах при решении их НСКО-алгоритмом гиперплоскость сразу занимает «примерно нужное» положение, а потом корректируется до наступления 100%-й разделимости.

Про линейные классификаторы см. [Ту, Гонсалес, 1978].

§5.5. Метод опорных векторов

Идея метода уже была озвучена: поскольку в случае линейно разделимой выборки существует много разделяющих гиперплоскостей, необходимо выбрать ту, которая «максимально хорошо» разделяет классы, т.е. максимизирует отступ (зазор). Расстояние между гиперплоскостью и ближайшей точкой (объектом обучающей выборки) должно быть максимально. Рассмотрим подробнее формализацию задачи.

В задаче с двумя непересекающимися классами, $Y = \{+1,-1\}$, и обучающей выборкой $\{x^t\}_{t=1}^m$, объекты которой заданы своими признаковыми описаниями $\tilde{x}^1,\ldots,\tilde{x}^m \in \mathbf{R}^n$ (здесь рассматриваем непополненное признаковое пространство) необходимо построить линейный классификатор, т.е. гиперплоскость, разделяющую объекты двух классов, с уравнением $\tilde{w}^{\mathrm{T}}\tilde{x} + b = 0$.

Нетрудно показать [Burges, 1998], [Воронцов], что в случае линейно разделимой выборки построение гиперплоскости, максимизирующей отступ, эквивалентно решению следующей задачи оптимизации:

$$\begin{cases} \dfrac{1}{2}\|\tilde{w}\|^2 = \dfrac{1}{2}\tilde{w}^{\mathrm{T}}\tilde{w} \to \min \\ y(x^t)(\tilde{w}^{\mathrm{T}}\tilde{x}^t + b) \geq 1, \qquad t \in \{1,2,\ldots,m\}, \end{cases}$$

(первая строка системы формализует максимизацию отступа, а вторая строка – линейную разделимость выборки, множитель 1/2 традиционно вводят для удобства), которая сводится к следующей задаче квадратичного программирования:

$$\begin{cases} L = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda_i\lambda_j y(x^i)y(x^j)\tilde{x}^{i\,\mathrm{T}}\tilde{x}^j - \sum_{i=1}^{m}\lambda_i \to \min\limits_{\lambda_1,\ldots,\lambda_m} \\ \lambda_t \ge 0, \qquad t \in \{1,2,\ldots,m\}, \\ \sum_{t=1}^{m}\lambda_t y(\tilde{x}^t) = 0 \end{cases}$$

Сведение производится с помощью метода множителей Лагранжа. После нахождения ответа $\lambda_1,\ldots,\lambda_m$ вектор $\tilde{w}$ вычисляется по формуле

$$\tilde{w} = \sum_{t=1}^{m}\lambda_t y(\tilde{x}^t)\tilde{x}^t .$$

В этом выражении лишь немногие λ_t отличны от нуля. Соответствующие им объекты определяют направление вектора нормали, а следовательно, разделяющей гиперплоскости. Геометрически эти объекты, которые называют **опорными**, лежат на границе разделяющей полосы. См. рис. 5.5.1.

Рис. 5.5.1. Оптимальная разделяющая гиперплоскость, полоса между классами, определяемая ею, и опорные объекты.

Для вычисления свободного члена уравнения разделяющей гиперплоскости используют любую пару «противоположных» опорных объектов $\tilde{x}^r$, $\tilde{x}^s$:

$$b = -\frac{1}{2}\cdot\tilde{w}^{\mathrm{T}}\cdot(\tilde{x}^r - \tilde{x}^s),$$

$y(\tilde{x}^r) = -y(\tilde{x}^s)$ (классифицированы по-разному), $\lambda_r > 0$, $\lambda_s > 0$.

Если линейной разделимости нет, то задача меняется:

$$\begin{cases} \frac{1}{2}\|\tilde{w}\|^2 + C\sum_{t=1}^{m}\xi_t \to \min \\ y(x^t)(\tilde{w}^{\mathrm{T}}\tilde{x}^t + b) \ge 1 - \xi_t, \quad t \in \{1,2,\ldots,m\}, \\ \xi_t \ge 0, \qquad t \in \{1,2,\ldots,m\}, \end{cases}$$

Мы разрешаем нарушение равенства $y(x^t)(\tilde{w}^{\mathrm{T}}\tilde{x}^t + b) \ge 1$, при этом точка, которая его нарушила, может

1) оказаться всё ещё с нужной стороны гиперплоскости,

2) оказаться на «чужой» полуплоскости.

Это показано на рис. 5.5.2.

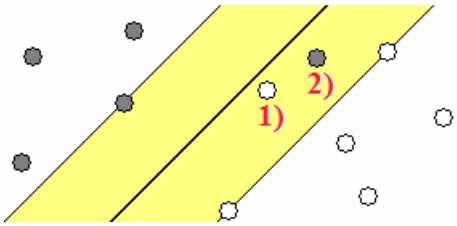


Рис. 5.5.2. Точки, которые нарушают неравенство $y(x^t)(\tilde{w}^{\mathrm{T}}\tilde{x}^t + b) \geq 1$.

Поэтому в задачу вводятся «фиктивные переменные» ξ_t, которые показывают, насколько точка залезла за пределы своей области. Оказывается, что наша новая задача сводится к «почти такой же», как и в линейно разделимом случае, задаче квадратичного программирования:

$$\begin{cases} L \to \min\limits_{\lambda_1,\ldots,\lambda_m} \\ 0 \leq \lambda_t \leq C, \qquad t \in \{1,2,\ldots,m\}, \\ \sum\limits_{t=1}^{m} \lambda_t y(\tilde{x}^t) = 0, \end{cases}$$

Заметим, что множитель C у слагаемого $\sum\limits_{t=1}^{m} \xi_t$ регулирует, какое из двух слагаемых в минимизируемой функции мы «больше хотим» минимизировать: отвечающее за отступ или суммарную ошибку.

Функция L зависит от попарных скалярных произведений $\tilde{x}^{i\,\mathrm{T}}\tilde{x}^j$, $i \in \{1,2,\ldots,m\}$, $j \in \{1,2,\ldots,m\}$. Поэтому только этой информации достаточно знать об обучающей выборке (и, естественно, верную классификацию объектов обучающей выборки).

Допустим, что мы хотим строить гиперплоскость не в исходном признаковом пространстве, а в другом, которое индуцируется функцией $\psi : X \to \Psi$ (т.е. в пространстве Ψ с объектами обучающей выборки $\psi(\tilde{x}^1),\ldots,\psi(\tilde{x}^m)$). Тогда нам достаточно знать величины $\psi(\tilde{x}^i)^{\mathrm{T}}\psi(\tilde{x}^j)$, $i \in \{1,2,\ldots,m\}$, $j \in \{1,2,\ldots,m\}$. Их часто удаётся вычислять, используя понятие **ядра** – функции $K(\tilde{x},\tilde{z}) : X \times X \to \mathbf{R}$, которая представима в виде $K(\tilde{x},\tilde{z}) = \psi(\tilde{x})^{\mathrm{T}}\psi(\tilde{z})$ для некоторой функции ψ. Условия, при которых функция является ядром, определены в теореме Мерсера (см. [Хайкин, 2006], [Воронцов]).

Пример. Рассмотрим задачу XOR: $\tilde{x}^1=(+1,+1)^{\mathrm{T}}\in K_1$, $\tilde{x}^2=(-1,+1)^{\mathrm{T}}\in K_2$, $\tilde{x}^3=(-1,-1)^{\mathrm{T}}\in K_1$, $\tilde{x}^4=(+1,-1)^{\mathrm{T}}\in K_2$ и функцию $K(\tilde{x},\tilde{z})=(\tilde{x}^{\mathrm{T}}\tilde{z})^2$. Из цепочки равенств

$$K(\tilde{x},\tilde{z})=(\tilde{x}^{\mathrm{T}}\tilde{z})^2=((x_1,x_2)\cdot(z_1,z_2)^{\mathrm{T}})^2=(x_1z_1+x_2z_2)^2=x_1^2z_1^2+2x_1x_2z_1z_2+x_2^2z_2^2=$$
$$=(x_1^2,\sqrt{2}x_1x_2,x_2^2)\cdot(z_1^2,\sqrt{2}z_1z_2,z_2^2)^{\mathrm{T}}$$

следует, что применение ядра $K(\tilde{x},\tilde{z})=(\tilde{x}^{\mathrm{T}}\tilde{z})^2$ эквивалентно следующему изменению признакового пространства:

$$(x_1,x_2)\to(x_1^2,\sqrt{2}x_1x_2,x_2^2)$$

$(+1,+1)$	$(1,+\sqrt{2},1)$
$(+1,-1)$	$(1,-\sqrt{2},1)$
$(-1,+1)$	$(1,-\sqrt{2},1)$
$(-1,-1)$	$(1,+\sqrt{2},1)$

В новом пространстве $\{(x_1',x_2',x_3')\}$ оптимальной разделяющей гиперплоскостью будет гиперплоскость $x_2'=0$. Поэтому разделяющая поверхность в исходной задаче запишется как $\sqrt{2}x_1x_2=0$.

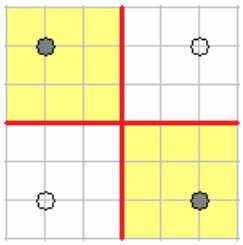


Рис. 5.5.3. Решение задачи XOR методом SVM с ядром $K(\tilde{x},\tilde{z})=(\tilde{x}^{\mathrm{T}}\tilde{z})^2$.

Интерпретируя функцию $\sqrt{2}x_1x_2$ как оценку принадлежности классам, можно построить следующий график оценки принадлежности (см. рис. 5.5.4 и ср. с рис. 5.3.1).

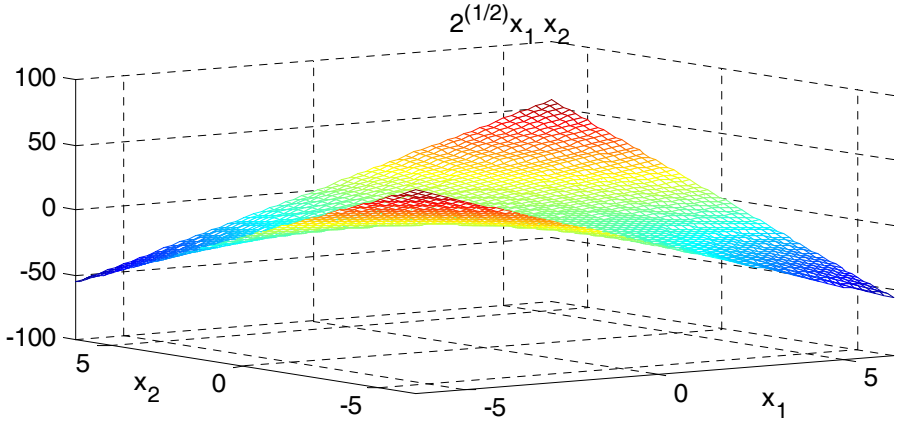


Рис. 5.5.4. Функция $\sqrt{2}x_1x_2$.

На практике часто используются следующие ядра:

$$K(\tilde{x},\tilde{z}) = (\tilde{x}^{\mathrm{T}}\tilde{z})^d,$$

$$K(\tilde{x},\tilde{z}) = (\tilde{x}^{\mathrm{T}}\tilde{z}+1)^d \text{ (полиномиальное ядро),}$$

$$K(\tilde{x},\tilde{z}) = \exp\left(-\frac{\|\tilde{x}-\tilde{z}\|^2}{2\sigma^2}\right) \text{ (RBF-ядро),}$$

$$K(\tilde{x},\tilde{z}) = \tanh\left(c_1\tilde{x}^{\mathrm{T}}\tilde{z}+c_2\right) \text{ (персептронное ядро).}$$

Замечание. Мы так и не рассказали самое главное: как решать задачу квадратичного программирования[1]. Лучше об этом почитать в литературе по методам оптимизации. Ниже будут описаны пакеты прикладных программ, в которых реализован SVM[2]. При их использовании важно понимать основную концепцию и смысл всех параметров (в частности, смысл функции ядра).

Про использование SVM в задачах классификации и регрессии см. также [Gunn, 1998].

[1] В задание практикума на ЭВМ для студентов 3го курса ВМК МГУ не входит реализация SVM «с нуля».

[2] В зависимости от версии учебного пособия пакеты прикладных программ описаны в следующих главах или во второй части пособия.

Глава 6. НЕЙРОННЫЕ СЕТИ

§6.1. Определение нейронной сети

Не будем утомлять читателя сведениями из биологии и описанием строения нейронов, а сразу перейдём к сути. Рассматривается признаковая задача классификации (потом перейдём к регрессии). Напомним, что линейный классификатор ищет целевую функцию $y: X \to Y$ в виде

$$\varphi(w_0 f_0(x) + w_1 f_1(x) + \ldots + w_n f_n(x)),$$

где $f_0(x) = 1 \ \ \forall x \in X$,

$$\varphi(a) = \begin{cases} 1, a \geq 0, \\ 0, a < 0, \end{cases}$$

(это для задачи с двумя классами $Y = \{0,1\}$). Графически такой классификатор показан на рис. 6.1.1. Его часто называют **персептроном.**

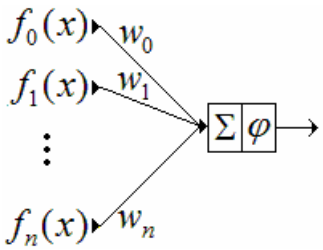


Рис. 6.1.1. Персептрон.

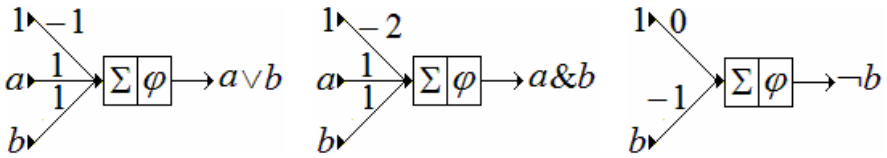


Рис. 6.1.2. Дизъюнкция, конъюнкция и отрицание.

Персептрон очень похож на функциональный элемент[1]. Более того, стандартные булевы операции реализуются персептроном (см. рис. 6.1.2). Поэтому подобно функциональным элементам из них можно строить схемы. В нейроматематике такие конструкции называются **нейронными сетями (нейросетями).** На рис. 6.1.3 показана нейросеть, состоящая из двух персептронов.

Этой нейросети соответствует функция

$$a(f_1(x), f_2(x)) = \varphi(-1 - 2\varphi(-2 + f_1(x) + f_2(x)) + f_1(x) + f_2(x)),$$

[1] Поскольку данное учебное пособие предназначено, прежде всего, для студентов 3 курса факультета ВМК МГУ, которые имеют опыт синтеза схем из функциональных элементов, использование этого термина существенно упрощает объяснение принципа работы нейронных сетей. См. [*Яблонский С.В.* Введение в дискретную математику: Учебное пособие для вузов. / Под ред. В.А. Садовничего. – 3-е изд., стер. – М.: Высш.шк.; 2001. – 384 с.].

поэтому **нейросети – это представления целевых функций в виде суперпозиции элементарных** (реализуемых отдельными нейронами, сложением, умножением на константу)**!** Нетрудно убедиться, что

$$a(0,0) = \varphi(-1-2\varphi(-2)) = \varphi(-1) = 0,$$
$$a(0,1) = a(1,0) = \varphi(-1-2\varphi(-1)+1) = \varphi(0) = 1,$$
$$a(1,1) = \varphi(-1-2\varphi(0)+2) = \varphi(-1) = 0.$$

Таким образом, этой нейросетью разрешима задача XOR. В некотором смысле, она не разрешима линейным классификатором, но разрешима двумя линейными классификаторами, если их соединить, как показано на рис. 6.1.3. Решение стало возможным, поскольку произошло **изменение признакового пространства.** Посмотрим (см. табл. 6.1.1), что поступает на входы первого и второго нейронов при предъявлении контрольных объектов задачи XOR (константный вход не указан). Первый нейрон «видит» задачу XOR, а вот второй ещё одно значение для каждого объекта, т.е. как бы **ещё один признак**, который соответствует ответу первого нейрона. Второй нейрон решает уже задачу с линейно разделимой выборкой (см. рис. 6.1.4).

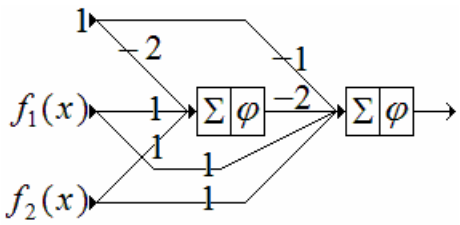


Рис. 6.1.3. Нейросеть, решающая задачу XOR.

0 0 0 1 1 0 1 1	0 0 0 0 1 0 1 0 0 1 1 1

Табл. 6.1.1. Входы нейронов сети рис. 6.1.3 (слева – входы первого, справа – второго).

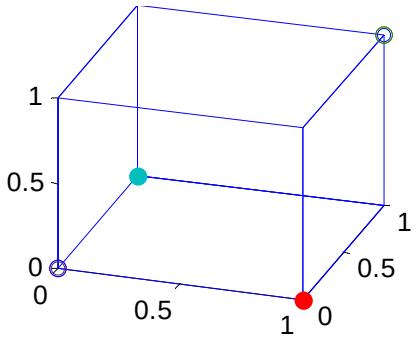


Рис. 6.1.4. Какими «видит» объекты задачи XOR второй нейрон.

На рис. 6.1.5 показана **двухслойная нейросеть. В первом (скрытом) слое** у неё три **нейрона,** а во **втором (выходном) слое –** два нейрона. Каждая дуга имеет определённый вес, с которым суммируются **входы нейрона** (см.

рис. 6.1.1), затем применяется **функция активации** (на рис. 6.1.1 это φ), после чего **сигнал идёт дальше по нейросети**. Такая нейросеть вполне может решать задачи с двумя пересекающимися классами. Если первый нейрон выходного слоя **активируется** (выдаёт положительное значение), то объект, признаковое описание которого дано на вход сети, она относит к первому классу, а если второй – относит ко второму. Основной вопрос: как настроить веса сети (**обучить нейросеть**), чтобы она правильно классифицировала объекты?

Замечание. Нейросеть на рис. 6.1.5 иногда называют трёхслойный, поскольку считают ещё **нулевой (входной) слой** $f_0(x)$, $f_1(x)$,…, $f_n(x)$.

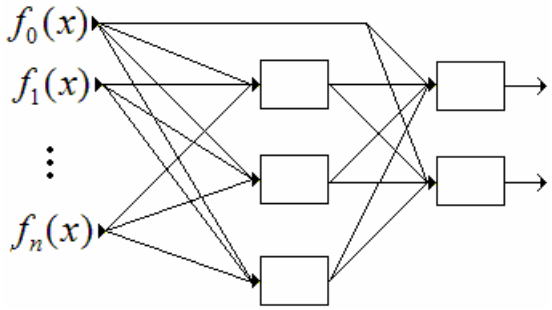


Рис. 6.1.5. Пример многослойной нейросети.

Наиболее распространены **многослойные нейросети**, в которых нейроны располагают по слоям, нейроны первого слоя на вход получают значения всех признаков (и одного выделенного константного), а нейроны каждого последующего получают значения всех нейронов предыдущего слоя, естественно с некоторыми весами, а также выделенный константный признак, и передают своё значение всем нейронам следующего слоя. Нейроны последнего слоя выдают ответ. Пример такой сети показан на рис. 6.1.5, а на рис. 6.1.3 показана сеть, которая не является многослойной[1]. Многослойные нейросети, состоящие из персептронов, называются **многослойными персептронами**.

ДЗ *Докажите, что*

1. Любая булева функция представима в виде двухслойного персептрона.

2. В любой признаковой (*в* $\mathbf{R}^n$) *задаче классификации* (*с конечной обучающей выборкой*) *трёхслойный персептрон может быть настроен на контрольной выборке со 100%-й точностью.*

Нейросеть не обязательно строить из персептронов (т.е. использовать пороговую функцию активации φ). Более того, каждый нейрон может иметь свою функцию активации, но она фиксируется, а вот веса изменяются с целью улучшения качества работы нейросети. Процедура изменения весов называется

[1] Иногда и такие сети называют «многослойными». Сети на рис. 6.1.3 и 6.1.5 объединяет то, что все они являются сетями **прямого распространения сигнала**, т.е. выходы нейронов не соединены со входами нейронов предыдущих слоёв.

обучением нейросети. Как правило, применяют следующие функции активации

Название	Определение	MatLab-код
Функция Хэвисайда (пороговая, функция скачка)	$\theta(z)=\begin{cases}1, & z\ge 0,\\ 0, & z<0.\end{cases}$ Иногда используют $\operatorname{sgn}(z)$	`z>=0` `sign(z)`
Кусочно-линейная	$\lambda(z)=\begin{cases}z, & \lvert z\rvert<1,\\ z/\lvert z\rvert, & \lvert z\rvert\ge 1.\end{cases}$	`kl = @(z) (z>=1)-(z<=-1)+(abs(z)<1).*z`
Сигмоидная функция	$S(z)=\dfrac{1}{1+e^{-z}}$	`s = @(x) 1./(1+exp(-z))`
Гиперболический тангенс	$2S(2z)-1=\dfrac{1-e^{-2z}}{1+e^{-2z}}$	`t = @(x) 2*s(2*x)-1`
Логарифмическая функция	$\log(z+\sqrt{z^2+1})$	`g = @(x) log(x+sqrt(x^2+1))`
	$v(z)=\dfrac{z}{\sqrt{z^2+1}}$	`v = @(z) z./(sqrt(1+z.^2))`
Тождественная функция	z	`l = @(x) x`

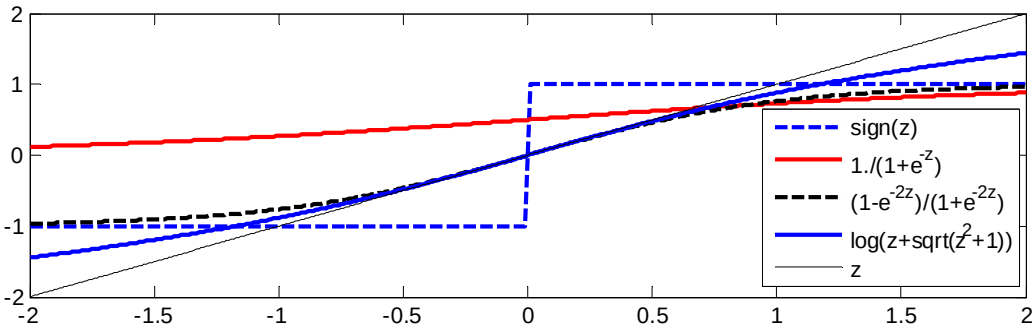


Рис. 6.1.6. Различные функции активации.

Есть ещё **стохастические нейроны**, например с функцией активации

$$SP(z)=\begin{cases}1, & \text{с вероятностью } P=\dfrac{1}{1+e^{-z}},\\ 0, & \text{с вероятностью } (1-P).\end{cases}.$$

Некоторые методы настройки нейросетей ориентированы только на гладкие функции активации. Обратите внимание, что

$$S'(z)=S(z)(1-S(z)),\ v'(z)=v^3(z)/z^3.$$

Теперь уже должно быть понятно, как нейросеть решает задачи регрессии. Отметим только, что многое зависит от нейронов последнего слоя: если их функция активации выдаёт значения из отрезка, то такая сеть по

определению не может решать задачи регрессии, в которых значения целевой функции определены на всей вещественной оси. В этом случае часто используют тождественную функцию активации для нейронов последнего слоя.

Замечание. Обратите внимание, что многослойная сеть неявно делает преобразования признакового пространства. Фрагмент нейросети без первого скрытого слоя ничего не знает о значениях признаков... он «видит» только то, что даёт на выход первый слой. Аналогично, второй слой преобразует информацию от первого, и начальное признаковое описание «превращается» в выход второго слоя и т.д. В конце концов, признаковое описание превращается в выход нейросети. Так в процессе работы сети меняется начальная информация.

§6.2. Алгоритм обратного распространения ошибки

Опишем один из самых популярных алгоритмов настройки нейронных сетей. Будем рассматривать **только дифференцируемые функции активации** (каждый нейрон может иметь свою, но для простоты все их обозначаем символом φ). Рассмотрим нейрон сети (см. рис. 6.2.1). Будем считать, что это нейрон «k» (это не число, а абстрактный символ, «имя нейрона», ниже все индексы будут такими именами, чтобы не связываться со сложными обозначениями).

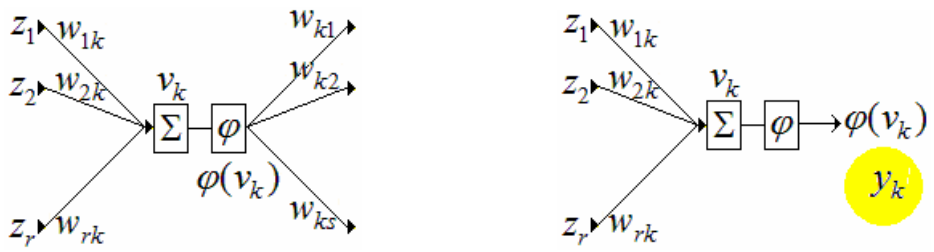


Рис. 6.2.1. Нейрон сети скрытого слоя (слева) и последнего слоя (справа).

На вход нейрон получает значения $z_1, z_2, \ldots, z_r$ (от нейронов предыдущего слоя или входного, т.е. значения признаков), затем эти значения суммируются с соответствующими весами: $v_k = w_{1k} z_1 + w_{2k} z_2 + \ldots + w_{rk} z_r$, затем применяется функция активации, полученное значение $\varphi(v_k)$ передаётся нейронам следующего слоя с весами $w_{k1}, w_{k2}, \ldots, w_{ks}$ или подаётся на выход, если рассматриваемый нейрон лежит в последнем слое. Это описан **прямой ход вычислений нейросети**, по которому она функционирует.

Опишем теперь **обратный ход** (который применяется для настройки сети). Если нейрон «k» лежит в последнем слое, то ошибка на нём известна, поскольку известен верный ответ (на элементе обучающей выборки), который мы обозначим y_k. В **методе обратного распространения ошибки** градиентно минимизируется функционал **энергии ошибки**

$$\frac{1}{2}\sum(\varphi(v_k)-y_k)^2,$$

в котором суммирование (по k) идёт по всем нейронам последнего слоя. Каждый нейрон вносит свою ошибку в сумму: $e_k=\varphi(v_k)-y_k$. Его веса меняют методом градиентного спуска по формуле:

$$w_{ik}:=w_{ik}-\eta e_k\varphi'(v_k)z_k$$

(обоснования см. в [Воронцов], [Хайкин, 2006], хотя их просто вывести самостоятельно). Если рассматриваемый нейрон находится в скрытом (среднем) слое, то для него считаем его ошибку по формуле

$$e_k=(w_{k1}\cdot e_1\cdot\varphi'(v_1)+w_{k2}\cdot e_2\cdot\varphi'(v_2)+\ldots+w_{ks}\cdot e_s\cdot\varphi'(v_s)). \qquad (6.2.1)$$

Внимание! Здесь индексы имеют «символический характер»: e_k – ошибка рассматриваемого нейрона «k», который соединён связями с нейронами следующего слоя (с весами $w_{k1},w_{k2},\ldots,w_{ks}$), которые имеют ошибки $e_1,e_2,\ldots,e_s$; $\varphi'(v_1),\varphi'(v_2),\ldots,\varphi'(v_s)$ – это производные их функций активаций. Веса нейрона пересчитываются по той же формуле:

$$w_{ik}:=w_{ik}-\eta e_k\varphi'(v_k)z_k.$$

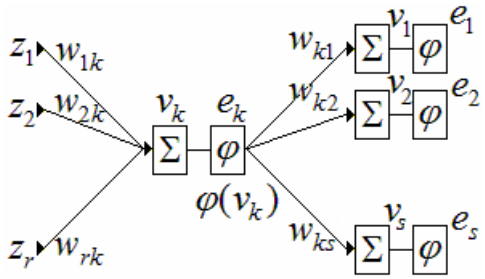


Рис 6.2.2. Обратный ход.

Метод обратного распространения ошибки заключается в чередовании прямого хода и обратного: сначала прямым ходом считаются выходы нейронной сети, определяются ошибки последнего слоя, затем обратным ходом определяются ошибки всех нейронов, одним шагом градиентного метода корректируются веса связей. Потом опять всё повторяется до тех пор, пока энергия ошибки не начнёт меняться слабо.

Применение формулы (6.2.1) и рис. 6.2.2 поясняют название «обратный ход»: сеть как бы начинает работать в обратном направлении.

Достоинства метода	Недостатки метода
Относительная простота программирования	Структура сети фиксирована в процессе обучения (хотя можно придумать эвристику изменения структуры)
Легко обобщается на различные архитектуры	Медленная сходимость; **паралич сети** (когда все параметры начинают слабо

	изменяться, хотя ещё не достигнут локальный минимум энергии ошибки)

Приведём несколько советов по использованию алгоритма обратного распространения ошибки [Хайкин, 2006]:

1. Последовательный режим обучения сети считается предпочтительнее пакетного.
2. Следующий пересчёт весов сети делают для контрольного объекта, который существенно отличается от объекта, использованного на предыдущей итерации.
3. Часто советуют применять функцию активации $1.7159 \cdot \tanh(2z/3)$.
4. Целевые значения должны лежать внутри области значений функций активации выходных нейронов.
5. Лучше предварительно обработать данные: устранить корреляцию признаков, отцентрировать, нормализовать.
6. Для борьбы с параличом иногда **перетряхивают веса**, т.е. меняют их на небольшие случайные значения, а также используют «крутые» функции активации (см. п. 3).

ДЗ *Проверьте, действительно ли следование этим рекомендациям ускоряет настройку нейронной сети*?

При применении формулы адаптации весов $w_{ik} := w_{ik} - \eta_{ik}\delta_{ik}$, где δ_{ik} – некоторое выражение (в методе обратного распространения – $e_k\varphi'(v_k)z_k$), а η_{ik} – темп обучения (в данном случае добавляем индекс, чтобы подчеркнуть, что он свой у каждого параметра), иногда эффективна эвристика **delta-bar-delta**[1]. Основная идея – разумно варьировать темп обучения каждого параметра:

1) если значение δ_{ik} имеет один знак несколько итераций подряд, то следует увеличить темп обучения η_{ik},

2) если значение δ_{ik} имеет разный знак на нескольких последовательных итерациях, то следует уменьшить темп обучения η_{ik}.

Неформально: если куда-то шагаем, но никак не доходим, то надо делать «шаг шире», а если постоянно «перешагиваем» точку оптимума, то надо уменьшить шаг.

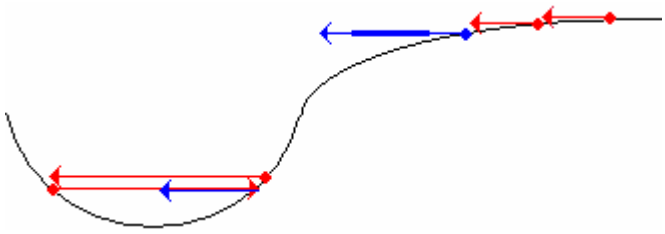


Рис. 6.2.3. Идея эвристики delta-bar-delta.

[1] Так называется реализация алгоритма обратного распространения с описанной ниже эвристикой; библиографию и ссылки см. в [Хайкин, 2006].

§6.3. Настройка нейронной сети в системе MatLab

Покажем, насколько просто реализовывать алгоритм обратного распространения в системе MatLab. Обучим двухслойную нейронную сеть на данных задачи XOR. В первом слое – два элемента, в выходном – один. Функция активации всех нейронов сигмоидная. Несмотря на то, что в последнем слое у выходного нейрона функция активации принимает значения из отрезка [0,1], этой нейросетью можно решать задачи классификации, сравнивая выходное значение с порогом 0.5. Метки одного класса равны 0, другого – 1, поэтому их можно считать регрессионными метками, а выходное значение интерпретировать в терминах «степени принадлежности классу».

Ниже приведён код для системы MatLab.

Настройка нейронной сети

```
% функция активации
S  = @(z) 1./(1+exp(-z));
% её производная
Sd = @(z) S(z).*(1-S(z));

% выборка (XOR)
X = [0 0; 1 1; 0 1; 1 0];
Y = [0 0 1 1]';

% параметры сети
w1 = rand([3 1]);
w2 = rand([3 1]);
w3 = rand([3 1]);

Niter = 2000; % число итераций (Нет критерия останова!)
Er = zeros([1 Niter]); % ошибка по итерациям на одном из объектов

for j=1:Niter
    for i = 1:size(X,1) % по объектам контроля
        % ПРЯМОЙ ХОД
        x = X(i,:); % текущий объект
        v1 = [1 x]*w1; % первый нейрон
        v2 = [1 x]*w2; % второй нейрон
        v3 = [1 S(v1) S(v2)]*w3; % третий нейрон
        out = S(v3); % выход
        % ОБРАТНЫЙ ХОД
        e3 = out - Y(i); % ошибка на выходе
        e1 = w3(2)*e3*Sd(v3);
        e2 = w3(3)*e3*Sd(v3);

        w3 = w3 - e3.*Sd(v3).*[1 S(v1) S(v2)]';
        w1 = w1 - e1.*Sd(v1).*[1 x]';
        w2 = w2 - e2.*Sd(v2).*[1 x]';
    end; % по объектам контроля
    Er(j) = abs(e3); % это ошибка ДО последней коррекции
end;
```

```
% функция, реализуемая сетью
NS = @(x) S([S(x*w1(2:end)+w1(1).*ones([size(x,1) 1]))
S(x*w2(2:end) + w2(1).*ones([size(x,1) 1]))]*w3(2:end) +
w3(1).*ones([size(x,1) 1]))

% визуализация
x = -0.5:0.05:1.5;
y = -0.5:0.05:1.5;
[x y] = meshgrid(x,y);
z = NS([x(:) y(:)]);% так просто вычисляются выходы сети!
z = reshape(z, size(x));
mesh(x,y,z)
hold on;
scatter3(X(:,1),X(:,2),Y,50,Y,'filled')
```

На рис. 6.3.1 показано, как убывает ошибка приближения метки четвёртого объекта в процессе настройки. Было проведено три эксперимента: два со случайными начальными приближениями весов из отрезка [0,1], а третий – с нулевыми начальными приближениями. **ДЗ** *Исследуйте поведение функции энергии ошибки. Почему функции на приведённом графике сначала* (*на отрезке* [1, 10]) *резко возрастают*? *Верно ли, что при нулевых начальных приближениях сеть настраивается медленней*?

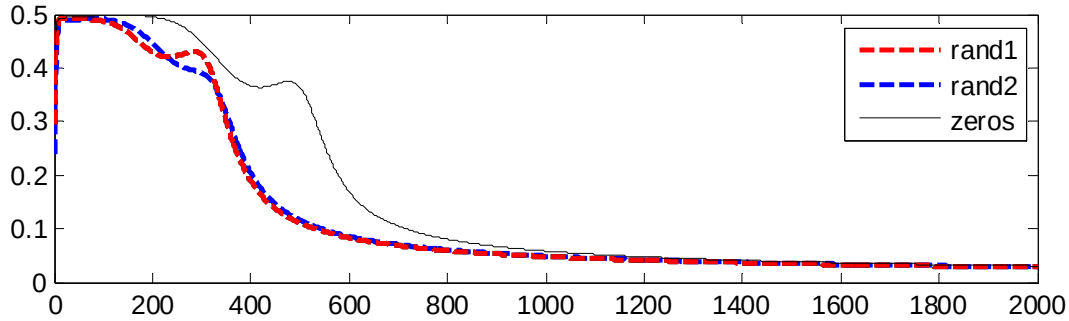


Рис. 6.3.1. Ошибка приближения (на одном из объектов) от числа итераций.

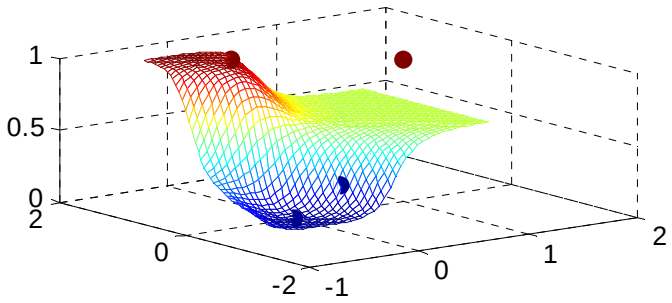


Рис. 6.3.2. Значения функции, реализуемой нейросетью при первом запуске.

На рис. 6.3.2 и рис. 6.3.3 показана функция, реализуемая нейросетью после 1000 итераций. Обратите внимание, что не всегда происходит настройка

на обучающую выборку! **ДЗ** *Объясните это явление. Попробуйте увеличить число итераций. Выработайте критерии остановки.*

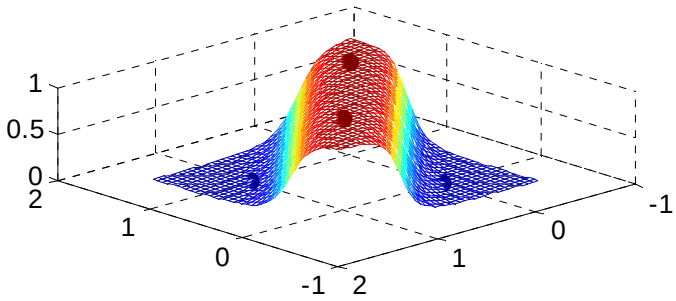


Рис. 6.3.3. Значения функции, реализуемой нейросетью при втором запуске.

На рис. 6.3.4 показана функция, реализуемая нейросетью с другой структурой (см. рис. 6.1.3). Обратите внимание, что **функциональность сети существенно зависит от её структуры! ДЗ** *Убедитесь в этом, проведя серию экспериментов.*

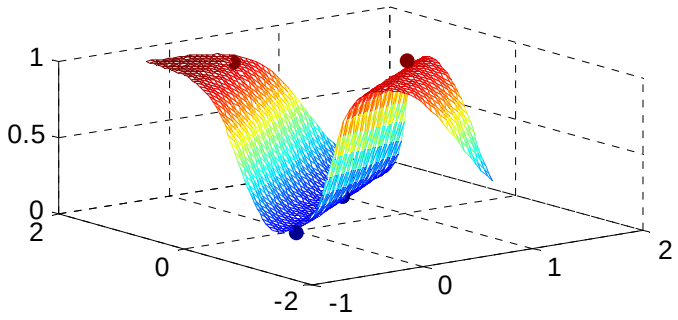
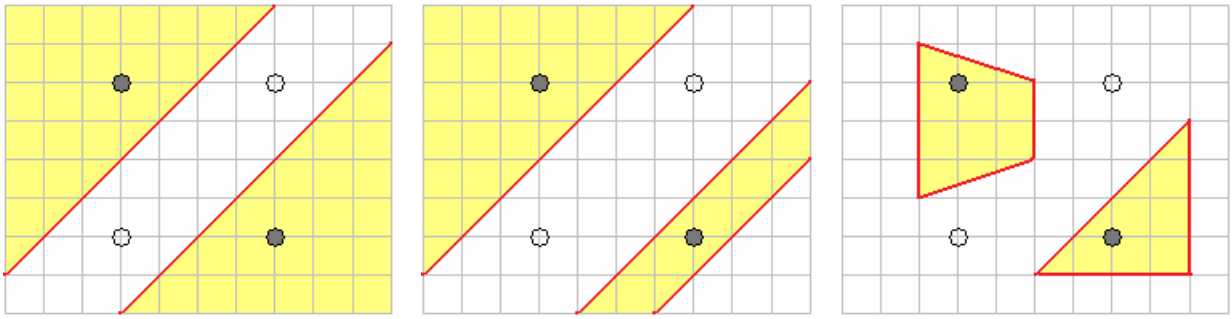


Рис. 6.3.4. Значения функции, реализуемой нейросетью другой архитектуры.

Замечание. Студенты часто сталкиваются с такой проблемой: реализовали алгоритм обратного распространения, а «он не работает»... функция, которая реализуется нейронной сетью, ничего не имеет похожего с тем, что должно по идее получаться. В большинстве случаев (если нет ошибок в коде) **надо просто увеличить число итераций**! Сходимость алгоритма не такая уж и быстрая. Нейросеть со структурой, изображённой на рис. 6.1.3, начинает «нормально работать» только после 2000 итераций! **ДЗ** *Попробуйте объяснить это явление. Реализуйте эвристику дельта-бар-дельта.*

§6.4. Переобучение нейронной сети

Мы нашли несколько решений задачи XOR с помощью нейронных сетей разной структуры. Соответствующие разделяющие поверхности показаны на рис. 6.4.1 а-б. Нетрудно построить двухслойную нейросеть, состоящую из персептронов (7 в первом слое, 1 во втором), которая реализует разделяющую поверхность, представленную на рис. 6.4.1 в.



а **б** **в**

Рис. 6.4.1. Различные решения задачи XOR с помощью нейронных сетей.

Из этих иллюстраций видно, в чём заключается переобучение для нейронных сетей: происходит формальная настройка на обучающую выборку, но при этом не наблюдается обобщающей способности. В худшем случае разделяющая поверхность классов может просто «окружить объекты одного из классов», см. рис. 6.4.2.

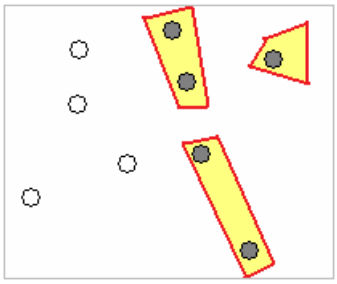


Рис. 6.4.2. Эффект переобучения.

Для борьбы с переобучением иногда применяют **методы упрощения сети:**

1) **упрощение структуры сети** (строят большую сеть, а затем постепенно удаляют отдельные связи или нейроны, стараясь сохранить хорошую функциональность на контроле).

2) **наращивание сети** (строят небольшую сеть, а затем, если её качество неудовлетворительно, постепенно добавляют в неё новые нейроны).

Для второй стратегии можно предложить следующий способ настройки сети, состоящей из персептронов (напомним, что алгоритм обратного распространения годится только для дифференцируемых функций активаций):

1) обучить персептрон. Если его качество приемлемое, то сеть построена, иначе

2) обучить второй персептрон, обучить персептрон, который берёт входы от двух уже построенных (и тождественного признака). Если качество полученной двухслойной сети приемлемое, то сеть построена. В противном случае возможно несколько стратегий:

3а) Попробовать снова обучить персептроны в первом слое, а затем выходной персептрон.

3б) Увеличить число персептронов в первом слое.

3c) Если увеличение числа персептронов в первом слое не даёт эффекта, то можно увеличить число слоёв.

ДЗ *Как обучать персептроны первого слоя, чтобы они имели разные весовые коэффициенты* (*не реализовывали одну и ту же функцию*)? *Например, если все их обучать по всей контрольной выборке НСКО-алгоритмом, то ответ будет один и тот же.*

§6.5. Эксперименты с нейронными сетями

Нейросети **очень универсальный** инструмент, который подходит практически для всех типов задач машинного обучения, **но не для всех задач.** Чтобы лучше научиться с ними работать, рекомендуется решить описанные ниже типы задач. Проводя эксперименты, пробуйте:

1. Менять число нейронов в слое.
2. Менять число слоёв.
3. Менять функции активации.
4. Менять архитектуру сети.

(пункты 1-2 обязательны для выполнения). Выработайте рекомендации пользователю по всем этим метапараметрам: архитектуре, функциям активации, числу слоёв, числу нейронов в слоях.

Распознавание графических образов

Обучающая выборка является набором правильно классифицированных бинарных матриц. Каждая матрица интерпретируется как бинарное изображение. На рис. 6.5.1 показана матрица и соответствующее ей изображение.

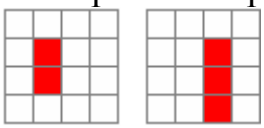
```
1000
1101
0100
0101
```

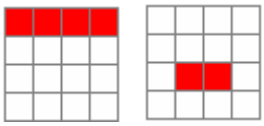
Рис. 6.5.1. Бинарная матрица и соответствующее ей изображение.

Не составит труда придумать задачи классификации на два непересекающихся класса. Ниже приведены некоторые примеры. Отметим, что они все разные по сложности (для решения с помощью фиксированной нейросети). Некоторые задачи могут быть трудными для нейросетей, см. [Минский, Пейперт].

Задачи:

Вертикальные корабли / горизонтальные корабли





Вертикальные и горизонтальные полосы / диагональные полосы

Точка / не точка

Буква «А» / буква «Г»

Множество точек чётной мощности / нечётной

Связные / несвязные фигуры (здесь 4-связность)

В простейшем варианте матрица кодируются бинарным (здесь 16-мерным вектором), на рис. 6.5.2 показан персептрон, который берёт входы с такой матрицы (нейрон «не понимает», что на вход поступает именно матрица).

ДЗ *Подумайте, как «сохранить структуру матрицы» при предъявлении её нейросети.*

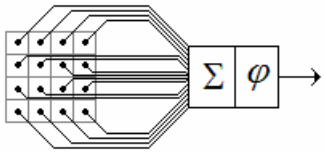


Рис. 6.5.2. Персептрон, обучаемый по матрице.

ДЗ *Придумайте свои типы задач. Проведите с ними эксперименты. Попробуйте упорядочить задачи по сложности.*

ДЗ *Попробуйте решать две задачи разными нейросетями и одной. Например, можно построить нейросеть с двумя нейронами в выходном слое и обучить так, чтобы первый активировался, когда предъявляется вертикальный кораблик* (*а не горизонтальный*), *а второй – когда предъявляется кораблик с чётным числом палуб* (*а не с нечётным*).

«Достаточно нетривиальные» реальные изображения классифицировать таким «примитивным» способом не удастся. При работе с реальными изображениями на вход сети даются не значения о цвете отдельных пикселей, а значения специальных функций от изображений (они зависят от решаемой

задачи). **ДЗ** *После приобретения достаточного опыта на модельных данных попробуйте порешать задачи с реальными бинарными изображениями. Лучше использовать изображения букв и цифр. Попробуйте сравнить разные модели алгоритмов.*

Прогнозирование (равномерного) временного ряда

Обычно идут по ряду $(x_1, x_2, \ldots)$ с некоторым окном ширины L, сеть настраивают на окне, т.е. значения $(x_i, x_{i+1}, \ldots, x_{i+L-1})$ поступают на вход, а на выходе желаемый отклик – $(x_{i+L}, x_{i+L+1}, \ldots, x_{i+L-R-1})$, где R – **ширина** или **горизонт прогноза**. На практике для прогноза отклика чаще строят R разных нейросетей (каждую тренируют по отдельности).

Решать задачи прогнозирования можно с реальными и модельными данными. В качестве модельных данных обычно используют функции, которые «очень похожи» на настоящие временные ряды: в них есть тренд (полином небольшой степени), сезонность (периодическая функция) и шум (случайная добавка), например функцию вида

$$f(t) = c_1 t^2 + c_2 t + c_3 + \sin(c_4 t) + \cos(c_5 t) + c_6 \xi_t,$$

где ξ_t имеет нормальное распределение со средним 0 и дисперсией 1/3.

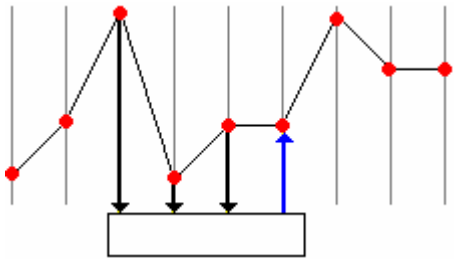


Рис. 6.5.3. Схема прогнозирования одного значения по окну ширины 3.

Сжатие входных данных

Когда исходные данные имеют признаковое описание в пространстве большой размерности, часто «приходится» генерировать новое признаковое пространство, которое имеет меньшую размерность, но «помнит» особенности исходного. Примером является применение метода главных компонент. Также для этого применяют **нейросеть «с узким горлом»**. В такой многослойной нейросети (см. рис. 6.5.4) один из средних слоёв имеет небольшое число нейронов (такой слой называется **«узким горлом»**), а число выходных нейронов совпадает с числом признаков (входов, отличных от константного). При настройке сети добиваются такой функциональности, чтобы она получала на выходе признаковое описание объекта, поданного на выход. Если сеть функционирует так с небольшой ошибкой, то значений нейронов «узкого горла» достаточно для восстановления исходной признаковой информации.

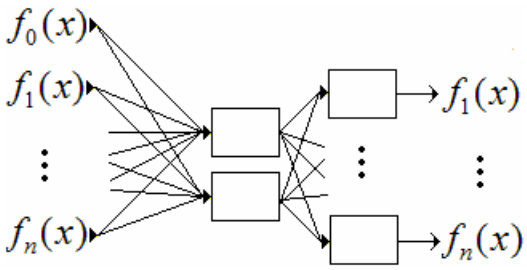


Рис. 6.5.4. Нейросеть «с узким горлом».

Таким образом, число нейронов в узком горле соответствует размерности нового признакового пространства. Если в узком горле использовать два или три нейрона, то их значения можно использовать для визуализации данных.

ДЗ *Постройте нейросеть для визуализации данных в пространстве малой размерности. Придумайте модельную задачу для исследования и способ генерации обучающей выборки. Например, выборка, состоящая из объектов вида*

$$((a+b)^2,(a-2b)^2,3ab,2a^2+b,a^2-b),$$

«естественно» переводится в пространство вида $\{(a,b)\}$ (*т.е., по крайней мере, допускает вложение в 2-мерное пространство*).

Глава 7. МЕТОДЫ РЕГРЕССИИ

§7.1. Непараметрическая регрессия

Одним из самых простых и стандартных приёмов в непараметрической регрессии при оценке значения $y(x)$ является «усреднение» известных значений $y(x^1),\ldots,y(x^m)$ с учётом «похожести» x на объекты $x^1,\ldots,x^m$:

$$A(x)=\frac{\sum_{t=1}^{m} w_t(x)y(x^t)}{\sum_{t=1}^{m} w_t(x)}. \qquad (7.1.1)$$

Если в качестве **весовых функций** в этой формуле взять

$$w_t(x)=K\left(\frac{\rho(x,x^t)}{h}\right),$$

где $K:[0,+\infty)\to[0,+\infty)$ – невозрастающая функция, а ρ – метрика на множестве объектов, то получаем **формулу Надарая-Ватсона**. Часто функцию K выбирают так, что она обращается в ноль при больших значениях аргумента, поэтому реально суммирование в (7.1.1) происходит только по ближайшим объектам. Как всегда, сумму можно менять на другие способы усреднения (см. §3.3), в результате чего получаются более робастные (устойчивые к шумам) методы.

Пример. Главный недостаток, который «бросается в глаза» при анализе формулы (7.1.1) – «плохая настройка» на обучающую выборку. На рис. 7.1.1 показан график функции, восстановленный по точкам $(1,5)$, $(3,7)$, $(4,4)$ при $K(z)=\exp(-z^2)$. Как видим, график не проходит через точки, зато показывает, как функция «должна себя вести».

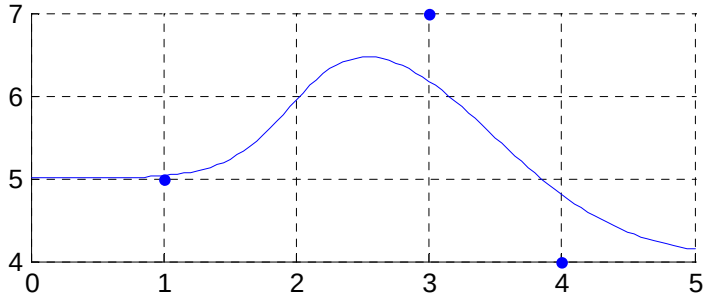


Рис. 7.1.1. Применение формулы Надарая-Ватсона

На рис. 7.1.2 показан результат выполнения следующего MatLab-кода.

Регрессия по формуле Надарая-Ватсона

```
K = @(x) exp(-x.^2); % функция K
x = 0:0.05:5; % отрезок
f = sin(x); % истинные значения функции
X = x(1:10:end); % обучающая выборка - объекты
Y = sin(X); % - их метки
```

```
t = repmat(x',1,length(X)) - repmat(X,length(x),1);
t = arrayfun(K, t);
sumt = sum(t, 2);
t = sum(t.*repmat(Y,length(x),1), 2)./sumt; % значения ф-лы Н-В
clf; hold on; grid on; % Графика
plot(x, f, 'b'); % как должно быть
scatter(X, Y, 20, 'filled'); % обучение
plot(x, t, 'k'); % что получилось
```

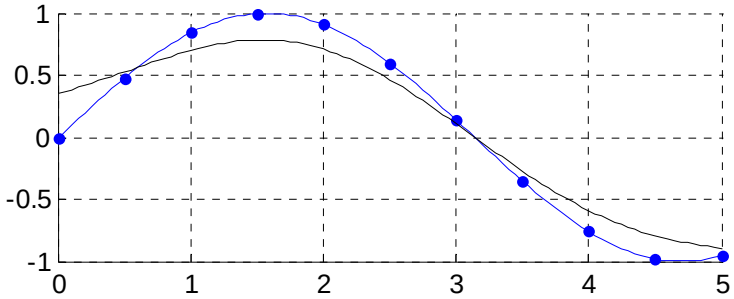


Рис. 7.1.2. Восстановление функции `y = sin(x)` по формуле Надарая-Ватсона.

ДЗ *Усовершенствуйте метод так, чтобы восстановленный график проходил по точкам обучающей выборки. Проведите эксперименты при различных функциях* K *и значении параметра* h.

ДЗ *Предложите методы борьбы с выбросами* (*например, чтобы при восстановлении функции* `y = sin(x)` *ответ не менялся при добавлении в обучение нескольких выбросов*).

ДЗ *Напишите процедуру, которая автоматически устраняет выбросы, выбирает функцию* K, *значения параметра* h, *способ усреднения. Заметим, что выбор параметра* h *может зависеть от* x (*см.* [Воронцов]).

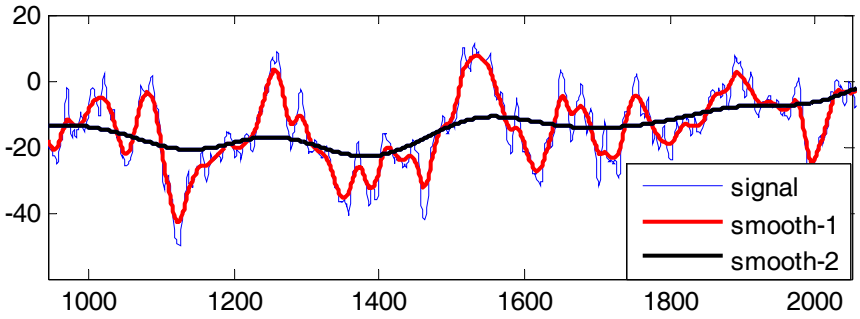


Рис. 7.1.3. Пример сглаженной электрокортикограммы (при двух разных значениях h**).**

Замечание. Приведённый метод чаще правильнее называют **сглаживанием**. Его можно применять, когда следует заменить некоторую хаотично меняющуюся функцию на достаточно гладкую, удобную для анализа (см. рис. 7.1.3).

§7.2. Линейная регрессия

Предположим, что мы ищем представление функции, заданной прецедентно в виде

$$\sum_{j=1}^{r} w_j g_j(x). \qquad (7.2.1)$$

Тогда, чтобы она принимала нужные значения на обучающей выборке, необходима совместность системы

$$\begin{cases} \sum_{j=1}^{r} w_j g_j(x^1) = y(x^1), \\ \vdots \\ \sum_{j=1}^{r} w_j g_j(x^m) = y(x^m), \end{cases}$$

или

$$\begin{bmatrix} g_1(x^1) & \dots & g_r(x^1) \\ \vdots & \ddots & \vdots \\ g_1(x^m) & \dots & g_r(x^m) \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ \vdots \\ w_r \end{bmatrix} = \begin{bmatrix} y(x^1) \\ \vdots \\ y(x^m) \end{bmatrix}.$$

Подобные матричные уравнения вида $X \cdot \tilde{w} = \tilde{b}$ мы научились решать в §5.4[1] (минимизируя функцию $J = \| X \cdot \tilde{w} - \tilde{b} \|^2$). Нахождение решения в виде (7.2.1) называется **обобщённой линейной регрессией**. Поиск решения в виде линейной комбинации значений признаков

$$\sum_{j=1}^{n} w_j f_j(x)$$

называется **линейной регрессией**.

Пример. Пример, рассмотренный в §7.1, легко переделать в пример восстановления **полиномиальной регрессии** (обобщённой линейной регрессии при $g_j(z) = z^{j-1}$), заменив центральный блок кода строками

```
mX = repmat(X', 1, 4).^repmat(0:3, length(X), 1);
w = (mX'*mX)\(mX'*Y');
t = (repmat(x', 1, 4).^repmat(0:3, length(x), 1))*w;
```

или строками

```
p = polyfit(X, Y, 3);
t = polyval(p, x);
```

Первые легко обобщить на произвольный базис функций, а вторые показывают, как полиномиальная регрессия реализована в MatLabe. Обязательно изучите код этих функций (наберите в системе MatLab «**`edit polyfit`**»). На рис. 7.2.1 показано оптимальное приближение наших данных с помощью функции вида

$$w_0 + w_1 x^1 + w_2 x^2 + w_3 x^3.$$

---

[1] Напомним, что здесь обозначение матрицы X совпадает с обозначением пространства допустимых объектов, но это не должно привести к путанице.

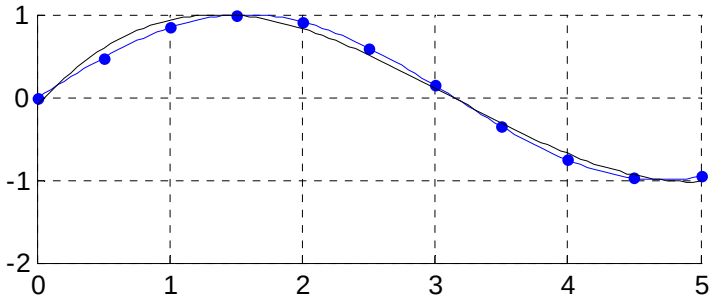


Рис. 7.2.1. Полиномиальная регрессия.

Основной проблемой при решении уравнения $X \cdot \tilde{w} = \tilde{b}$ может быть вырожденность (или близость определителя к нулю) матрицы $X^{\mathrm{T}} \cdot X$. Чаще всего эту матрицу «изменяют небольшой добавкой», заменяя решение

$$\tilde{w} = (X^{\mathrm{T}} \cdot X)^{-1} \cdot X^{\mathrm{T}} \cdot \tilde{b}$$

решением

$$\tilde{w} = (X^{\mathrm{T}} \cdot X + c \cdot I) \cdot X^{\mathrm{T}} \cdot \tilde{b},$$

где I – единичная матрица (бинарная матрица с единицами только на главной диагонали), а c – константа. Такая регрессия называется **гребневой регрессией**, а указанная подмена решений соответствует минимизации нового функционала $J = \| X \cdot \tilde{w} - \tilde{b} \|^2 + c \cdot \| \tilde{w} \|^2$ (т.е. это регуляризация).

Альтернативный способ борьбы с вырожденностью – изменение базиса для представления функции. Например, в случае линейной регрессии сокращают признаковое пространство, удаляя признаки, которые коррелируют с остальными, являются неинформативными и т.д. Один из способов такого удаления рассмотрен в следующем параграфе. Часто перебирают базисы в надежде найти «самый лучший» для данной задачи. При этом применяют методы глобальной оптимизации (см. главу 8).

§7.3. Метод главных компонент

Для визуализации данных и генерации признаков часто применяют **метод главных компонент**. Его можно также использовать в линейной регрессии для «удаления лишних признаков». Геометрический смысл метода следующий. Если исходная выборка представляет собой «облако точек» в многомерном пространстве $\mathbf{R}^n$, то существует прямая, проекции точек на которую обладают наибольшим дисперсионным разбросом. Из всех ортогональных ей прямых существует прямая, проекции точек на которую обладают наибольшим дисперсионным разбросом и т.д. Имея две первые прямые, можно изобразить наше облако точек на плоскости (которая проходит через эти прямые), в проекции на которую оно «максимально размыто».

Примером применения метода главных компонент является результат выполнения описанного ниже MatLab-кода (см. рис. 7.3.1).

```
X = rand([1000 2]);
X(X(:,2) > X(:,1).^2+0.5, :) = [];
X(X(:,2) < X(:,1)-0.5, :) = []; % создали выборку
subplot(1,2,1); % визуализация исходных данных
scatter(X(:,1), X(:,2), 20, 'filled'); grid on;
Xmean = mean(X);
X = bsxfun(@minus, X, Xmean); % центрирование
[U,S,V] = svd(X, 0); % SVD-преобразование X = U*S*V';
subplot(1, 2, 2); % визуализация того, что получилось
scatter(U(:,1) ,U(:,2), 20, 'filled'); grid on;
```

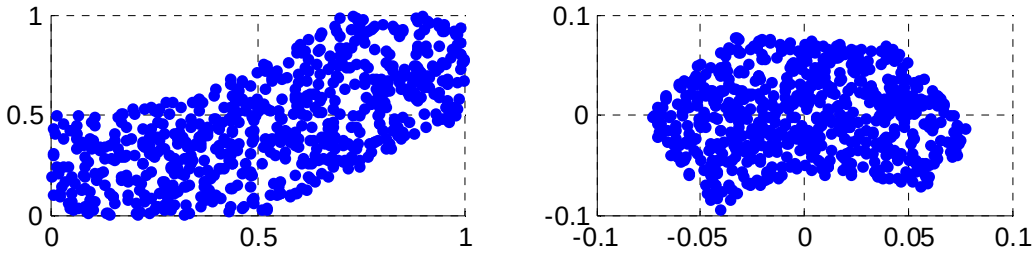


Рис. 7.3.1. Выборка до (слева) и после (справа) преобразования методом главных компонент.

Метод главных компонент основан на **сингулярном разложении матриц** (функция **svd** в MatLabe). Опишем кратко необходимые сведения. Матрица вида $H^{\mathrm{T}}H$ называется в линейной алгебре **матрицей Грама** (матрицей попарных скалярных произведений), является симметричной и неотрицательно определённой, имеет n линейно независимых собственных векторов $\tilde{v}^1,\ldots,\tilde{v}^n$, которым соответствуют собственные значения $\lambda_1,\ldots,\lambda_n$, $\lambda_1 \geq \ldots \geq \lambda_n \geq 0$:

$$H^{\mathrm{T}}H\tilde{v}^j = \lambda_j \tilde{v}^j, \quad j \in \{1,2,\ldots,n\}.$$

Пусть r – число ненулевых собственных значений, тогда

$$\tilde{u}^j = \frac{1}{\sqrt{\lambda_j}} H\tilde{v}^j -$$

– собственный вектор матрицы HH^{T}, соответствующий собственному значению λ_j, и

$$H = \sum_{j=1}^{r} \sqrt{\lambda_j}\,\tilde{u}^j (\tilde{v}^j)^{\mathrm{T}}.$$

Последнее равенство называется сингулярным разложением матрицы H (в MatLab-коде, который приведён выше, это соответствует равенству **X = U*S*V'**). Отметим, что матрицы, составленные из векторов $\tilde{v}^j$ ($[\tilde{v}^1,\ldots,\tilde{v}^n]$) и $\tilde{u}^j$ являются унитарными (т.е. из **X = U*S*V'** получаем, что **X*V = U*S**). Сумма первых k членов сингулярного разложения даёт наилучшую аппроксимацию исходной матрицы H матрицей ранга, не превосходящего k. Причём ошибка аппроксимации считается по формуле

$$\sum_{j=k+1}^{n} \lambda_j . \qquad (7.3.1)$$

По этой же формуле вычисляется погрешность представления объектов (строк матрицы H) в новом k-мерном пространстве. В этом пространстве объекты являются строками матрицы

$$H \cdot [\tilde{v}^1,\ldots,\tilde{v}^k].$$

Строки именно такой двухстолбцовой матрицы и выводит представленный выше код. Отметим, что в нём не происходит понижения размерности пространства (т.е. $n = k$). **ДЗ** *Реализуйте понижение размерности.*

Прямой с наибольшим дисперсионным разбросом оказывается прямая, соответствующая вектору $\tilde{v}^1$. Ортогональная ей прямая с наибольшим дисперсионным разбросом соответствует вектору $\tilde{v}^2$ и т.д. При сокращении размерности можно выбрать столько собственных векторов, в пространстве какой размерности мы хотим работать (можно при этом руководствоваться ошибкой (7.3.1)). Для визуализации достаточно перейти в 2х или 3х-мерное пространство. Отметим, что в новом признаковом пространстве признаки не коррелируют. Нулевые собственные значения соответствуют найденным линейным зависимостям (соответствующие признаки можно не включать в новое признаковое пространство). Это иллюстрирует результат работы следующей последовательности MatLab-команд, которая моделирует случай, когда четвертый признак линейно выражается через первые два, а третий «почти выражается» (к линейной комбинации добавляется шум).

```
>> X = rand([1000 2]);
>> X(:,3) = X(:,1)+X(:,2)+rand([1000 1])/10;
>> X(:,4) = X(:,1)-X(:,2);
>> Xmean = mean(X);
>> X = bsxfun(@minus,X,Xmean);
>> [U,S,V] = svd(X,0);
>> diag(S)

ans =
   16.3582
   15.6783
    0.5138
    0.0000
```

ДЗ *Напишите программу для визуализации многомерных данных методом главных компонент.*

Отметим, что в приведённых кодах мы центрировали наши объекты (строки матриц), поскольку хотели оценивать дисперсионный разброс относительно «центра облака точек» (т.е. математического ожидания, если трактовать строки, как реализации многомерной случайной величины).

Полезную информацию по методу главных компонент и SVD-разложению можно найти в [Воронцов], [Стрижов, Пташко, 2007], [Голяндина, 2004].

Опишем кратко идею метода «Гусеница» анализа временных рядов (см. [Голяндина, 2004]). Пусть дан (равномерный и одномерный) ряд $\tilde{f} = (f_1,\ldots, f_N)$. Зададимся шириной окна L и представим матрицу

$$X = \begin{bmatrix} f_1 & f_2 & \cdots & f_K \\ f_2 & f_3 & \cdots & f_{K+1} \\ \cdots & \cdots & \cdots & \cdots \\ f_L & f_{L+1} & \cdots & f_N \end{bmatrix},$$

где $K = N - L + 1$, в виде сингулярного разложения:

$$X_1 + X_2 + \ldots + X_d, \qquad (7.3.2)$$

где $X_j = \sqrt{\lambda_j}\tilde{u}^j(\tilde{v}^j)^{\mathrm{T}}$. Теперь сложим в выражении (7.3.2) некоторые «похожие» матрицы. Этот «неформальный» этап называется группировкой (т.е. группировкой слагаемых). Переходим к выражению

$$X'_1 + X'_2 + \ldots + X'_r.$$

Превратим теперь каждую матрицу X'_j во временной ряд $\tilde{f}^j$. Для этого обратим внимание, что каждое значение ряда $\tilde{f}$ заполняет соответствующую антидиагональ матрицы X (множество элементов матрицы с постоянной суммой индексов). На любой антидиагонали матрицы X'_j могут стоять попарно различные значения, но их можно усреднить и получить ряд

$$\tilde{f}^j = (f_1^j,\ldots, f_N^j),$$

где f_i^j – среднее арифметическое чисел $x^j_{a,i+1-a}$ матрицы $X'_j = \| x^j_{ab} \|$. Ясно, что тогда

$$\tilde{f} = \tilde{f}^1 + \ldots + \tilde{f}^r,$$

т.е. мы получили разложения нашего ряда в сумму рядов. При удачном выборе ширины окна и процедуры группировки часто удаётся получить таким образом «естественное» разложения ряда на тренд, периодические составляющие и шум.

ДЗ *Реализуйте метод «Гусеница». Для группировки попробуйте применить модификации некоторых методов кластеризации.*

Глава 8. МЕТОДЫ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ

§8.1. Задача глобальной оптимизации

Для описания основных методов глобальной оптимизации рассмотрим упрощённую задачу: будем максимизировать функцию $f(\tilde{z})$, заданную в вершинах многомерного булева куба, $\tilde{z} = (z_1,\ldots,z_n) \in Z = \{0,1\}^n$, $f: Z \to \mathbf{R}$. Отметим, что задача максимизации такой функции встречается, например, при **отборе (селекции) признаков**: булевский вектор $\tilde{z}$ описывает, какие признаки выбираются (они помечены единичными координатами), а f – качество классификации (или регрессии) в таком признаковом подпространстве.

Замечание. Когда признаков очень много, есть шумовые признаки, есть признаки, которые коррелируют друг с другом, отбор (селекция) признаков просто необходим, поскольку позволяет избежать переобучения и провести более тщательный анализ признакового пространства (чем меньше признаков – тем быстрее работают многие алгоритмы). **ДЗ** *Попробуйте доказать, что если объекты двух классов не разделялись гиперплоскостью, то при добавлении шумовых признаков* (*пусть значения таких признаков будут равномерно распределены на отрезке* [0,1]) *наступит линейная разделимость. Оцените, сколько шумовых признаков достаточно добавить для линейной разделимости* (*см.* [Дьяконов, 2006]).

Считаем, что наша функция f реализована в виде **«чёрного ящика»**: мы обращаемся к этому «чёрному ящику» с аргументом $\tilde{z}$ и получаем значение $f(\tilde{z})$, причём вычисление функции может занять достаточно долгое время (как в задаче селекции признаков). Если бы функция f была дифференцируемой функцией непрерывного аргумента, то можно было бы воспользоваться итерационным градиентным методом с шагом

$$\tilde{z} = \tilde{z} + c\frac{\partial f(\tilde{z})}{\partial \tilde{z}}$$

или итерационным методом Ньютона (если можем вычислить Гессиан

$$H_f(z_1,\ldots,z_n) = \left\| \frac{\partial}{\partial z_i} \frac{\partial f(z_1,\ldots,z_n)}{\partial z_j} \right\|_{ij n\times n})$$

с шагом

$$\tilde{z} = \tilde{z} + c[H_f(\tilde{z})]^{-1}\frac{\partial f(\tilde{z})}{\partial \tilde{z}}.$$

В нашем случае нет даже «красивой формулы», которая определяет функцию. Задачи оптимизации «чёрного ящика» возникают, когда значение функции получается в результате работы какого-то алгоритма (т.е. когда функция f задана не аналитически, а алгоритмически).

Рис. 8.1.1. Схема работы «чёрного ящика».

Для задачи глобальной оптимизации не существует наилучшего алгоритма. Можно придумать функцию f, которая везде равна нулю, кроме одной какой-нибудь точки. Ясно, что любой метод оптимизации (который на каждой итерации запрашивает значение функции f в какой-то точке пространства Z) при достаточно большом n найдёт решение за приемлемое время с «очень маленькой» вероятностью. Но метод должен справляться с «совсем простыми» функциями типа

$$f(\tilde{z}) = w_1 z_1 + \ldots + w_n z_n,$$

а лучше – с «более экзотическими» функциями, например

$$f(\tilde{z}) = (w_1 z_1 + \ldots + w_k z_k) \cdot \sin(w_{k+1} z_{k+1} + \ldots + w_n z_n).$$

ДЗ *Придумайте функции на вершинах булева куба для тестирования методов глобальной оптимизации.* Один из способов «придумывания» таких функций следующий. Придумать «нетривиальную» функцию нескольких непрерывных аргументов на $[0,1]^s$ (с большим числом локальных максимумов, но достаточно гладкую, см. рис. 8.1.2), а затем вместо каждого аргумента подставить линейную комбинацию вида $w_1 z_1 + \ldots + w_n z_n$, $w_1 + \ldots + w_n = 1$.

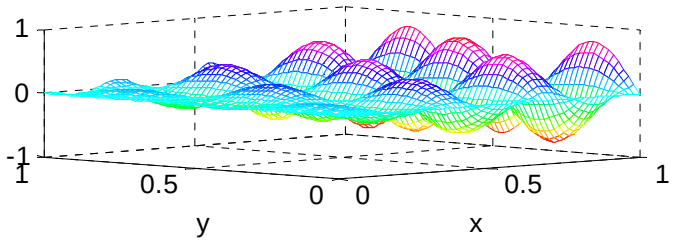


Рис 8.1.2. Функция $\sin(x) \cdot \sin(20x) \cdot \sin(10y)$.

§8.2. Полный перебор

Полный перебор на практике (в задачах большой размерности) никогда не завершится, но можно

1. Грамотно его организовать (сначала перебирать потенциально лучшие точки).
2. Организовать «квазиполный перебор» (придумать эвристики, которые прекращают рассмотрение каких-то потенциально неперспективных вариантов).

Что такое правильная организация «квазиполного» перебора понятно после хорошей практики в программировании логических игр (см. главу «Программирование логических игр»). Если функция f «достаточно однородна», то при переборе можно «копить информацию» о ней. Например, если изменение какого-то аргумента в разных точках пространства не

оказывает влияния на значение функции, то она может зависеть от него фиктивно, поэтому больше его не меняем (перебор сокращается). Можно также отлавливать монотонную зависимость от каких-то переменных и т.д. Помните: **наша функция не случайная**, а отражает работу какого-то алгоритма (т.е. логика в её поведении есть, осталось эту логику побыстрее выяснить). **ДЗ** *Подумайте, какие свойства функции можно отлавливать в процессе перебора. Обратите внимание, что функция может обладать некоторыми свойствами лишь в какой-то области пространства.*

Замечание. Часто при отборе признаков применяют такую стратегию: перебирают всевозможные тройки признаков, оценивают качество решения в таких маломерных пространствах, удаляют только те признаки, которые в совокупности с другими парами не составили хороших признаковых пространств.

§8.3. Направленный поиск

Общая схема типичного **градиентного (жадного) алгоритма** такая

Градиентный алгоритм

1. Пусть $\tilde{z}$ случайная точка нашего пространства Z (выбор начальной точки).
2. Если $f(\tilde{z}) \le f(\tilde{z}^t) = \max\limits_{\tilde{z}^t \in U(\tilde{z})} [f(\tilde{z}^t)]$, тогда $\tilde{z} = \tilde{z}^t$ и переходим к п.2 (переход в точку, которая не уменьшает значение функции).
3. Иначе (пришли в локальный максимум) выдаём ответ.

Здесь $U(\tilde{z})$ – окрестность точки $\tilde{z}$. В простейшем случае – окрестность первого порядка: $U(\tilde{z}) = \{\tilde{z}^1, \ldots, \tilde{z}^n\}$, где $\tilde{z}^t = (z_1, \ldots, z_{t-1}, 1 - z_t, z_{t+1}, \ldots, z_n)$ – t-й сосед точки $\tilde{z} = (z_1, \ldots, z_n)$.

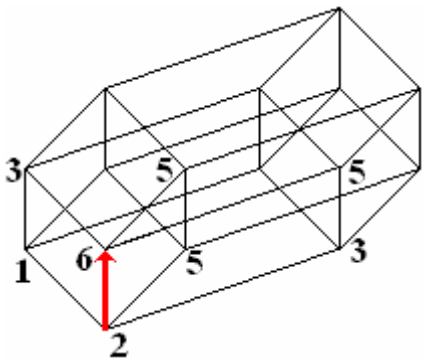


Рис. 8.3.1. Шаг градиентного алгоритма.

Пример. Пусть начальная точка **(0000)**, **f(0000) = 2**, в её окрестности значения:

f(1000) = 1,
f(0100) = 6,
f(0010) = 5,
f(0001) = 3,

переходим в точку **(0100)**, в её окрестности значения
f(1100) = 3,
f(0110) = 5,
f(0101) = 5,
f(0000) = 2 (отсюда пришли, поэтому на это значение можно не смотреть) меньше, поэтому найденный максимум **f(0100) = 6**. См. рис. 8.3.1.

Усовершенствования градиентного алгоритма связаны с выбором начальной точки и изменением поведения в точке локального максимума:

1. Разумно несколько раз запускать алгоритм с разными начальными точками $\tilde{z}$ (из разных областей пространства Z).

2. Можно перед запуском алгоритма генерировать несколько точек и выбирать в качестве начальной ту, в которой значение функции наибольшее.

3. В локальном максимуме разумно продолжать движение в сторону больших значений. Эта идея используется в методе **симуляции отжига**, где из локального максимума переходят в точку $\tilde{z}^t$ с вероятностью

$$\exp([f(\tilde{z}^t) - f(\tilde{z})]/T)$$

(перебирают соседние точки и осуществляют переход с такой вероятностью), где T – параметр, называемый **температурой**, который достаточно большой в начале работы алгоритма, а затем уменьшается. **ДЗ** *Зачем вводить этот параметр?*

4. Можно организовать «квазиполный перебор»: из каждой точки идти не только в сторону «лучшего соседа», а сразу в двух направлениях: к лучшему соседу и ещё к кому-нибудь (выбирать вторую точку окрестности по значению функции или случайно выбирать точку из окрестности). В подобном переборе есть вероятность, что мы в рекурсии несколько раз попадём в одни и те же точки, поэтому хранят список рассмотренных точек (или рассмотренных за несколько последних итераций).

5. Можно также «собирать информацию о функции» и использовать её в дальнейшем (см. §8.2).

ДЗ *Придумайте, как «равномерно замостить» вершины булева куба начальными точками.*

ДЗ *Придумайте эвристику поведения в точке локального максимума* (*обратите внимание, что алгоритм не доложен зацикливаться: на следующем шаге он не должен возвращаться назад*). *Вспомните «эффект горизонта» в логических играх и методы борьбы с ним.*

Часто начальной точкой выбирают $\tilde{z} = (0,\ldots,0)$ и «двигаются вверх по булеву кубу» (при отборе признаков это соответствует тому, что мы стартуем с пустого множества признаков и по одному наращиваем его, пока качество

классификации/регрессии улучшается). Применяют также другую стратегию: из точки (1,…,1) «идут вниз».

В **методе луча (beam search)** хранят k лучших неисследованных значений (окрестность которых не изучали). При выборе следующей точки выбирается «лучший кандидат» из хранимого списка. Поясним на нашем примере при $k = 3$.

Пример. Пусть начальная точка **(0000)**, **f(0000) = 2**, в её окрестности значения
f(1000) = 1,
f(0100) = 6,
f(0010) = 5,
f(0001) = 3,
лучшие неисследованные точки и значения функции в них (в порядке убывания значений): **[(0100), 6; (0010), 5; (0001), 3]**, берём первого кандидата из этого списка. В окрестности точки **(0100)** новые значения
f(1100) = 3,
f(0110) = 5,
f(0101) = 5,
т.е. список лучших неисследованных точек: **[(0010), 5; (0110), 5; (0101), 5]**. Для точки **(0010)** новые значения исследуемой функции
f(1010) = 4,
f(0011) = 3,
а список лучших неисследованных точек: **[(0110), 5; (0101), 5; (1010), 4]**. и т.д. (найденное пока наибольшее значение **6** и соответствующий аргумент **(0100)**, естественно, хранят). Завершают работу алгоритма по истечении времени, выделенного на поиск экстремума, или если значения функций в нашем списке несколько итераций подряд уменьшаются.

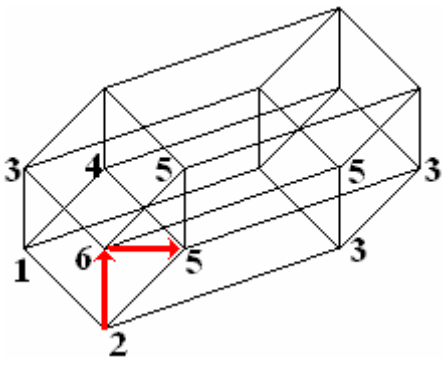


Рис 8.3.2. Два шага методом луча.

В следующем алгоритме мы идём последовательно по слоям булева куба. Чтобы не смотреть все точки слоя (тогда алгоритм будет иметь экспоненциальную сложность), мы смотрим фиксированное число соседей предыдущего слоя с наибольшими значениями исследуемой функции.

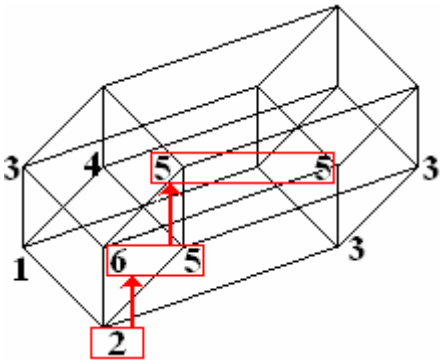


Рис. 8.3.3. Два шага локальным поиском для $k = 2$.

Локальный поиск

1. Пусть $W = \{(0,\ldots,0)\}$, $\tilde{z}^* = (0,\ldots,0)$.
2. Выбрать из множества $\{\tilde{z}^t \mid \tilde{z} = (z_1,\ldots,z_n) \in W, z_t = 0\}$ k точек с наибольшем значением функции f (или все точки, если мощность этого множества меньше k). Пусть теперь множество W состоит из них.
3. Если среди точек множества W есть точка со значением большим, чем $f(\tilde{z}^*)$, то запомнить её в переменной $\tilde{z}^*$.
4. Если $W \neq \{(1,\ldots,1)\}$, то переходим к п. 2.
5. Выдать ответ: $\tilde{z}^*$, $f(\tilde{z}^*)$.

Отметим, что аналогичная идея используется в многорядном итерационном алгоритме МГУА (см. [Воронцов]).

§8.4. Генетические алгоритмы[1]

Выберем случайно k точек в пространстве Z: $\{\tilde{z}^1,\ldots,\tilde{z}^k\}$. Представим, что точки – существа, живущие в некотором мире (в терминах теории генетических алгоритмов их принято называть **индивидами**). Будем называть текущее множество рассматриваемых точек **популяцией**. Вектор $\tilde{z} = (z_1,\ldots,z_n)$ описывает **генотип** конкретного существа (часто такой вектор называют **хромосомой**), а значение $f(\tilde{z})$ – его **приспособленность**. Как известно, самые неприспособленные умирают… поэтому удалим r точек, $r < k-1$, с наименьшим значением функции f в них (это называется процедурой **селекции**). Выжившие дают потомство… поэтому разобьем их на пары[2] (пока не важно как). Существа $\tilde{z} = (z_1,\ldots,z_n)$ и $\tilde{u} = (u_1,\ldots,u_n)$ порождают потомка с генотипом $\tilde{v} = (v_1,\ldots,v_n)$, где $v_t = 1$ при $z_t = u_t = 1$, $v_t = 0$ при $z_t = u_t = 0$, а в противном случае полагаем $v_t = 1$ с вероятностью 0.5 и, соответственно, $v_t = 0$ с

[1] Изобретены Джонам Холландом (John Holland). См. [*John Holland* Adaptation in Natural and Artificial Systems // University of Michigan Press, 1975].
[2] Заметим, что отсутствует понятие пола. Потомство может дать любая пара существ.

вероятностью 0.5 (т.е. потомок унаследовал гены, которые есть у обоих родителей). Этот этап называется **размножением**, а приведённая реализация – **равномерным кроссовером**. Теперь всё можно повторить по циклу: опять производим селекцию, потом размножение и т.д. Это общая идея генетических алгоритмов. Различные модификации связаны с устранением недостатков исходной схемы. Например, неплохо бы обеспечить «разнообразие в популяции». Если самые приспособленные начнут активно размножаться, то их гены будут передаваться потомкам и координаты точек быстро стабилизируются. Поэтому, как и в природе, каждый потомок может мутировать, т.е. с малой вероятностью меняется значение каждого его гена (процедура **мутации**).

Схема генетического алгоритма.

1. Инициализация.
2. Селекция.
3. Скрещивание (размножение).
4. Мутации.
5. Переход к п. 2.

Опишем некоторые изменения и дополнения, которые можно использовать при реализации алгоритмов на практике (не претендуя на полноту обзора, а только для «возбуждения» воображения).

Возможные изменения в селекции

1. Чтобы уже найденные хорошие решения «не портили» процедуру поиска, как и в природе, существо может умереть не только от неприспособленности, но и по причине старения (после фиксированного числа итераций убираем эту точку-существо из популяции).

2. «Плохо приспособленное» существо может породить хорошо приспособленное, поэтому можно отбирать существ по вероятностному критерию: чем ниже приспособленность, тем выше вероятность удаления существа из популяции, чтобы у слабых был шанс выжить.

3. Можно разбить существа на пары, но не с целью размножения, а с целью конкуренции… смоделировать бой между парой существ, в которой наиболее приспособленный побеждает с большей вероятностью и «убивает» слабейшего (соответствующая точка удаляется).

4. Иногда в популяцию добавляют индивидов, которые сильно отличаются от её представителей («в стаю приходят чужаки»; можно инвертировать хромосомы представителей популяции), чтобы обеспечить поиск решения по всему пространству.

5. Есть варианты алгоритмов с несколькими популяциями, которые «живут независимо», но иногда могут смешиваться или конкурировать, уничтожать друг друга.

Замечание. Бой между двумя существами с приспособленностями p_1, p_2 можно смоделировать следующим образом: генерируем случайную величину $\xi \in [0,1]$, если она меньше порога

$$\frac{p_1}{p_1 + p_2},$$

то убиваем первое существо, а иначе – второе.

Возможные изменения в скрещивании

1. Часто при скрещивании применяют схему **одноточечного кроссовера**: если родители $\tilde{z} = (z_1,\ldots,z_n)$ и $\tilde{u} = (u_1,\ldots,u_n)$, то их потомок $(z_1,\ldots,z_r,u_{r+1},\ldots,u_n)$ для некоторого r: $1 \le r < n$. Вообще, классическая схема – порождение двух потомков: $(z_1,\ldots,z_r,u_{r+1},\ldots,u_n)$ и $(u_1,\ldots,u_r,z_{r+1},\ldots,z_n)$. Наверное, понятно, что такое **двухточечный кроссовер**…

2. Возможны совершенно разные виды скрещивания: просто разбить всех по парам[1] или только лучших существ, выбирать пары из популяции с возвратом (за одну итерацию одно существо может несколько раз быть родителем), причём выбирать существо с вероятностью, которая зависит от его приспособленности.

3. Один из вариантов генетического алгоритма – **алгоритм с постоянным числом индивидов**, в котором случайно выбирается пара родителей, которые порождают двух потомков, а потомки «вытесняют» двух существ из популяции (занимают их место). Естественно, можно вытеснять только при определённых условиях, например при более высокой приспособленности молодых особей.

4. В **конвейерной** версии алгоритма разыгрывается случайная величина на отрезке [0,1]. Если она меньше 0.1, то берётся случайный индивид, удаляется из популяции и помещается в новую популяцию, которая изначально пустая (он «перешёл в следующее поколение»). В противном случае (с вероятностью 90%), берётся случайная пара индивидов, удаляется из популяции[2], порождает двух потомков, которых добавляют в новую популяцию. Кстати, в этом варианте алгоритма нет мутации! **ДЗ** *Попробуйте её всё-таки сделать, посмотрите на результат.*

Возможные изменения в мутации

1. Мутировать могут все особи на каждой итерации алгоритма, а можно оставлять наиболее приспособленных без изменений (принцип **элитаризма**).

2. Увеличивать вероятность мутации, если перестала расти приспособленность популяции (средняя или приспособленность её лучшего представителя).

---

[1] Если существ нечётное число, то одно из существ не выбирается.

[2] Если там остался один, то действия повторяют предыдущий вариант.

3. Кстати, можно поискать градиентным алгоритмом в окрестности наиболее приспособленных особей ещё более приспособленных… Это смесь генетики и локального поиска.

Отдельная проблема при применении генетических алгоритмов – кодирование особей. Мы рассмотрели очень простой случай булева пространства. **ДЗ** *Как применить генетические алгоритмы для оптимизации функций вещественных переменных?* При использовании генетических алгоритмов для *k*-значной информации часто её переводят в бинарную с помощью кодов Грея. Стандартные коды близких по значению чисел могут сильно отличаться. В коде Грея (см. табл.) коды соседних чисел отличаются ровно на один бит. В MatLabe такую таблицу кодировок можно получить с помощью команд

```
X = dec2bin(0:7)=='1',
X(:,2:end) = bitxor(X(:,2:end),X(:,1:end-1)) %.
```

Число	Стандартный код	Код Грея
0	**000**	**000**
1	**001**	**001**
2	**010**	**011**
3	**011**	**010**
4	**100**	**110**
5	**101**	**111**
6	**110**	**101**
7	**111**	**100**

Табл. Кодирование чисел от 0 до 7.

Генетические алгоритмы несложно программировать. Приведём несколько «набросков MatLab-кода» для реализации некоторых операций.

`ZU = [z, u]'`
`i = rand(size(z))>0.5;`
`v1 = ZU([i, ~i]')`
`v2 = ZU([~i, i]')`
	Равномерный кроссовер

`k = ceil((n-1)*rand(1));`
`v1 = [z(1:k); u(k+1:end)]`
`v2 = [u(1:k); z(k+1:end)]`
	Одноточечный кроссовер

`i = rand(size(z))<epsilon` `z(i) = ~z(i)`	Мутация

Роевой алгоритм

Изложим лишь основную идею **роевого алгоритма**, который чаще применяется для оптимизации функций вида $f:[0,1]^n \to \mathbf{R}$ или $f:\mathbf{R}^n \to \mathbf{R}$. В этом алгоритме также случайно выбирается k точек пространства. Можно представить, что точки – пчёлы роя (их координаты), «полёт которых по пространству» в большой степени случаен, но каждая пчела

1. Стремится к точке с максимальным значением, которую она успела найти.
2. Стремится к точке с максимальным значением из всех точек, которые успел посетить весь рой.
3. Стремится к точке с максимальным значением, которую успела найти одна из её подруг (точка-подруга выбирается случайно из роя).

Все три направления стремления пчелы суммируются (быть может, с какими-то случайными коэффициентами), и она совершает движение в этом суммарном направлении.

Есть интересная идея в теории глобальной оптимизации: привлечение методов обучения по прецедентам. Если вычисление функции f достаточно трудоёмко, то можно спрогнозировать её значения на базе уже известных (получаем задачу регрессии) или хотя бы определить область больших значений (задача классификации).

тип алгоритма	**достоинства и недостатки**
Полный перебор	Высокая точность при экспоненциальном времени работы.
Направленный перебор (градиентный, метод луча и т.д.)	Для многих классов функций приемлемая точность, простая реализация, быстрая работа. Часто нет механизмов «достаточно полного» перебора пространства.
Стохастический перебор (генетический алгоритм и т.д.)	Хорошо работает при удачном выборе всех параметров, простой в реализации и модификации, избегает локальных максимумов. Но удачный выбор параметров – отдельная задача оптимизации!

Применение рассмотренных алгоритмов иногда связывают с проблемой «**Exploration vs. Exploitation**», основная суть которой состоит в том, что надо одновременно преследовать две цели

1. Просмотреть как можно больше (новых) точек из всего пространства поиска (задача исследования).
2. Не пропустить хорошее решение и по максимуму использовать уже полученную информацию (задача использования).

Например, преследуя первую цель, надо увеличивать размер окрестности в градиентном алгоритме (чтобы исследовать как можно больше новых точек), преследуя вторую – уменьшать, чтобы не тратить время на потенциально плохие точки, а заняться исследованием области, где значения функции достаточно высокие. Аналогично, вероятность мутации согласно первой цели надо увеличить, а согласно второй – уменьшить.

Хорошим примером задачи «**Exploration vs. Exploitation**» является **задача про 3 баннера**. Допустим, у Вас есть сайт, на котором можно

разместить 3 баннера и рекламодатели, которые платят за каждый клик по их баннеру, размещённому на Вашем сайте. Таким образом, надо максимизировать количество кликов (возможно, с весами, если все платят по-разному). «Интенсивность кликов» по баннерам определяется в большой степени «интересностью» темы **в данный момент времени**: на сайт про олимпиаду будут заходить перед и во время её проведения, сайт Интернет-магазина «Всё для зимнего отдыха» будет пользоваться спросом в осенне-зимний период, персональная страничка поп-звезды будет особенно популярной, если недавно произошёл крупный скандал с её участием. Таким образом, некоторые баннеры «популярны» с какой-то периодичностью, некоторые случайно время от времени, некоторые достаточно стабильно долгий период, но затем перестают быть популярными. Если установить «закономерность популярности» какого-нибудь баннера, то удастся выставлять его на сайт в нужное время. Для этого надо, чтобы он изначально долго был на сайте для сбора статистики его популярности. С другой стороны, когда на сайте долго «висит» один и тот же баннер, мы ничего не знаем про другие и теряем потенциальные деньги (не решаем задачу исследования). Конечно, используя три баннерных позиции, можно одновременно решать три задачи: на одной позиции держать баннер для исследования (его статистики), на другой – менять баннеры для исследования (всего пространства), на третьей – ставить уже исследованный для «зарабатывания денег» (т.е. использования), но вопрос об оптимальной стратегии всё равно остаётся. Такую задачу хорошо решить на практикуме на ЭВМ, при этом ясно, как моделировать данные. Заметим, что тут оптимизируется функция вида $f : N \times N \times N \times \mathbf{R}^+ \to \mathbf{Z}^+ = \{0,1,2,\ldots\}$ (первые три аргумента – номера баннеров из множества N, а последний – время).

Хороший обзор по алгоритмам оптимизации «чёрного ящика» представлен в [Luke, 2009] (там такие алгоритмы названы «метаэвристиками»).

Глава 9. АЛГОРИТМЫ, ОСНОВАННЫЕ НА ВЫЧИСЛЕНИИ ОЦЕНОК

§9.1. Тестовые алгоритмы

Рассмотрим задачу классификации на два непересекающихся класса с бинарными признаками. Описания объектов обучения из первого и второго классов не пересекаются (т.е. начальная информация непротиворечива). Для простоты будем всё пояснять на примере[1]. Признаковые описания объектов можно по строкам записать в $(m_1 + m_2) \times n$ матрицу «объект-признак», где m_j – число объектов j-го класса, $j \in \{1,2\}$. Пусть эта матрица

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} \} \in K_1 \\ \\ \\ \} \in K_2 \\ \\ \\ \end{matrix} .$$

с ij-м элементом равным $f_j(x^i)$.

Построим всевозможные тупиковые тесты матрицы «объект-признак». **Тестом** называется такое множество столбцов (или, для простоты, множество их номеров), что в образованной ими подматрице строки-подописания объектов из разных классов различны. Например, столбцы с номерами 4 и 5 образуют тест, поскольку в подматрице

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{matrix} \} \in K_1 \\ \\ \\ \} \in K_2 \\ \\ \\ \end{matrix}$$

матрицы «объект-признак» первые три строки отличны от последних трёх строк. Тест называется **тупиковым тестом**, если никакое его собственное подмножество тестом не является. Таким образом, $\{3,4,5\}$ – тест, но не тупиковый, поскольку множество $\{4,5\}$ является тестом, причём уже тупиковым, ведь множества $\{4\},\{5\}$ тестами не являются. Неформально говоря, тест соответствует задаче, из которой удалили некоторые признаки, но её постановка осталась непротиворечивой, а тупиковой тест – такой «минимальной» задаче (больше уже из неё не получится удалить признаки без нарушения непротиворечивости).

Итак, по матрице «объект-признак» строим новую матрицу со строками

$$(f_1(x^i) \oplus f_1(x^t), \ldots, f_n(x^i) \oplus f_n(x^t))$$

[1] В этой главе вместо описания псевдокода алгоритма будем показывать его работу на конкретном примере, поскольку описания алгоритмов потребуют введения ненужного формализма, а основная идея понятна и без описания.

для всех $x^i \in K_1$, $x^t \in K_2$ (значения складываются по модулю два):

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{matrix} 1 \oplus 4 \\ 2 \oplus 4 \\ 3 \oplus 4 \\ 1 \oplus 5 \\ 2 \oplus 5 \\ 3 \oplus 5 \\ 1 \oplus 6 \\ 2 \oplus 6 \\ 3 \oplus 6, \end{matrix}$$

справа для удобства помечено, какие строки исходной матрицы «объект-признак» складываются. Такая матрица называется **матрицей сравнений**. Если из неё удалить повторяющиеся строки, то получим матрицу

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(удаление производится для сокращения вычислений и не влияет на результат). Рассмотрим первую строку этой матрицы. Она описывает сходство описаний 1-го и 4-го объектов (а также 2-го и 5-го): они отличаются по 2-й и 5-й координатам признакового описания (поэтому 2-й и 5-й элементы этой строки равны единице). А все строки матрицы описывают сходства и отличия признаковых описаний пар объектов из разных классов.

Матрица сравнений описывает, какие столбцы надо взять в тест. Например, первая строка матрицы «говорит», что в тест должен входить 2-й или[1] 5-й столбец, а иначе описания двух объектов из разных классов совпадут. Вторая строка «говорит», что в тест должен войти хотя бы один столбец с номером из {1,2,3,4} и т.д. Эту информацию можно формализовать в виде логического выражения:

$$(f_2 \vee f_5) \& (f_1 \vee f_2 \vee f_3 \vee f_4) \& (f_3 \vee f_4) \& (f_1 \vee f_5)$$

(в качестве переменных для удобства мы взяли обозначения наших признаков). Раскроем скобки в этом выражении, только сделаем это «по-умному»:

$$((f_2 \vee f_5) \& (f_1 \vee f_5)) \& ((f_1 \vee f_2 \vee f_3 \vee f_4) \& (f_3 \vee f_4)) =$$
$$(f_1 \& f_2 \vee f_5) \& (f_3 \vee f_4),$$

здесь пользуемся очевидным свойством поглощения:

$$(a \vee b) \& (a \vee c) = a \vee bc.$$

В итоге получаем выражение

$$f_1 \& f_2 \& f_3 \vee f_1 \& f_2 \& f_4 \vee f_3 \& f_5 \vee f_4 \& f_5.$$

Если вспомнить, что переменная f_t кодирует фразу «t-й столбец принадлежит тесту», то получим, что тестами будут множества столбцов

$$\{1,2,3\}, \{1,2,4\}, \{3,5\}, \{4,5\}.$$

[1] Здесь неисключающее ИЛИ.

Нетрудно доказать, что описанный нами алгоритм корректный, т.е. действительно получается **множество всех тупиковых тестов**, если
1. Построить по матрице «объект-признак» матрицу сравнения,
2. По матрице сравнения выписать КНФ[1] (конъюнктивную нормальную форму) монотонной булевой функции,
3. Преобразовать КНФ в сокращённую ДНФ.

Существенно (для построения именно всех **тупиковых** тестов), что на последнем этапе мы использовали правила поглощения (можно формально перемножить скобки, а потом вычёркивать из ДНФ конъюнктивные слагаемые по правилу поглощения[2] $a \vee a \,\&\, b \to a$ пока это возможно), это гарантирует построение именно сокращённой ДНФ.

Последние этапы 2-3 нацелены на построение всех тупиковых покрытий матрицы сравнений. Множество столбцов называется **покрытием**, если в образованной ими подматрице нет нулевой строки. Покрытие называется **тупиковым (неприводимым) покрытием**, если никакое его собственное подмножество покрытием не является. **ДЗ** *Докажите строго, что задача о построении множества всех тупиковых тестов в задаче с двумя непересекающимися классами и бинарными признаками сводится к задаче построения множества всех тупиковых покрытий матрицы сравнений.*

Существуют и другие (более эффективные) способы построения множества всех тупиковых покрытий, см. [Дюкова, 2003]. **ДЗ** *Реализуйте алгоритм построения множества всех тупиковых покрытий по бинарной матрице.*

Тест даёт нам простую задачу (со сравнительно небольшим числом признаков), в которой информация всё ещё непротиворечива (объекты разных классов различны). Можно решить её **любым алгоритмом классификации** (ниже приведена формула для совсем тривиального алгоритма), собрать решения задач для всех тестов и устроить голосование (возможно с весами). Итак, вычисляем значение **оценки принадлежности объекта** x **к** j**-му классу:**

$$\Gamma_j(x) = \frac{1}{N_j} \sum_{\Omega \in \Omega^*} \sum_{x^t \in \tilde{K}_j} \left(w(\Omega) \cdot w^t \cdot \prod_{\omega \in \Omega} I[\, f_\omega(x^t) = f_\omega(x)] \right),$$

[1] Читателю (студенту 3го курса ВМК МГУ) должны быть известны термины «конъюнктивная нормальная форма» (КНФ) и «дизъюнктивная нормальная форма» (ДНФ).
[2] Название привила вполне естественное: выражение a «поглощает» выражение $a \,\&\, b$, поскольку $a \vee (a \,\&\, b) = a$.

здесь Ω^* – множество всех тупиковых тестов, $w(\Omega)$ – вес теста Ω («степень доверия» этому тесту), w^t – вес объекта x^t («степень доверия» этому объекту), N_j – некоторый нормировочный множитель (часто бывает логично положить $N_j = |m_j|), \tilde{K}_j = \{x^t \in K_j \,|\, t \in \{1,2,\ldots,m\}\}$ (объекты из обучения, которые принадлежат j-му классу). Объект относим к тому классу, оценка принадлежности к которому выше. Это и есть простейший **тестовый алгоритм классификации.**

Пример. Для нашего примера классифицируем объект $(1,1,1,1,1)$. Оценка принадлежности к первому классу:

$$\Gamma_1(1,1,1,1,1) = \frac{1}{N_1}\left(w(\{1,2,4\})\cdot w^1 + w(\{3,5\})\cdot w^2 + w(\{3,5\})\cdot w^3\right),$$

оценка принадлежности ко второму классу:

$$\Gamma_2(1,1,1,1,1) = \frac{1}{N_2}\left(w(\{4,5\})\cdot w^4 + w(\{4,5\})\cdot w^6\right).$$

Если все веса положить равными единице, то объект следует отнести к первому классу.

Веса обычно выбираются

1) экспертом (если есть априорная информация о важности объектов и признаков),

2) получаются в результате решения задачи оптимизации: обучающая выборка разбивается на две части, по первой – строятся тесты, по второй – выбираются веса так, чтобы максимизировать функционал качества алгоритма.

Замечание. Вес теста задать эксперту достаточно сложно, гораздо проще задать веса признаков $\{w_\omega\}_{\omega=1}^n$, а вес теста определять по весам, входящих в него признаков:

$$w(\Omega) = \frac{1}{|\Omega|}\sum_{\omega\in\Omega} w_\omega .$$

§9.2. Алгоритмы с представительными наборами

Опишем ещё один алгоритм решения нашей задачи. Выпишем матрицу описаний обучающих объектов второго класса

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Пусть некоторая булева функция $F:\{0,1\}^n \to \{0,1\}$, $n=5$, обращается в ноль только в точках, перечисленных построчно в этой матрице. Тогда

$$F(f_1, f_2, f_3, f_4, f_5) =$$

$$=(\neg f_1 \vee f_2 \vee f_3 \vee \neg f_4 \vee \neg f_5)\,\&\,(f_1 \vee f_2 \vee \neg f_3 \vee f_4 \vee f_5)\,\&\,(f_1 \vee \neg f_2 \vee f_3 \vee \neg f_4 \vee \neg f_5).$$

Выполним конъюнктивные перемножения, как всегда «по-умному», сначала умножив первую и третью скобки, везде применяя поглощения:

$$(f_1\,\&\,f_2 \vee \neg f_1\,\&\,\neg f_2 \vee f_3 \vee \neg f_4 \vee \neg f_5)\,\&\,(f_1 \vee f_2 \vee \neg f_3 \vee f_4 \vee f_5) =$$
$$= f_1\,\&\,f_2 \vee \neg f_1\,\&\,\neg f_2\,\&\,\neg f_3 \vee \neg f_1\,\&\,\neg f_2\,\&\,f_4 \vee \neg f_1\,\&\,\neg f_2\,\&\,f_5 \vee f_1\,\&\,f_3 \vee$$
$$\vee\, f_2\,\&\,f_3 \vee f_3\,\&\,f_4 \vee f_3\,\&\,f_5 \vee f_1\,\&\,\neg f_4 \vee f_2\,\&\,\neg f_4 \vee \neg f_3\,\&\,\neg f_4 \vee$$
$$\vee\, \neg f_4\,\&\,f_5 \vee f_1\,\&\,\neg f_5 \vee f_2\,\&\,\neg f_5 \vee \neg f_3\,\&\,\neg f_5 \vee f_4\,\&\,\neg f_5.$$

Теперь выпишем матрицу описаний обучающих объектов первого класса

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix},$$

оставим в нашей ДНФ только те конъюнкции, которые обращаются в единицу хотя бы на каком-то описании объектов первого класса (хотя бы на одной строке матрицы), а остальные конъюнкции вычеркнем. Получим ДНФ

$$f_1\,\&\,f_2 \vee f_1\,\&\,f_3 \vee f_2\,\&\,f_3 \vee f_3\,\&\,f_5 \vee f_1\,\&\,\neg f_4 \vee f_2\,\&\,\neg f_4 \vee$$
$$\vee\, \neg f_4\,\&\,f_5 \vee f_1\,\&\,\neg f_5 \vee f_2\,\&\,\neg f_5 \vee \neg f_3\,\&\,\neg f_5 \vee f_4\,\&\,\neg f_5.$$

Рассмотрим любую элементарную конъюнкцию этой ДНФ. Например, конъюнкция $f_2\,\&\,\neg f_4$ «говорит», что объект вида $(-,1,-,0,-)$ встречается среди обучающих объектов первого класса, но не встречается среди обучающих объектов второго класса (прочерки соответствуют любым значениям). Причём если из конъюнкции удалить хотя бы одну букву, то такое свойство нарушается. По сути, конъюнкция задаёт пару $((2,4),(1,0))$ «набор номеров признаков», «значения на этих признаках». Такая пара называется **тупиковым представительным набором**, а ДНФ соответствует множеству всех тупиковых представительных наборов.

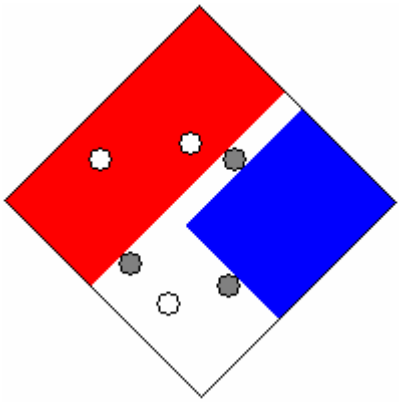


Рис 9.2.1. Интервал, покрывающий точки своего класса, и интервал, не покрывающий точки своего класса.

Дадим геометрическую интерпретацию описанному на примере алгоритму построения всех тупиковых представительных наборов. Множество наборов, которые обращают элементарную конъюнкцию в единицу, образуют **булев подкуб** (его часто называют **интервалом**). Первая построенная ДНФ (до

вычёркивания) соответствует всевозможным интервалам, которые не задевают точки второго класса, при этом их расширения (интервалы, которые соответствуют конъюнкциям после удаления в них какой-то буквы) таким свойством не обладают: в них сразу же попадает точка второго класса. Среди этих интервалов есть те, которые и не содержат точек первого класса (именно такие конъюнкции мы вычёркиваем). Любой из оставшихся интервалов содержит точку первого класса и как бы расширяет её (сама точка является 0-мерным интервалом) до тех пор, пока границы расширения не упрутся в точки второго класса. И множество тупиковых представительных наборов всевозможными подобными расширениями «аппроксимирует наш первый класс».

Простейший **алгоритм с представительными наборами** строится следующим образом: синтезируется ДНФ D_1, которая расширяет «первый класс» (соответствует множеству Λ_1 всех тупиковых представительных наборов первого класса), и ДНФ D_2, которая расширяет «второй класс» (соответствует множеству Λ_2 всех тупиковых представительных наборов второго класса), для классифицируемого объекта x вычисляются оценки принадлежностей к классам

$$\Gamma_j(x) = \frac{1}{N_j} \sum_{K \in D_j} w(K) \cdot K(\tilde{x}),$$

N_j – нормировочный множитель, $w(K)$ – вес конъюнкции K, объект относится к тому классу, оценка принадлежности к которому выше.

Формулу для оценки принадлежностей можно переписать в терминах представительных наборов:

$$\Gamma_j(x) = \frac{1}{N_j} \sum_{((\omega_1,\ldots,\omega_s),(h_1,\ldots,h_s))=\lambda \in \Lambda_j} w(\lambda) \cdot \prod_{i=1}^{s} I[h_i = f_{\omega_i}(x)],$$

$$I[h_i = f_{\omega_i}(x)] = \begin{cases} 1, & h_i = f_{\omega_i}(x), \\ 0, & h_i \neq f_{\omega_i}(x), \end{cases}$$

Замечание. Не обязательно строить множество **всех** представительных наборов. Один из методов построения алгоритма с представительными наборами заключается в том, чтобы изначально строить не сокращённые ДНФ по матрицам нулевых наборов (для первого и второго классов), а произвольные. Известно, что **методом Блейка**, применяя к произвольной ДНФ операции поглощения и обобщённого склеивания

$a \,\&\, A \vee \bar{a} \,\&\, B \to a \,\&\, A \vee \bar{a} \,\&\, B \vee A \,\&\, B$ (обобщённое склеивание по переменной a),

можно получить сокращённую. Поэтому множества тупиковых представительных наборов строят по произвольным ДНФ, а в случае плохого качества классификации ДНФ расширяют по методу Блейка (при этом

переменные, по которым производится обобщённое склеивание, можно выбирать так, чтобы качество алгоритма повышалось).

Замечание. В алгоритме «Кора», который успешно зарекомендовал себя на практике, строятся представительные наборы длины не выше 3х (что можно сделать полным перебором). Такую же идею можно использовать в тестовом алгоритме: строить лишь «небольшие» тесты. Интуитивно понятно, что чем короче тест, тем больше его обобщающая способность.

§3. Алгоритмы вычисления оценок

Используем идею тестовых алгоритмов для задачи с произвольными признаками. Пусть признаковое пространство $M_1 \times \ldots \times M_n$, где M_i – метрическое пространство с метрикой ρ_i (или просто множество, на котором введена функция сходства ρ_i, аксиомы метрики не будут использованы). Пусть Ω – множество номеров некоторых признаков, т.е. $\varnothing \neq \Omega \subseteq \{1,2,\ldots,n\}$. Такое множество называется **опорным**, а совокупность Ω^* опорных множеств, по которой синтезируется алгоритм, называется **системой опорных множеств**. Для любого опорного множества определена **функция близости на данном опорном множестве** $\mathrm{B}_\Omega(x^t, x)$, которая зависит от подописаний $(f_\omega(x))_{\omega\in\Omega}$ и $(f_\omega(x^t))_{\omega\in\Omega}$ объектов x и x^t.

Примеры функций близости:

$$\mathrm{B}_\Omega^{\varepsilon}(x^t,x) = \begin{cases} 1, & \sum\limits_{\omega\in\Omega} \rho_\omega(f_\omega(x^t), f_\omega(x)) \le \varepsilon, \\ 0, & \text{иначе,} \end{cases}$$

$$\mathrm{B}_\Omega^{\varepsilon_1,\ldots,\varepsilon_n}(x^t,x) = \begin{cases} 1, & \forall \omega \in \Omega \ \ \rho_\omega(f_\omega(x^t), f_\omega(x)) \le \varepsilon_\omega, \\ 0, & \text{иначе,} \end{cases} \qquad (9.3.1)$$

$$\mathrm{B}_\Omega^{\varepsilon_1,\ldots,\varepsilon_n,q_1,q_2}(x^t,x) = \begin{cases} 1, & \begin{cases} |\{\omega\in\Omega \mid \rho_\omega(f_\omega(x^t), f_\omega(x)) \le \varepsilon_\omega\}| \ge q_1, \\ |\{\omega\in\Omega \mid \rho_\omega(f_\omega(x^t), f_\omega(x)) > \varepsilon_\omega\}| \le q_2, \end{cases} \\ 0, & \text{иначе.} \end{cases}$$

Последняя функция обращается в единицу, если объекты x и x^t близки с точностью до порогов $\varepsilon_1,\ldots,\varepsilon_n$ на не менее чем q_1 признаках опорного множества Ω и далеки не более чем на q_2 признаках этого опорного множества.

Алгоритм вычисления оценок является суперпозицией оператора вычисления оценок (распознающего оператора) и решающего правила. В задаче с l классами (вообще говоря, пересекающимися) каждый объект x имеет метку вида

$$y(x) = (y_1(x),\ldots,y_l(x)),$$

где $y_j(x)=1$, если объект x принадлежит j-му классу, и $y_j(x)=0$ иначе. **Оператор вычисления оценок** по обучающей выборке $\{(x^t, y(x^t))\}_{t=1}^m$ и контрольной выборке $\{x_t\}_{t=1}^q$ строит $q\times l$**-матрицу оценок принадлежностей контрольных объектов классам**, ij-й элемент которой

$$\Gamma_j(x_i)=\frac{1}{N_j}\sum_{\Omega\in\Omega^*}\sum_{x^t\in\tilde{K}_j} w(\Omega)\cdot w^t\cdot \mathrm{B}_\Omega(x^t,x) \qquad (9.3.2)$$

(смысл всех параметров был объяснён при описании тестового алгоритма). **Решающее правило** получает по этой матрице бинарную $q\times l$**-матрицу классификаций**, ij-й элемент которой соответствует ответу на вопрос о принадлежности i-го объекта j-му классу (равен 1, если алгоритм «говорит», что принадлежность есть).

Решающие правила достаточно простые. Например, **пороговое решающее правило –**

$$C(\| \gamma_{ij} \|_{q\times l}) = \| y_{ij} \|_{q\times l},$$

где

$$y_{ij}=\begin{cases}1, & \gamma_{ij}\ge c,\\ 0, & \gamma_{ij}<c.\end{cases}$$

Если допускается, что алгоритм может отказываться от классификации, то рассматривают решающее правило с двумя порогами:

$$y_{ij}=\begin{cases}1, & \gamma_{ij}\ge c_1,\\ \Delta, & c_2\le\gamma_{ij}<c_1,\\ 0, & \gamma_{ij}<c_2.\end{cases}$$

(символ Δ интерпретируют как отказ от классификации).

Нетрудно видеть, что в задаче с бинарными признаками при специальном выборе функции близости и системы опорных множеств, совпадающей с множеством всех тупиковых тестов, алгоритм вычисления оценок «превращается» в тестовый алгоритм.

В общем случае функция близости не обязана принимать значения из $\{0,1\}$, кроме того, используют более общие формулы оценки близости:

$$\Gamma_j(x_i)=\sum_{a,b=0,0}^{1,1} N_j^{a,b}\cdot\sum_{\Omega\in\Omega^*}\sum_{x^t\in\tilde{K}_j^a} w(\Omega)\cdot w^t\cdot[\mathrm{B}_\Omega(x^t,x)]^b,$$

$\tilde{K}_j^1=\tilde{K}_j$, $\tilde{K}_j^0=\{x^t\}_{t=1}^m\setminus\tilde{K}_j$, $[\mathrm{B}]^1=\mathrm{B}$, $[\mathrm{B}]^0=1-\mathrm{B}$, $N_j^{a,b}$ – некоторый нормировочный коэффициент, например

$$N_j^{a,b}=\frac{(-1)^{a+b}}{|\tilde{K}_j^a|}.$$

Здесь слагаемое при $a=1,\ b=1$ является «классическим» и имеет простой смысл: объект из класса должен быть близок к обучающим объектам из этого класса, слагаемое при $a=0,\ b=0$ учитывает, что при этом он должен быть далёк от остальных обучающих объектов. Отрицательное слагаемое при $a=1,\ b=0$ (отрицательное, поскольку нормировочный коэффициент $N_j^{1,0}$ выбирается отрицательным) учитывает, что надо понижать оценку за j-й класс, если объект далёк от представителей этого класса, а отрицательное слагаемое при $a=0,\ b=1$ «наказывает» за близость к представителям других классов.

§9.4. Эффективные формулы вычисления оценок и оптимизация по весам

Рассмотрим алгоритм вычисления оценок с формулой (9.3.2) (для простоты выкладок), функцией близости (9.3.1), системой опорных множеств $\Omega^* = 2^{\{1,2,\ldots,n\}} \setminus \{\varnothing\}$ (все непустые подмножества множества признаков) и весом опорного множества

$$w(\Omega) = \sum_{\omega\in\Omega} w(\omega).$$

Тогда в формуле (9.3.2) содержится экспоненциальное число слагаемых, и она становится практически не вычислима при большом числе признаков. «Спасает» тот факт[1], что

$$\Gamma_j(x_i) = \frac{1}{N_j}\sum_{x^t\in\tilde{K}_j} w^t \cdot \sum_{\Omega\in\Omega^*}\left(\sum_{\omega\in\Omega} w(\omega)\right)\cdot \mathrm{B}_\Omega^{\varepsilon_1,\varepsilon_2,\ldots,\varepsilon_n}(x^t,x) =$$

$$\frac{1}{N_j}\sum_{x^t\in\tilde{K}_j} w^t \cdot \sum_{\omega=1}^{n} w(\omega)\cdot |\{\Omega\in\Omega^* \mid \omega\in\Omega,\ \mathrm{B}_\Omega^{\varepsilon_1,\varepsilon_2,\ldots,\varepsilon_n}(x^t,x)=1\}|.$$

В этой формуле уже совсем «немного» слагаемых, если уметь быстро вычислять значения

$$V(\omega) = |\{\Omega\in\Omega^* \mid \omega\in\Omega,\ \mathrm{B}_\Omega^{\varepsilon_1,\varepsilon_2,\ldots,\varepsilon_n}(x^t,x)=1\}|$$

(число опорных множеств с признаком ω, на которых функция близости обращается в единицу). Нетрудно доказать, что

$$V(\omega) = \begin{cases} 2^{|I|-1}, & \omega\in I,\\ 0, & \omega\notin I, \end{cases}$$

где $I = \{\omega\in\Omega \mid \rho_\omega(f_\omega(x^t), f_\omega(x)) \le \varepsilon_\omega\}$.

Для других систем опорных множеств также существуют такие «эффективные формулы вычисления оценок». Например, для

$$\Omega^* = \{\Omega\in 2^{\{1,2,\ldots,n\}} \mid \|\Omega\| = k\}$$

(все опорные множества мощности k) при $|I|<k$ выполняется $V(\omega)=0$ для всех $\omega\in\{1,2,\ldots,n\}$, а при $|I|\ge k$

[1] Впервые отмеченный Ю.И. Журавлёвым.

$$V(\omega)=\begin{cases} C_{|I|-1}^{k-1}, & \omega\in I,\\ 0, & \omega\notin I. \end{cases}$$

Итак, формула вычисления оценок имеет «эффективный» вид:

$$\Gamma_j(x_i)=\frac{1}{N_j}\sum_{x^t\in\tilde{K}_j} w^t\cdot\sum_{\omega=1}^{n} w(\omega)\cdot V(\omega)=\frac{1}{N_j}\sum_{t=1}^{m}\sum_{\omega=1}^{n} w^t\cdot w(\omega)\cdot V(\omega)\cdot I[x^t\in\tilde{K}_j],$$

т.е. это билинейная форма по переменным $w^1,\ldots,w^m$ (веса объектов) и $w(1),\ldots,w(n)$ (веса признаков). Если мы знаем ответы на контрольной выборке (изначально мы выборку с известной классификацией разбили на обучение и контроль[1]), то при использовании порогового решающего правила для верной классификации контрольной выборки необходима совместность системы неравенств

$$\Gamma_j(x_i)\ge c \text{ при } y_j(x_i)=1,$$
$$\Gamma_j(x_i)<c \text{ при } y_j(x_i)=0.$$

Это система билинейных неравенств. **ДЗ** *Попробуйте настраивать веса алгоритма следующим способом. Сначала фиксируете одну группу переменных* (*например* $w^1,\ldots,w^m$) *и решаете систему* (*линейных неравенств*) *относительно второй группы* (*т.е.* $w(1),\ldots,w(n)$), *затем наоборот. При решении системы линейных неравенств можно использовать стандартные методы настройки линейного классификатора.*

[1] Правильнее использовать следующую терминологию. Выборка может быть **обучающей** (используется для обучения), **валидационной** (для настройки некоторых метапараметров), **контрольной** (для оценки работы алгоритма).

Глава 10. ОБЪЕДИНЕНИЕ ЭЛЕМЕНТАРНЫХ КЛАССИФИКАТОРОВ

§10.1. Бустинг, AdaBoost

Бустинг (boosting) является очень простым и эффективным способом решения задач классификации (и многих других задач анализа данных), используя простые семейства алгоритмов. Рассмотрим простейшую задачу классификации на два класса: $Y = \{+1,-1\}$. Припишем каждому объекту x^t вес w^t. Здесь можно трактовать веса как вероятность предъявления этого объекта классификатору. Таким образом,

$$\sum_{t=1}^{m} w^t = 1,$$

а вероятность ошибки классификатора h

$$e(h) = \sum_{\substack{t=1,\\ h(x^t)\neq y(x^t)}}^{m} w^t .$$

Будем считать, что для любого распределения весов $W = (w^1,\ldots,w^m)$ можно построить классификатор (из некоторого фиксированного семейства), у которого вероятность ошибки меньше 0.5 (вполне естественное предположение). Будем последовательно строить классификаторы $h_1,\ldots,h_s$, объединяя их в один классификатор H:

$$H(x) = \operatorname{sgn}\left(\sum_{j=1}^{s} \alpha_j h_j(x)\right).$$

Построение можно производить по следующему алгоритму.

Алгоритм AdaBoost[1]

0. Пусть начальное вероятностное распределение

$$W = \left(\frac{1}{m},\ldots,\frac{1}{m}\right).$$

1. Цикл по j от 1 до s.

1.1. Построить классификатор h_j, который допускает ошибку $e(h_j)$ (вычисляется по распределению W).

1.2. Пусть

$$\alpha_j = \frac{1}{2}\ln\left(\frac{1-e(h_j)}{e(h_j)}\right).$$

«Перестроить» распределение $W = (w^1,\ldots,w^m)$:

[1] Алгоритм описан в [*Freund Y., Schapire R.* A Short Introduction to Boosting // Journal of Japanese Society for Artificial Intelligence, 14(5): 771 – 780].

$$w^t = \frac{w^t \exp(-\alpha_j y(x^t) h_j(x^t))}{\sum_{t=1}^{m} w^t \exp(-\alpha_j y(x^t) h_j(x^t))}.$$

2. Ответ алгоритма – классификатор

$$H(x) = \operatorname{sgn}\left(\sum_{j=1}^{s} \alpha_j h_j(x) \right).$$

Замечание. При «перестройке» распределения мы умножаем старую вероятность w^t на $\exp(-\alpha_j)$, если классификатор h_j не ошибается на объекте x^t, и на $\exp(+\alpha_j)$ – в противном случае.

Замечание. Описанная процедура называется усилением (boosting) классификаторов. При этом классификаторы h_j называются «слабыми» (weak[1]). На шаге 1.1, естественно, надо стремиться построить классификатор с наименьшей ошибкой. Большие веса получают те объекты, которые «плохо» классифицировались на предыдущих шагах. Таким образом, на каждом последующем шаге мы стремимся новым классификатором исправить ошибки предыдущих классификаторов. Это порождает ненужную настройку на шумовые объекты (поэтому иногда объекты с очень большими весами исключают).

ДЗ *Предложите несколько эвристик обнаружения шумовых объектов в процессе работы алгоритма AdaBoost. Проведите эксперименты на модельных данных.*

В качестве элементарных (слабых) классификаторов для бустинга обычно используют очень простые классификаторы, например «пороговые по значению признака»

$$h(x) = \begin{cases} +1, & f_j(x) \geq c, \\ -1, & f_j(x) < c. \end{cases}$$

(т.е. классификаторы, которые сравнивают значение t-го признака с порогом c). Ясно, что таких классификаторов для конкретной задачи конечное число: достаточно рассмотреть все признаки и все пороги (промежуточные значения упорядоченного ряда значений рассматриваемого признака), а их суперпозиция H может разделять даже «сложно устроенные» классы.

[1] Терминология объясняется тем, что качество таких классификаторов может быть довольно низким (необходимо только, чтобы они ошибались меньше, чем в половине случаев). При бустинге качество классификации на обучающей выборке увеличивается (доказано, что оно стремится к 100%).

Проиллюстрируем сказанное выше на примере. Пусть объекты – точки в двухмерном пространстве. При построении классификатора просматриваем все классификаторы вида

$$h(x)=\begin{cases}+1, & f_j(x)\geq f_j(x^t),\\ -1, & f_j(x)<f_j(x^t),\end{cases}\qquad h(x)=\begin{cases}+1, & f_j(x)< f_j(x^t),\\ -1, & f_j(x)\geq f_j(x^t),\end{cases}$$

перебирая $j\in\{1,2\}$ и все эталонные объекты x^t. Выбираем самый лучший среди всех этих классификаторов. Ниже представлен MatLab-код, который реализует эту идею.

Бустинг пороговых классификаторов

```
m = 1000; % число объектов
X = rand([m 2]); % обучение
Y = (X(:,1)-0.5).^2 + (X(:,2)-0.5).^2 < 0.1; % классы

[tmp1 tmp2] = meshgrid(0:0.05:1, 0:0.05:1);
Xc = [tmp1(:) tmp2(:)]; % контроль
q = size(Xc, 1); % объектов в контроле

% алгоритмы
A = [];
% записаны по строкам
% A(i,1) - направление (1 или 2)
% A(i,2) =1 (>= порога) 0 (< порога)
% A(i,3) - порог
% A(i,4) - значение alpha

% визуализация
% scatter(X(:,1),X(:,2),20,Y,'filled')
scatter(X(Y>0,1), X(Y>0,2), 20, X(Y>0), 's', 'r');
hold on;
scatter(X(Y<=0,1), X(Y<=0,2), 20, X(Y<=0), 'o', 'b', 'filled');

% веса объектов
W = ones(size(Y))/m;

% просто инициализация (выделение памяти)
Y1 = zeros(size(Y));
Y2 = zeros(size(Y));

for I = 1:100 % просто построить 100 алгоритмов

    % выбор алгоритма
    % по первому направлению
    [Xs I] = sortrows(X, 1);
    Ys = Y(I);
    Ws = W(I);

    % приём для вычисления ошибки
```

```
    Y1(:) = 0;
    Y1(Ys==1) = Ws(Ys==1);
    Y1 = cumsum(Y1);

    Y2(:) = 0;
    Y2(Ys==0) = Ws(Ys==0);
    Y2 = cumsum(Y2(end:-1:1));
    Y2 = Y2(end:-1:1);

    Y2 = [0; Y1(1:end-1)] + Y2;

    [mx imx] = max(Y2);
    [mn imn] = min(Y2);

    if (mn<1-mx) a = [1 1 Xs(imn,1)]; e = mn;
    else         a = [1 0 Xs(imx,1)]; e = 1 - mx;
    end;

    % по второму
    [Xs I] = sortrows(X,2);
    Ys = Y(I);
    Ws = W(I);

    % приём для вычисления ошибки
    Y1(:) = 0;
    Y1(Ys==1) = Ws(Ys==1);
    Y1 = cumsum(Y1);

    Y2(:) = 0;
    Y2(Ys==0) = Ws(Ys==0);
    Y2 = cumsum(Y2(end:-1:1));
    Y2 = Y2(end:-1:1);

    Y2 = [0; Y1(1:end-1)] + Y2;

    [mx imx] = max(Y2);
    [mn imn] = min(Y2);

    if (mn<1-mx) if (mn<e) a = [2 1 Xs(imn,2)]; e = mn; end;
    else         if (1-mx<e) a = [2 0 Xs(imx,2)]; e = 1 - mx; end;
    end;

    % визуализация
    figure(1);
    if a(1)==1
        plot([abs(a(3)) abs(a(3))], [0 1], 'k', 'LineWidth', 2);
    else
        plot([0 1], [abs(a(3)) abs(a(3))], 'k', 'LineWidth', 2);
    end;

    % бустинг
    alpha = 0.5*log((1-e)/e);
```

```
    W = W.*exp( alpha*(1 - 2*( ((X(:,a(1))>=a(3))==a(2)) == Y )));
    W = W/sum(W);

    % добавление алгоритма
    A(end+1,:) = [a(1:3) alpha];

    % текущий классификатор
    h = (2*((Xc(:, A(:,1))>=repmat(A(:,3)', q, 1)) ==
repmat(A(:,2)', q, 1))-1)*A(:,4)>0;

     figure(2);
    %scatter(Xc(:,1), Xc(:,2), 20, h, 'filled');
    clf;
    scatter(Xc(h>0,1), Xc(h>0,2), 20, 's', 'r');
    hold on;
    scatter(Xc(h<=0,1), Xc(h<=0,2), 20, 'o', 'b', 'filled');
    pause;

end;
```

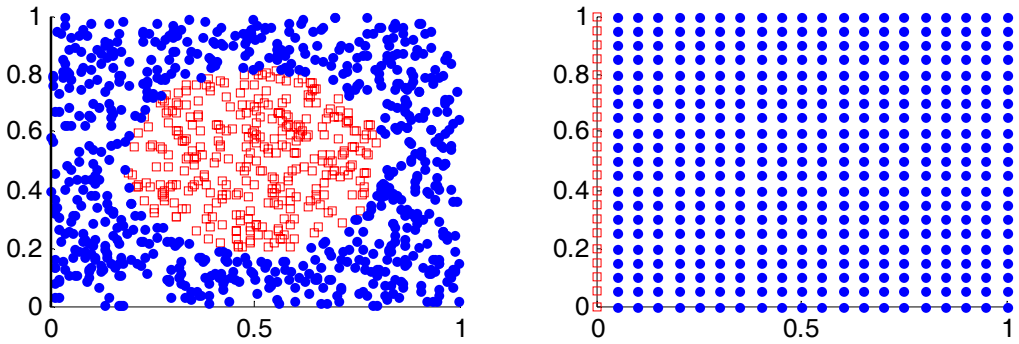


Рис. 10.1. Обучающая выборка (слева) и классификация первого слабого классификатора (справа).

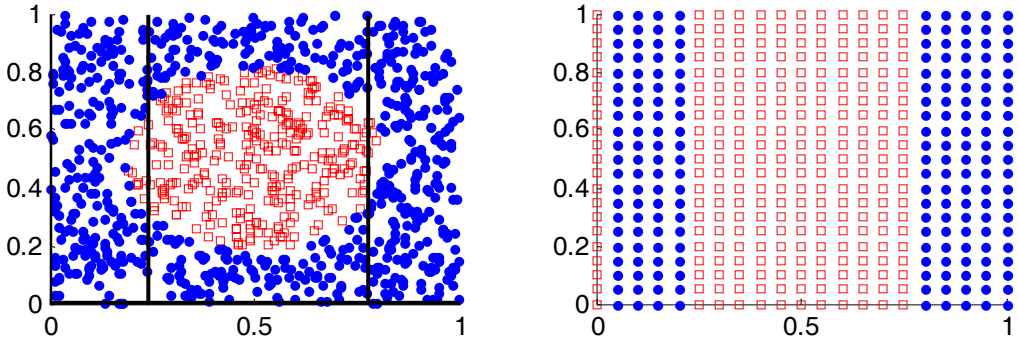


Рис. 10.2. Обучающая выборка с разделяющими линиями 4-х классификаторов[1] (слева) и классификация после 4-х итераций (справа).

[1] Разделяющие линии двух классификаторов почти совпадают с осями координат (проходят через самый левый и самый нижний объекты).

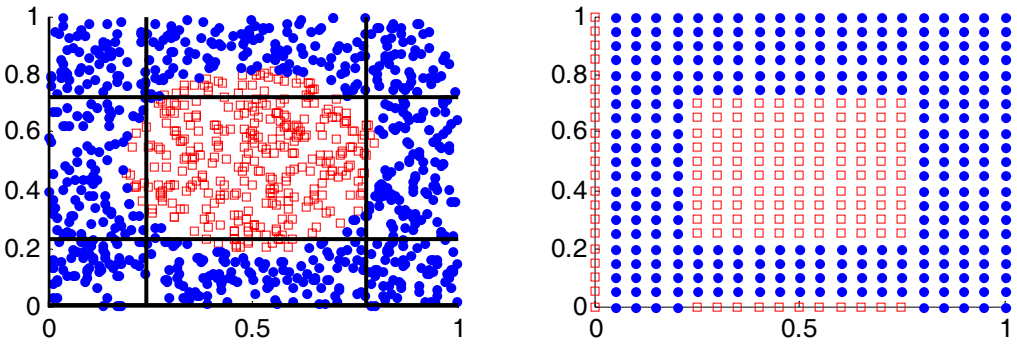


Рис. 10.3. Обучающая выборка с разделяющими линиями 6-ти классификаторов (слева) и классификация после 6-ти итераций (справа).

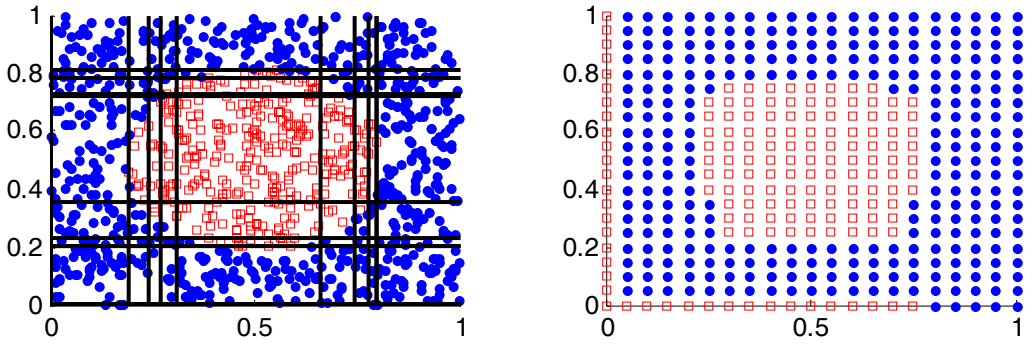


Рис. 10.4. Обучающая выборка с разделяющими линиями 30-ти классификаторов (слева) и классификация после 30-ти итераций (справа).

На рис. 10.1 – 4 показаны выводимые представленной программой графики. В процессе построения классификатора его разделяющая поверхность всё больше напоминает «истинную» – окружность.

ДЗ *Объясните краевые эффекты на рис.* 10.1 – 4. *Обратите внимание, что ошибка порогового классификатора по первому признаку* (*и по второму*) *выглядит так, как показано на рис.* 10.5. *Первый построенный классификатор проводит вертикальную разделяющую линию по самому левому объекту. Реализуйте бустинг без краевых эффектов.*

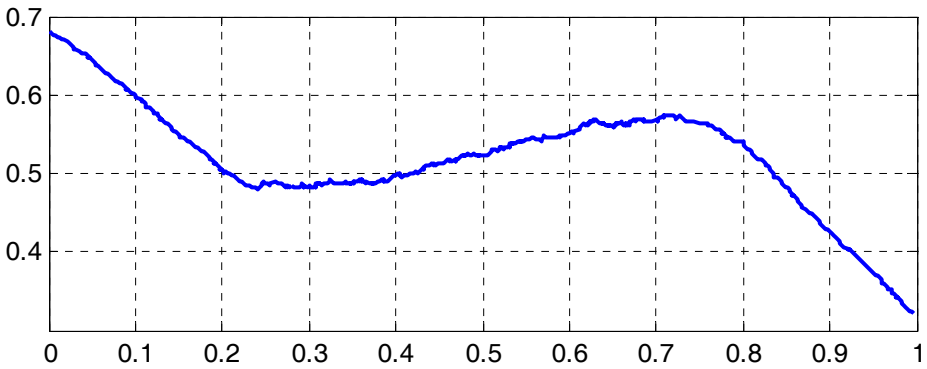


Рис. 10.5. Ошибка порогового классификатора в зависимости от величины порога.

ДЗ *Реализуйте бустинг с другими множествами слабых классификаторов:*

1. *Множеством разделяющих гиперплоскостей* (*случайных или «специально настраиваемых»*).
2. *Множеством алгоритмов типа ближайшего соседа* (*с небольшим числом эталонов*).
3. *Множеством простых логических алгоритмов* (*см. дальше*).

Попробуйте усовершенствовать предложенный выше код, сделать его универсальным (*он, кстати, далёк от совершенства*). *Обратите внимание, что при реализации бустинга необходимо эффективно кодировать алгоритмы* (*в представленном листинге алгоритмы представлены строками матрицы* **A**).

§10.2. Другие варианты бустинга и объединения элементарных классификаторов

Бустинг основан на возможности обучаться на выборке с весами. Вообще говоря, есть следующие варианты бустинга (усиления) элементарных (слабых) классификаторов (см. [Хайкин, 2006]):

1. Фильтрация (для классификаторов специальным образом выбирается обучающая подвыборка из потенциально бесконечной выборки, см. ниже).
2. Формирование подвыборок (классификаторы обучаются на подвыборках, которые выбираются в соответствии с текущим распределением, ошибка вычисляется относительно фиксированного множества объектов).
3. Перевзвешивание (используется фиксированная обучающая выборка, ошибка равна сумме весов неверно классифицированных объектов, см. AdaBoost).

Именно третий подход реализован выше. Второй реализовать относительно просто. Рассмотрим алгоритм первого подхода.

Бустинг фильтрацией

Строим три элементарных классификатора.

1. На конечной выборке (из m_1 элемента) обучаем первый классификатор.
2. Формируем конечную обучающую выборку X_2 (из m_2 элемента) для второго классификатора перебором бесконечной обучающей выборки $x^1, x^2, \ldots$
   - 2.1. Подбрасываем монетку m_2 раз.
   - 2.1.1. Если выпадает решка, то идём по обучающей выборке, пока не встретится объект, на котором первый классификатор ошибается – заносим его в X_2.
   - 2.1.2. Если выпадает орёл, то идём по обучающей выборке, пока не встретится объект, который первый классификатор классифицирует верно – заносим его в X_2.
3. Обучаем второй классификатор на выборке X_2.

4. Формируем конечную обучающую выборку X_3 (из m_3 элемента) для третьего классификатора перебором бесконечной обучающей выборки $x^1, x^2, \ldots$
4.1. Если объект x^j классифицируется первым и вторым классификатором по-разному, то включаем его во множество X_3 (и так до тех пор, пока не наберём m_3 элементов).
5. Обучаем третий классификатор на множестве X_3.
6. Классификацию осуществляем с помощью построенных трёх алгоритмов по большинству голосов.

Замечание. Описанный процесс формирования обучающей выборки с помощью результатов классификации уже построенных алгоритмов и называется **фильтрацией**. Например, после первой фильтрации первый алгоритм ошибается на половине объектов множества X_2. Таким образом, построенный второй алгоритм будет «не похож» на первый, а ведь голосование именно «непохожих» алгоритмов имеет смысл. Вторая фильтрация необходима, чтобы третий алгоритм «разрешал споры» между первым и вторым. Доказано, что если каждый из трёх алгоритмов имеет ошибку $e < 0{,}5$, то голосование трёх алгоритмов имеет ошибку не выше $3e^2 - 2e^3$. На рис. 10.2.1 показано, что три алгоритма имеют ошибку меньше, чем ошибка каждого отдельного алгоритма. Рекурсивно применяя бустинг фильтрацией, можно сколь угодно понизить ошибку.

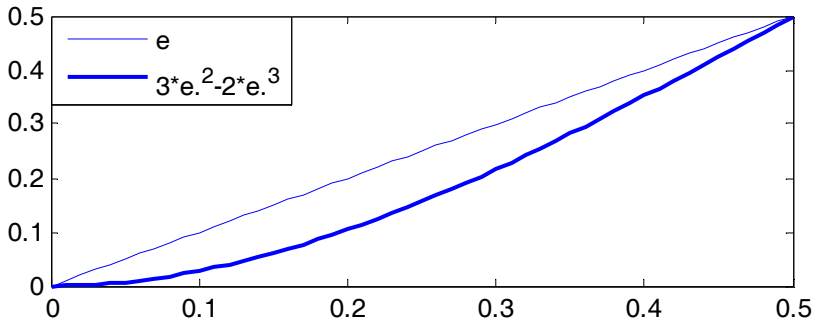


Рис. 10.2.1. График ошибки $3e^2 - 2e^3$.

Замечание. В западной литературе комитет большинства (построенный выше), т.е. «мета-алгоритм», который классифицирует по большинству голосов входящих в него элементарных классификаторов, принято называть ассоциативной машиной.

Бустинг строит один алгоритм на базе нескольких. Есть и другие способы объединения нескольких алгоритмов в один. При их использовании часто бывает полезно не просто строить алгоритмы семейства (точнее, настраивать их параметры), а строить **различные** алгоритмы (чтобы они ошибались по-разному и ошибки одних компенсировались правильными ответами других).

Для построения различных алгоритмов, как правило, используют фильтрацию (описана выше), бэггинг и метод случайных подпространств.

В **методе случайных подвыборок (бэггинге)** с помощью бутстрепа несколько раз формируют обучающую выборку и на ней настраивают алгоритм. Поскольку обучающая выборка каждый раз новая, то и алгоритмы будут получаться разные, если метод настройки существенно зависит от выборки (например, метод SVM строит гиперплоскость по опорным объектам, а от остальных объектов его результат не зависит, поэтому для бэггинга SVM не подходит).

В **методе случайных подпространств** перед обучением случайно выбирают признаки, которые используются для обучения (т.е. сужают признаковое пространство). Метод можно обобщить: каждому признаку приписать вес, после настройки алгоритма на признаковом подпространстве у всех признаков подпространства меняется вес (в зависимости от качества работы алгоритма). При дальнейшей генерации признаковых подпространств веса учитываются так, чтобы «хорошие» признаки выбирались чаще. В [Воронцов] предлагается использовать комбинацию бэггинга и метода случайных подпространств.

Вообще, часто пространство объектов разбивают на области и для каждой области настраивают свой алгоритм. Тогда такие области называют **областями компетентности алгоритмов**. При классификации объекта смотрят, в область компетентности какого алгоритма он попадает, и запускают соответствующий алгоритм. Есть методы одновременной настройки областей и алгоритмов (**ДЗ** *Попробуйте придумать их сами или посмотрите в* [Воронцов]). Часто области компетентности формируются из постановки задачи или строятся с помощью предварительной кластеризации объектов.

Библиографию и другую полезную информацию по бустингу можно посмотреть на сайте [Boosting].

§10.3. Решающие деревья

Поскольку очень популярно мнение, что самый лучший алгоритм классификации – бустинг над решающими деревьями, опишем в этой главе алгоритм решающих деревьев. Для простоты используем бинарные деревья. Без лишнего формализма покажем основную идею на примере. На рис. 10.3.1 изображено решающее дерево. Чтобы классифицировать объект x, вычисляем значение предиката $\varphi(x)$. Предположим, что это значение «истина», тогда спускаемся по дереву на следующий уровень по дуге помеченной 1 (если бы значение предиката было бы «ложь», то спуск проходил бы по дуге помеченной 0). Теперь вычисляем значение предиката $\varphi_1(x)$ (именно им помечена вершина, в которую мы пришли). Если его значение «ложь», то переходим на

уровень ниже по 0-дуге. В результате попадаем в вершину с пометкой класса 2. Поэтому наш объект x относим ко второму классу.

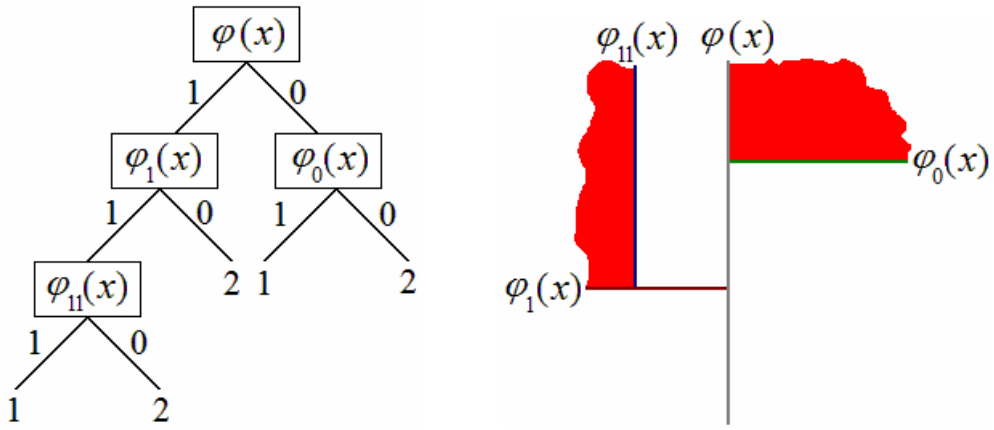


Рис. 10.3.1. Решающее дерево (слева) и разбиение пространства.

Таким образом, бинарное решающее дерево – это помеченное бинарное дерево с выделенной вершиной (корнем), внутренние вершины помечаются предикатами (точнее метками предикатов), а листья (вершины, которым инцидентно только одно ребро) помечаются метками классов. Весь процесс классификации с помощью решающих деревьев очень «прозрачен» и понятен. Его просто объяснить неспециалисту по теории классификации, поэтому решающие деревья часто используют в задачах, которые «приходят» из медицины, биологии, социологии и т.д. Здесь решающее дерево «превращается» в алгоритм для специалиста из другой области. Например, если в медицинской задаче $\varphi(x)$ = «температура выше 38 градусов», то для классификации надо первым делом измерить температуру, а дальше поступать так, как описано в соответствующем поддереве.

Если все предикаты в дереве имеют вид

$$\varphi(x) = \begin{cases} 1, & f_j(x) \ge c, \\ 0, & f_j(x) < c, \end{cases}$$

(т.е. сравнение с порогом значения какого-то признака), то дерево разбивает признаковое пространство на области классов примерно так, как показано на рис. 10.3.1. Нетрудно понять вид разделяющей поверхности для предикатов другого вида.

Ясно, что при построении дерева надо, чтобы область одного класса накрыла его представителей, а область другого класса – его представителей. Ниже представлен типичный алгоритм построения решающего дерева (типа ID3).

Рекурсивная функция `BuildTree`(U) построения дерева

Обращение к функции происходит с параметром U (текущий перечень объектов, в первом вызове – вся обучающая выборка)

1. Если все (почти все) объекты из U принадлежат одному классу, то создать лист с пометкой этого класса, возврат.
2. Найти предикат φ, $U_1 = \{x \in U \mid \varphi(x) = 1\}$, $U_0 = \{x \in U \mid \varphi(x) = 0\}$.
3. Если $U_0 = \varnothing$ или $U_1 = \varnothing$, то создать лист с пометкой класса, к которому принадлежит большинство объектов из U, возврат.
4. Создать вершину, помеченную предикатом φ, с потомками **`BuldTree`**(U_1) и **`BuldTree`**(U_0), возврат.

Во втором пункте происходит поиск предиката. Как правило, просто перебирают предикаты из некоторого конечного множества. В **методе случайных деревьев** перебирают предикаты вида

$$\varphi(x) = \begin{cases} 1, & \sum_{j=1}^{n} c_j f_j(x) \geq c, \\ 0, & \sum_{j=1}^{n} c_j f_j(x) < c, \end{cases}$$

где $c_1, \ldots, c_n, c$ – случайные коэффициенты. Несколько раз выбирают случайные коэффициенты и среди построенных предикатов выбирают тот, который максимизирует некоторый критерий.

Критериев для выборов предикатов существует достаточно много (см. [Воронцов]). Опишем один из них, наиболее часто используемый. Рассмотрим задачу с двумя классами: $Y = \{1,2\}$. Для множества объектов U естественным образом вводится понятие энтропия

$$H(U) = -p \log_2 p - (1-p) \log_2 (1-p),$$

где

$$p = \frac{|\{x \in U \mid y(x) = 1\}|}{|U|}$$

(считаем, что $0 \cdot \log_2 0 = 0$). Эта величина показывает, насколько во множестве U объекты «одноклассовы»: если все объекты принадлежат одному классу (неважно какому), то $H(U) = 0$, а если разным, то $H(U) > 0$. В частности, если представителей всех классов поровну, то $H(U) = 1$.

Интуитивно понятно, что при построении решающего дерева в каждой внутренней вершине надо разбивать выборку U на две подвыборки U_1 и U_2 так, чтобы энтропии $H(U_1)$, $H(U_2)$ были минимальны. Часто при выборе предиката используют следующий критерий **«Информационный выигрыш»**:

$$\mathrm{IGain}(\varphi, U) = H(U) - \left[\frac{|U_1|}{|U|} H(U_1) + \frac{|U_2|}{|U|} H(U_2) \right] \to \max_{\varphi},$$

$U_1 = \{x \in U \mid \varphi(x) = 1\}$, $U_2 = \{x \in U \mid \varphi(x) = 2\}$, который показывает уменьшение энтропии при выборе предиката φ.

ДЗ *Обобщите критерий на случай нескольких классов. Предложите свои критерии. Проведите эксперименты на модельных задачах. Попробуйте строить дерево «с заглядыванием вперёд»: оценивать уменьшение энтропии через два уровня* (*эта идея допускает несколько различных формализаций*).

При использовании решающих деревьев для бустинга предикаты выбирают достаточно простыми, а дерево строят до определённого уровня (считается, что «большие» деревья переобучены). Как правило, предикаты имеют вид

$$\varphi(x) = \begin{cases} 1, & g(x) \in G, \\ 0, & g(x) \notin G. \end{cases}$$

Дерево можно также «сокращать» после построения. Для этого используют выборку объектов с известной классификацией, которая не участвовала в обучении, и пытаются преобразовать дерево (например, заменить какое-то поддерево на поддерево одной из дочерних вершин) так, чтобы увеличить некоторый функционал, зависящий от текущего дерева и выборки (например число ошибок на выборке при классификации этим деревом). **ДЗ** *Проведите соответствующие эксперименты*.

Глава 11. КЛАСТЕРИЗАЦИЯ

§11.1. Задача кластеризации

В машинном обучении, кроме задач **обучения с учителем**, когда объекты $x^1,\dots,x^m$ заданы вместе с метками $y(x^1),\dots,y(x^m)$, есть задачи **обучения без учителя**, в которых заданы только объекты $x^1,\dots,x^m$ (без меток). Вот некоторые типы задач обучения без учителя:

1. **Кластеризация** (разбиение объектов на группы похожих).
2. **Обнаружение новизны** (нахождение объектов, не похожих на остальные).
3. **Поиск логических закономерностей**.
4. **Восстановление плотности**.
5. **Вложение объектов в пространство малой размерности** (с сохранением каких-то свойств).
6. **Выделение структуры** на множестве объектов (например, по изображению описать на формальном языке, что изображено).

«Центральной» задачей является **кластеризация**: разбиение объектов $x^1,\dots,x^m$ на группы похожих объектов – кластеры $K_1,\dots,K_l$ (число кластеров l может быть заранее не задано[1]). Формулировку задачи нельзя считать достаточно чёткой. Главная причина – непонятно, какое разбиение на кластеры правильное, а какое нет. Отметим, что это общее свойство задач обучения без учителя[2]… их ставят, когда хотят из данных «выудить какую-то информацию» (при этом, априорно не конкретизируя, что именно надо получить).
В частности, кластеризацию применяют для

1) сокращения объёма информации (кластер можно заменить одним его представителем, поскольку остальные объекты «на него очень похожи»),
2) выяснения структуры данных (например, описанные ниже иерархические методы позволяют разложить данные по «виртуальной системе каталогов»),
3) обнаружения новизны (хотя это и отдельная задача, но «особенные» объекты, как правило, являются представителями маленьких кластеров).

Отдельно отметим использование кластеризации для эффективного обучения с учителем: объекты контрольной выборки разбивают на кластеры и на каждом кластере обучают свой алгоритм классификации или регрессии (часто это даёт существенное улучшение качества классификации/регрессии на всей выборке), при классификации/регрессии сначала определяют, в какой кластер попадает объект, потом запускают соответствующий алгоритм.

---

[1] Используем символы K_j для обозначения кластеров. Раньше ими обозначали классы. Разница лишь в том, что кластеры заранее не заданы. Аналогично, l – число кластеров.

[2] В данном учебном пособии подробно не разбираются все задачи. Отметим, что восстановление плотности использовалось в байесовском классификаторе, вложение в пространство – в нейронных сетях, поиск закономерностей – в тестовых алгоритмах и алгоритмах с представительными наборами (представительный набор соответствует закономерности). Поиск новизны часто сводится к задаче кластеризации (см. дальше).

Будем считать, что объекты заданы в метрическом пространстве X с метрикой ρ. Иногда мы будем вычислять среднее арифметическое объектов. Если объекты задаются признаковым описанием, то понятно, как производить подобные вычисления. Если известны только попарные расстояния между объектами, то можно среднее арифметическое

$$\frac{1}{r}\sum_{i=1}^{r} z_r ,$$

заменить объектом

$$z_j = \arg\min_{z_j}(\max_i \rho(z_j, z_i))$$

(«центр» в метрическом пространстве).

Начнём с описания «самого популярного» алгоритма:

Алгоритм *k*-средних

0. Инициализация.

Случайно генерируем центры кластеров $z^1,\ldots,z^l$ в пространстве X.

1. Приписывание.

Относим объект к тому кластеру, к центру которого он ближе:

$$K_j = \{x^t \mid \rho(x^t, z^j) = \min_i \rho(x^t, z^i)\}$$

(если минимум достигается на нескольких центрах, то относим к одному из подходящих кластеров).

2. Адаптация.

Пересчитываем центры (чтобы они действительно стали «центрами»):

$$z^j = \frac{1}{|K_j|}\sum_{x^t \in K_j} x^t ,\ j \in \{1,2,\ldots,l\}$$

(если какой-то кластер пустой, то он либо совсем удаляется, либо центр остаётся на месте).

3. Если кластеризация не меняется две итерации подряд (не меняют положения центры кластеров), тогда алгоритм завершает работу, иначе переход к п. 1.

Термин «***k*-средних**» устоявшийся. В наших обозначениях алгоритм правильнее назвать «***l*-средних**» (хотя всё равно число центров кластеров может уменьшиться в результате работы алгоритма).

Замечание. В качестве центров кластеров можно выбрать случайные элементы выборки. Тогда на первой итерации все кластеры будут непустыми. **ДЗ** *Будут ли они непустыми на следующих итерациях? Обоснуйте ответ теоретически. Проведите эксперименты на ЭВМ.*

Есть «мягкая» форма алгоритма, в которой на первом этапе идёт не «жёсткое» приписывание объектов кластерам, а вычисление «степени принадлежности»:

$$p_j(x^t) = \frac{e^{-\beta \cdot \rho(x^t, z^j)}}{\sum_{i=1}^{l} e^{-\beta \cdot \rho(x^t, z^i)}},$$

а на втором – центры вычисляются по формуле

$$z^j = \frac{\sum_{t=1}^{m} p_j(x^t) \cdot x^t}{\sum_{t=1}^{m} p_j(x^t)}.$$

Кстати, в качестве ответа можно выдавать не только «чёткое разбиение на кластеры», а число кластеров и степень принадлежности к каждому из них. Подобная задача называется **нечёткой кластеризацией**. Её можно решать также EM-алгоритмом, если известно, что каждый кластер представляет реализацию случайной величины с распределением известного вида.

У алгоритма k-средних есть недостатки, которые присущи многим другим алгоритмам кластеризации:
1) результат не определён однозначно (многое зависит от начальной позиции центров),
2) необходимость выбора параметра l[3],

Ниже представлена простая реализация метода в виде MatLab-кода и результат её выполнения (см. рис. 11.1.1, 11.1.2).

Метод k-средних

```
% выборка
X = [randn([5 2]); randn([5 2]) + 1];
m = size(X,1); % число точек в выборке

k = 3; % число кластеров

% сгенерировать центры (k случ. точек выборки)
r = randperm(m);
centers = X(r(1:k),:);

olderror = 0;
newerror = Inf;
% пока ошибка не стабилизируется
while (olderror~=newerror)
    olderror = newerror;
```

[3] Во многих случаях необходимо ещё выбрать и метрику ρ!

```
    % матрица попарных расстояний
    D = sqrt(sum(abs(repmat(permute(X, [1 3 2]), ...
        [1 size(centers,1) 1]) - ...
        repmat(permute(centers, [3 1 2]), [m 1 1]) ).^2, 3));
    % ближайшие центры
    [D I] = min(D,[],2);

    % суммарная ошибка приближения центрами
    newerror = sum(sqrt(sum((X - centers(I,:)).^2,2)))

    % визуализация
    clf;
    scatter(X(:,1), X(:,2), 30, 'filled' ,'k');
    hold on;
    scatter(centers(:,1), centers(:,2), 50);
    fordraw = reshape([X centers(I,:) nan(size(X))]', [2 m*3])';
    plot (fordraw(:,1), fordraw(:,2))

    %пауза
    pause

    % пересчёт центров
    formult = sparse(1:m,I,1,m,k);
    centers = diag(1./sum(formult))*(formult'*X);
end;
% ответ в I
```

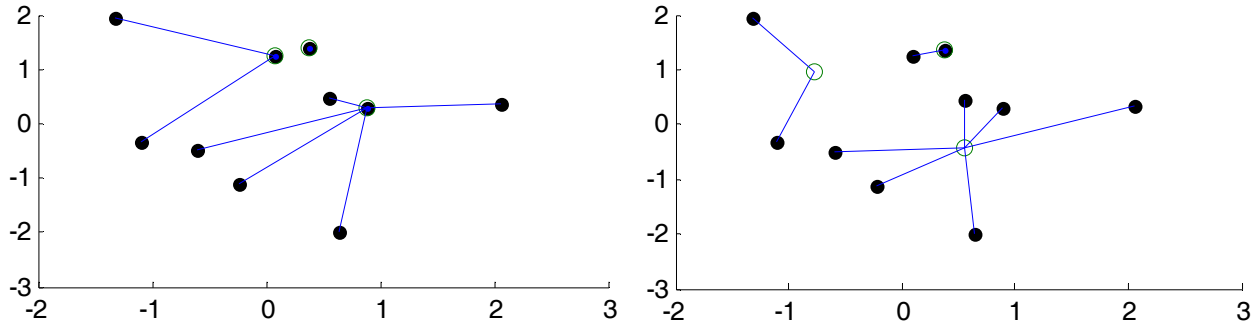


Рис. 11.1.1. Начальные положения центров (слева) и результат пересчёта (справа).

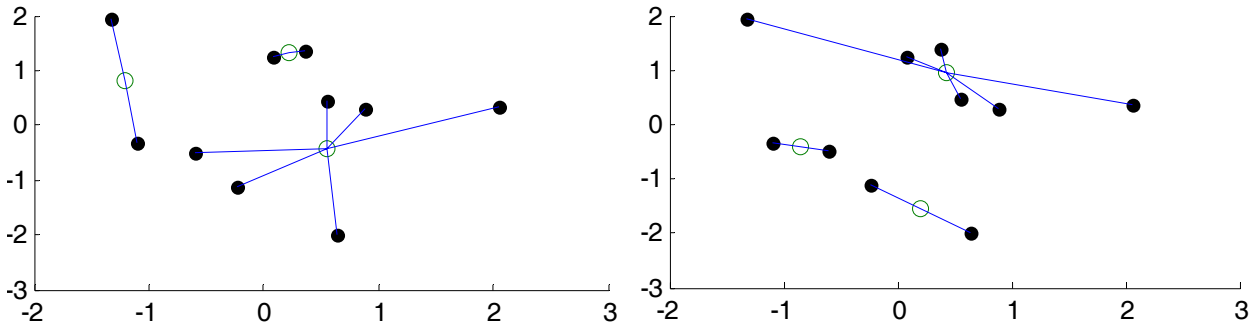


Рис.11.1.2. Две разные кластеризации (при различных начальных положениях центров).

Самым простым алгоритмом кластеризации считается следующий, который часто называют «основным» или «ближайшим соседом» (для кластеризации).

Ближайший сосед (алгоритм кластеризации)

0. Инициализация.

Пусть первый объект принадлежит первому кластеру. Считаем, что он пока его образует: $K_1 = \{x^1\}$, а больше кластеров нет: $l := 1$.

1. Цикл по всем (оставшимся) объектам:

$t = 2,3,\ldots,m$.

1.1. Находим ближайший кластер к объекту x^t:

$j : d(x^t, K_j) = \min\limits_i d(x^t, K_i)$.

1.2. Если он далеко от объекта ($d(x^t, K_j) > D$) и кластеров пока мало ($l < L$), то образуем новый кластер:

$l := l + 1$, $K_l = \{x^t\}$.

1.3. Иначе присоединяем объект к кластеру:

$K_j = K_j \cup \{x^t\}$.

1.4. Адаптация (пересчёт некоторых параметров, см. ниже).

Отметим сначала, что алгоритм имеет параметры L (оценка числа кластеров сверху) и D (формализует, что такое «далеко»). Отметим также, что в алгоритме вычисляется некоторая величина $d(x^t, K_j)$, которая формализует близость объекта и множества. Есть много способов её вычисления:

1. Расстояние до ближайшего элемента множества:

$$d(x^t, K) = \min_{z \in K} \rho(x^t, z).$$

2. Расстояние до дальнего элемента множества:

$$d(x^t, K) = \max_{z \in K} \rho(x^t, z).$$

3. Расстояние до центра множества:

$$d(x^t, K) = \rho(x^t, \frac{1}{|K|} \sum_{z \in K} z).$$

4. Среднее расстояние:

$$d(x^t, K) = \frac{1}{|K|} \sum_{z \in K} \rho(x^t, z)$$

(или медианное и т.д.).

Возможны различные их комбинации. В случае вычисления расстояния до центра необходим шаг 1.4 алгоритма, чтобы пересчитывать центр.

Замечание. Результат «ближайшего соседа» сильно зависит от порядка предъявления объектов.

Приведём ещё один алгоритм, который похож на *k*-средних:

Алгоритм ФорЭль

0. Инициализация.
Пусть $l := 1$ (пока один кластер).
1. Синтез нового кластера.
Случайно выбираем точку z^l пространства X (центр нового кластера).
2. Адаптация
Пусть $K_l = \{x^t \mid \rho(x^t, z^l) \le R\}$ (если кластер пустой, то переходим к п.1).
Пересчитываем центр

$$z^l = \frac{1}{|K_l|} \sum_{x^t \in K_l} x^t .$$

Если центр меняет своё положение при пересчёте, то переходим к п. 2.
3. Если остались объекты, не попавшие в кластеры, то
$l := l + 1$,
переход к п. 1.

В этом алгоритме есть «неудобный» параметр R, геометрический смысл которого ясен: это радиус (любого!) кластера. Каждый кластер представляет собой шар (точнее, точки, попавшие внутрь шара). Чтобы в п.2 не получались пустые кластеры, можно в качестве новой точки брать элемент выборки, который пока не вошёл ни в один кластер.

§11.2. Иерархическая кластеризация

Термин «**разбиение (объектов $x^1,\ldots,x^m$) на кластеры**» $\{K_1,\ldots,K_l\}$ формализуется условиями:

$$K_1 \cup \ldots \cup K_l = \{x^1,\ldots,x^m\},$$

$$|K_1| + \ldots + |K_l| = |\{x^1,\ldots,x^m\}| \text{ (непересечение кластеров)},$$

$$\forall j \in \{1,2,\ldots,l\} \quad K_j \neq \varnothing .$$

Говорят, что одно **разбиение вложено в другое**, если любой кластер первого является подмножеством какого-то кластера второго. Например, разбиение $\{\{x^1,x^2\},\{x^3,x^4\},\{x^5,x^6\}\}$ вложено в разбиение $\{\{x^1,x^2\},\{x^3,x^4,x^5,x^6\}\}$. Для удобства, в разбиении будем указывать только номера объектов (в примере выше – $\{\{1,2\},\{3,4\},\{5,6\}\}$ и $\{\{1,2,3,4\},\{5,6\}\}$). Алгоритмы иерархической кластеризации строят последовательность вложенных разбиений на кластеры. Часто это очень удобно: получаем последовательность кластеризаций, которые не противоречат друг другу, а уточняют друг друга, поскольку «моделируют» вложение объектов в «виртуальные каталоги». При желании можно выбрать кластеризацию с нужным числом кластеров. Часто нужно получить именно иерархическую кластеризацию (например при автоматической разбивке группы новостей по каталогам).

Пороговый граф $\Gamma(D)$ **с порогом** D, $D \in \mathbf{R}$, состоит из вершин $1,2,\ldots,m$, причём вершины i, j соединены ребром тогда и только тогда, когда $\rho(x^i, x^j) \le D$. Меняя D от нуля до $\max\limits_{i,j} \rho(x^i, x^j)$, получаем различные графы, начиная с графа изолированных вершин, заканчивая полным графом. В подходе **Single Link** (ближайший сосед) кластер соответствует связной компоненте порогового графа. В подходе **Complete Link** (дальний сосед) – полной компоненте.

Пример. Рассмотрим расположение объектов, показанное на рис. 11.2.1.

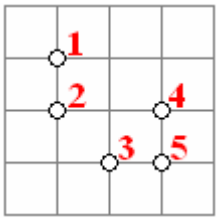
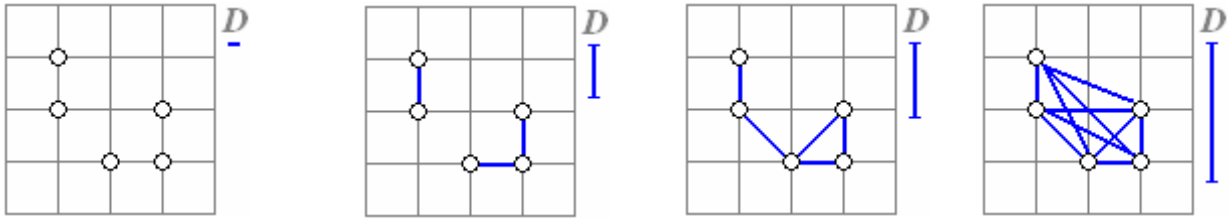


Рис. 11.2.1. Исходное положение точек.

Будем последовательно менять значение D, изображая граф $\Gamma(D)$.



`{{1},{2},{3},{4},{5}}` `{{1,2},{3,4,5}}` `{{1,2,3,4,5}}`

`{{1},{2},{3},{4},{5}}` `{{1,2},{3},{4,5}}` `{{1,2},{3,4,5}}` `{{1,2,3,4,5}}`

Рис. 11.2.2. Пороговый граф и разбиения методом Single Link (первая строка) и Complete Link (вторая).

На рис. 11.2.2 показаны различные пороговые графы для евклидовой метрики с указанием разбиений для метода Single Link и Complete Link. Заметим, что разбиение методом Complete Link неоднозначно: для второго графа можно выбрать `{{1,2},{3,4},{5}}`, а можно выбрать `{{1,2},{3},{4,5}}`. Такой эффект называется **ничьёй**. На практике выбирают любой вариант (ничьи в реальной ситуации встречаются достаточно редко). Третий граф связный, поэтому при увеличении порога метод Single Link не меняет разбиения. В нём появляется новый полный подграф: к ребру `{4,5}` добавляется вершина `3`. При дальнейшем увеличении порога меняется граф, но разбиение остаётся прежним, пока не получится полный граф.

Иерархические разбиения принято показывать на дендрограммах. Что это такое, понятно из рис.11.2.3. Показано, как последовательно объекты группируются в кластеры в нашем примере. По вертикали откладывается величина порога (при котором произошло соответствующее объединение).

Рис. 11.2.3. Дендрограммы метода Single Link (слева) и Complete Link (справа).

Описанные алгоритмы можно представить в другом виде:

Общая процедура слияния на основе порога

0. Инициализация.

Пусть исходное разбиение $R = \{\{x^1\},\{x^2\},\ldots,\{x^m\}\}$.

1. Увеличиваем порог D в цикле.

1.1. Слияние.

Если для каких-то кластеров K_i, K_j текущего разбиения выполняется $d(K_i, K_j) \le D$, то объединяем их в один кластер.

Часто удаётся «грамотно» варьировать порог D (в случае разобранных выше алгоритмов можно пробегать упорядоченное множество попарных расстояний между объектами). Кроме того, п.1.1 часто упрощают (объединяют два «самых близких» кластера):

Общая процедура слияния

0. Инициализация.

Пусть исходное разбиение $R = \{\{x^1\},\{x^2\},\ldots,\{x^m\}\}$.

1. Пока не получим нужное число кластеров (или не завершим построение всей дендрограммы)

1.1. Слияние.

Найти пару ближайших кластеров K_i, K_j:

$$(i, j) = \arg\min_{(i,j)} d(K_i, K_j)$$

и объединить их:

$$R := R \setminus \{K_i, K_j\} \cup \{K_i \cup K_j\}.$$

Расстояние d между кластерами опять же можно выбирать по-разному:

1. Расстояние между ближайшими точками (Single Link):

$$d(A, B) = \min_{\substack{a \in A \\ b \in B}} \rho(a, b).$$

2. Расстояние между дальними точками (Complete Link):

$$d(A,B) = \max_{\substack{a\in A\\ b\in B}} \rho(a,b).$$

3. Расстояние между центрами или какими-то выделенными точками:

$$d(A,B) = \rho\left(\frac{1}{|A|}\sum_{a\in A} a, \frac{1}{|B|}\sum_{b\in B} b\right).$$

4. Среднее арифметическое (или медиана и т.д.) попарных расстояний между представителями разных классов:

$$d(A,B) = \frac{1}{|A|\cdot|B|}\sum_{\substack{a\in A,\\ b\in B}} \rho(a,b).$$

Пример «выделенных точек»: в одноточечном кластере это точка, образующая кластер, а при слиянии двух кластеров это среднее арифметическое выделенных точек двух кластеров (отличается от центра).

Общая процедура слияния реализует стратегию «**снизу-вверх**». Есть ещё стратегия «**сверху-вниз**»: стартуем от разбиения из одного кластера, в который входят все объекты, а затем последовательно расщепляем его на подкластеры.

Ещё один графовый алгоритм кластеризации основан на построении **минимального остовного дерева** – дерева минимальной длины, которое соединяет все точки (длина ребра, соединяющего вершины i, j равна $\rho(x^j, x^i)$, длина дерева – сумма длин его рёбер).

Алгоритм построения минимального остовного дерева

1. В графе с изолированными вершинами $1,2,\ldots,n$ найти пару вершин с минимальным расстоянием $\rho(x^j, x^i)$ и соединить ребром (теперь эти вершины неизолированные).
2. Пока есть изолированные вершины

2.1. Соединить ребром изолированную вершину i и неизолированную вершину j, на которых достигается минимум

$$\min \rho(x^j, x^i)$$

(i пробегает все изолированные вершины, а j – все неизолированные).

После построения такого графа из него удаляют $l-1$ ребро (удаляют самые длинные рёбра). Оставшиеся l компонент связности объявляют кластерами. Обратите внимание, что здесь есть возможность получать заданное число кластеров!

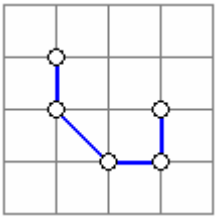
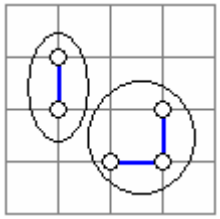



Рис.11.2.4. Минимальное остовное дерево и два кластера, образованные после удаления самого длинного ребра.

Описанный алгоритм можно успешно применять вместе с другими алгоритмами кластеризации. Например, сначала провести кластеризацию ФорЭлью с небольшим радиусом. Потом центры кластеров (теперь их рассматриваем как объекты) кластеризовать описанным алгоритмом. В результате получаем кластеризацию с заданным наперёд числом кластеров, имеющих понятную структуру. Каждый кластер – совокупность рядом расположенных шаров.

В принципе, задачу кластеризации можно формализовать следующим образом: расстояние $d(K_i, K_j)$ между различными кластерами должно быть большое, а между объектами одного кластера маленькое, поэтому необходимо минимизировать функционал вида

$$\frac{\sum_{i=1}^{l} \sum_{x^t, x^j \in K_i} \rho(x^t, x^j)}{\sum_{1 \le i < j \le l} d(K_i, K_j)}$$

по всевозможным разбиениям. Получаем задачу оптимизации (для «огромного переборного пространства», поскольку число различных разбиений огромно). Проблемы с выбором числа кластеров остаются, но их можно решить, «грамотно выбрав» функционал. Это является отдельной проблемой, поскольку формализации идеи «объекты в кластере близки, а сами кластеры далеки друг от друга» бывают разные. Несмотря на это, подобные функционалы полезны для выбора одной кластеризации среди множества[4]. Они могут служить дополнительным критерием отбора.

§11.3. Кластеризация с помощью нейронных сетей

Рассмотрим однослойную нейросеть (на рис. 11.3.1 изображена такая нейронная сеть с тремя нейронами в одном слое). Будем осуществлять кластеризацию следующим образом. Пронумеруем нейроны в выходном слое от 1 до l. Подаём объект на вход нейронной сети и относим его в кластер с номером выходного нейрона, на котором наблюдается максимальный сигнал. Такой нейрон называется **нейрон-победитель**.

[4] Например, среди множества кластеризаций, построенных алгоритмом *k*-средних при различных начальных положениях центров.

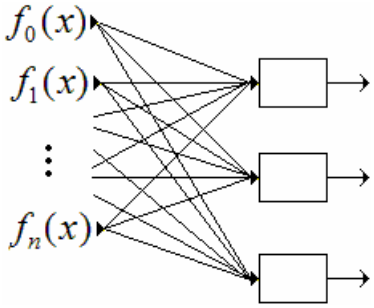


Рис. 11.3.1. Пример кластеризующей сети.

Настройка сети производится обычным градиентным алгоритмом. Прогоняем через нейронную сеть всю выборку. После подачи очередного объекта $\tilde{x}$ веса нейрона-победителя (веса дуг, которые в него ведут) корректируются по формуле

$$\tilde{w} = \tilde{w} + \eta(\tilde{x} - \tilde{w})$$

(η – темп обучения). В контексте классифицирующих нейронных сетей такое правило называют «**победитель получает всё**». Правило имеет понятный геометрический смысл. Можно считать, что вектор весов входов каждого нейрона это центр соответствующего кластера. «Центр» каждый раз сдвигается в сторону очередного своего представителя.

В **самоорганизующихся картах Кохонена** выходные нейроны располагают в узлах прямоугольной сетки (как показано на рис. 11.3.2). Каждому нейрону однозначно сопоставляется пара из множества $\{1,2,\ldots,l_1\}\times\{1,2,\ldots,l_2\}$ (координаты сетки). При этом между любыми двумя нейронами можно измерить расстояние (как расстояние между соответствующими парами, например в l_1-метрике). Нейрон, которому соответствует пара (i,j) будем называть ij-м нейроном, а вектор весов его входов обозначать через $\tilde{w}_{ij}$.

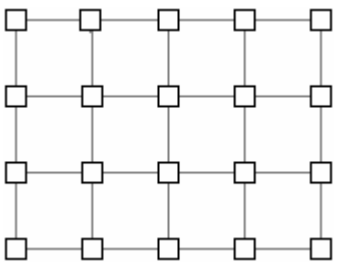


Рис. 11.3.2. Нейроны карты Кохонена.

Если победил ab-й нейрон, то веса всех нейронов корректируются по правилу

$$\tilde{w}_{ij} = \tilde{w}_{ij} + \eta \cdot h((a,b),(i,j))(\tilde{x} - \tilde{w}_{ij}),$$

где $h((a,b),(i,j))$ – функция, принимающая максимальное значение при $(a,b)=(i,j)$, например

$$\exp(-\alpha(|a-i|+|b-j|)),$$

где α – параметр. Таким образом, здесь не только нейрон-победитель «старается быть похожим» на объект, но и его соседи… чем дальше расположен на сетке нейрон от нейрона-победителя, тем меньше меняется его вес.

Обучение сети продолжается, пока не стабилизируются её ответы.

В заключение обратим внимание, что разные методы кластеризации используют разную «дополнительную информацию»: число кластеров или оценка этого числа, порог близости (формализация, что такое «близко»). Ниже в таблицах показана специфика методов.

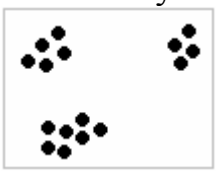
<table>
<tr><th colspan="6">Методы кластеризации</th></tr>
<tr><td colspan="3" rowspan="2">Иерархические</td><td colspan="3">Основанные на одном разбиении</td></tr>
<tr><td rowspan="2">Графовые</td><td colspan="2">Итерационные</td><td rowspan="3">Ближайший сосед</td></tr>
<tr><td>Коррекция центра</td><td>Статисти-ческие</td></tr>
<tr><td>Общая процедура слияния / разделения</td><td>Single Link, Complete Link</td><td>Мин. остовное дерево</td><td>k-средних, ФорЭль, Нейросетевые</td><td>EM</td></tr>
</table>

Алгоритм	**Параметры алгоритма**	
	Число кластеров	**Порог близости**
k-средних	+	
Ближайший сосед	+	+
ФорЭль	При использовании с минимальным остовным деревом	+
Single Link, Complete Link	Для получения конкретных кластеризаций	
Общая процедура слияния / разделения	Для получения конкретных кластеризаций	
Минимальное остовное дерево	Могут использоваться для получения кластеризации, но можно обойтись и без параметров.	

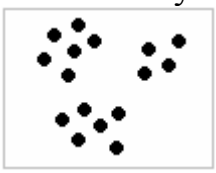
ДЗ *Предложите эвристику выбора числа кластеров. Обратите внимание, что в графовых алгоритмах она «вполне естественна».*

При тестировании алгоритмов кластеризации лучше использовать различные типы модельных задач, поскольку некоторые алгоритмы хороши именно для задач конкретных типов. Ниже приведём примеры часто используемых модельных задач.

Плотные сгустки

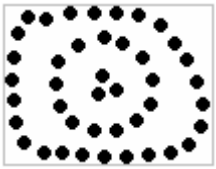


Неплотные сгустки

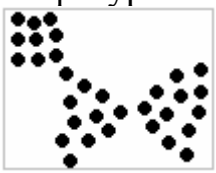


Ленты

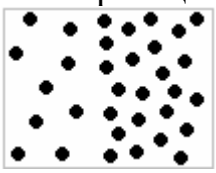
Кольца



Соприкасающиеся фигуры



Нетрадиционная кластеризация



Для нетрадиционной кластеризации (на рис. приведена кластеризация по плотности) необходимо «изобретение» новых алгоритмов (ясно, что все, рассмотренные выше, кластеризуют, основываясь на близости объектов).

Глава 12. «ШАМАНСТВО» В АНАЛИЗЕ ДАННЫХ

Чтобы начать решать задачу, можно иметь под рукой только систему визуализации и простой обработки данных. Идеально подходит для этого среда MatLab (даже без дополнительных библиотек). Продемонстрируем, как в этой среде можно решать реальные задачи анализа данных методом «пристального вглядывания».

§12.1. Классификация сигналов головного мозга

Пусть в матрице **X** размера 200×3000 (хотя размеры не так уж и принципиальны) построчно записаны описания сигналов, а в векторе **Y** (матрице размера 200×1) – их классификации на два непересекающихся класса: значения +1 и –1. Сразу отметим, что в этом параграфе все иллюстрации (и размеры матриц) приведены для реальных данных, которые были предложены участникам международного конкурса по классификации сигналов «BCI competition III» 2003 г. (данные Data set I [Tübingen], [BCI 3]). В задаче конкурса каждый сигнал отражал 3-секундную активность головного мозга во время некоторого ментального действия и снимался с частотой 1000Гц с помощью ECoG-электродной сетки размера 8x8 (64-мерный 3000-точечный сигнал). Основная специфика предложенной задачи (о ней мы ещё поговорим) заключалась в том, что обучающая (278 сигналов) и контрольная (100 сигналов) выборки были сформированы в различные дни (с интервалом в одну неделю). Мы немного упростим задачу и рассмотрим сигналы только с одного электрода, поэтому матрица имеет размеры не 200×64×3000[1], а 200×3000.

При визуализации сигналов не видно отличия представителей первого и второго классов, см. рис. 12.1.1 (хотя увидеть, что, например, сигналы одного класса имеют больше локальных максимумов, чем сигналы другого класса крайне сложно). Но самое интересное, что сигналы контрольной выборки совсем не похожи на сигналы обучающей выборки (их значения почти везде намного больше, разбросы сигналов также больше и т.д., см. рис. 12.1.1).

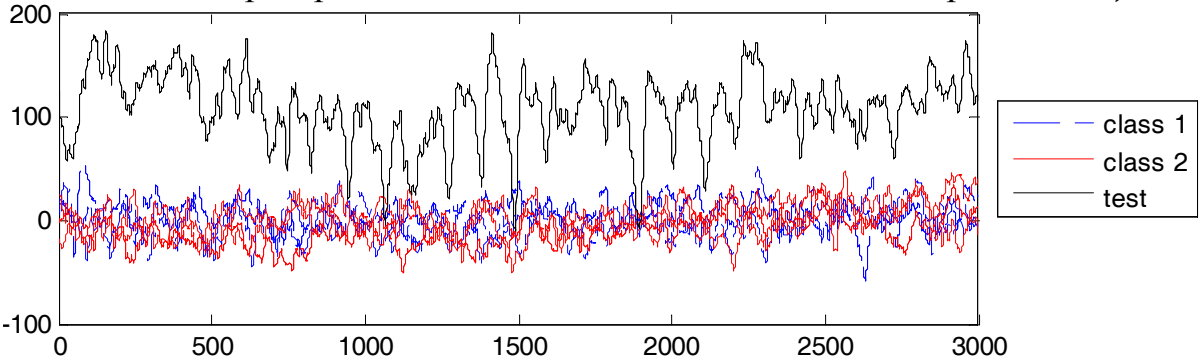


Рис. 12.1.1. Визуализация ECoG-сигналов головного мозга.

[1] Обратите внимание на размеры данных в реальных задачах!

Чтобы «смотреть» на сигналы, напишем следующую функцию[1] для визуализации данных в двумерном признаковом пространстве (подобные функции очень полезно иметь под рукой).

Функция визуализации

```
function ShowFs(f1,f2,Y)
% f1, f2 - признаки (векторы),
% Y - вектор (той же размерности) классификации
clf;
scatter(f1(Y>0), f2(Y>0), 40, '*', 'r'); % один класс
hold on;
scatter(f1(Y<0), f2(Y<0), 20, 's', 'filled', 'b'); % второй класс
hold off;
```

Простое обращение к этой функции **`ShowFs(mean(X'),std(X'),Y)`** даёт следующую картину, см. рис. 12.1.2.

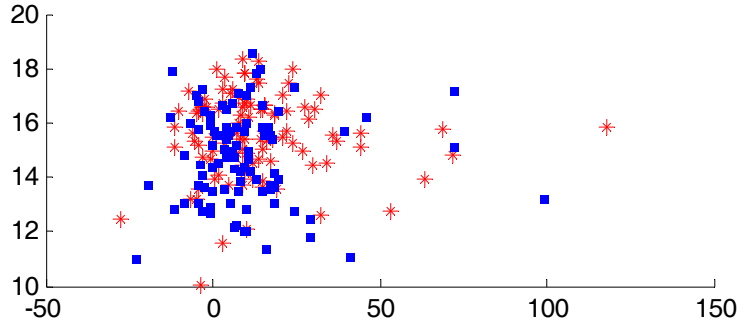


Рис. 12.1.2. Результат `ShowFs(mean(X'),std(X'),Y)`.

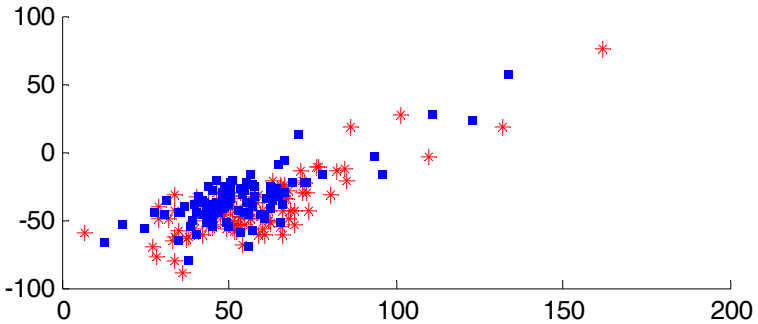


Рис. 12.1.3. Результат `ShowFs(max(X'),min(X'),Y)`.

По сути, мы посмотрели, как выглядят наши данные в пространстве двух признаков: **`mean(X')`** – среднее значение сигнала, **`std(X')`** – разброс значений сигнала (выборочная дисперсия). Используя стандартные функции MatLaba, таких признаков можно придумать «целую кучу»… На рис. 12.1.3 показаны объекты в пространстве

(максимальное значение сигнала, минимальное значение сигнала).

[1] Можно обойтись стандартной функцией **`scatter`**. Для данного учебного пособия её пришлось «усовершенствовать», чтобы при распечатке в градациях серого сохранялась наглядность.

Видно, что эти признаки коррелируют в бОльшей степени, чем предыдущая пара. Ведь зависимость признаков означает, что вместо облака точек мы видим «достаточно узкую» кривую, а линейная корреляция – прямую[1]. Подобную «сильно размытую» прямую мы видим на рис. 12.1.4. Естественно, почти никогда на практике не будет идеальной ровной прямой, которая получилась бы в модельной задаче, если мы первый признак сгенерировали случайно, а второй получили из него линейным преобразованием.

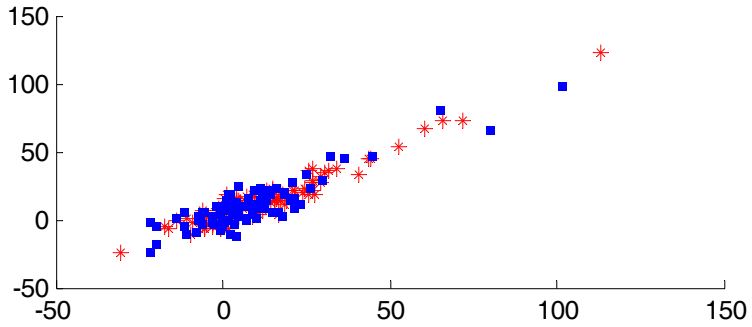


Рис. 12.1.4. Результат `ShowFs(mean(X(:,1:1500)'), mean(X(:,1501:end)'),Y).`

Однако даже это рисунок позволяет сказать нам, что сигналы, наверное, однородны. В том смысле, что их статистические свойства не зависят от времени отсчёта. По крайне мере, средние значения каждого сигнала выборки в первые 1.5 секунды и в последние не сильно отличаются.

Подобные наблюдения за различными признаковыми пространствами позволяют лучше понять природу сигналов. Как автоматизировать анализ сигналов, мы покажем дальше, но начинающему исследователю лучше попытаться найти хорошее признаковое пространство вручную.

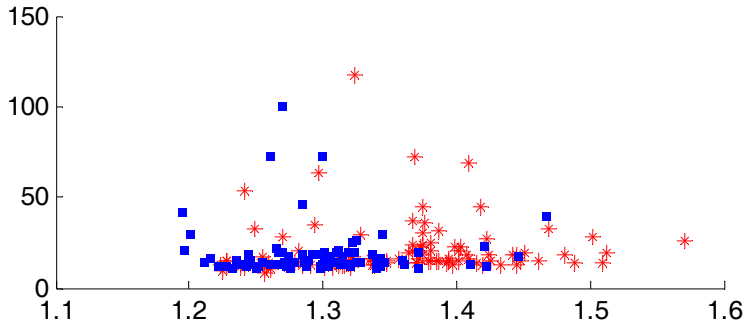


Рис. 12.1.5. Результат `ShowFs(mean(abs(diff(X'))),mean(abs(X')),Y).`

Используя суперпозиции не более чем из трёх примитивных функций MatLaba, можно получить рис. 12.1.5, на котором уже видна неплохая

[1] Естественно, только некоторые её точки.

разделимость классов. Итак, мы «нащупали» интересный признак: **`mean(abs(diff(X')))`**, переводящий сигнал $(u_1,\ldots,u_n)$ в значение

$$\frac{1}{n-1}\sum_{i=1}^{n-1}|u_{i+1}-u_i|,$$

«физический смысл» которого – скорость изменения сигнала. На самом деле, различные эксперименты по генерации эвристических признаков в этой задаче приводят к выводу, что лучше работают эвристики, которые оценивают интенсивность скачков сигнала.

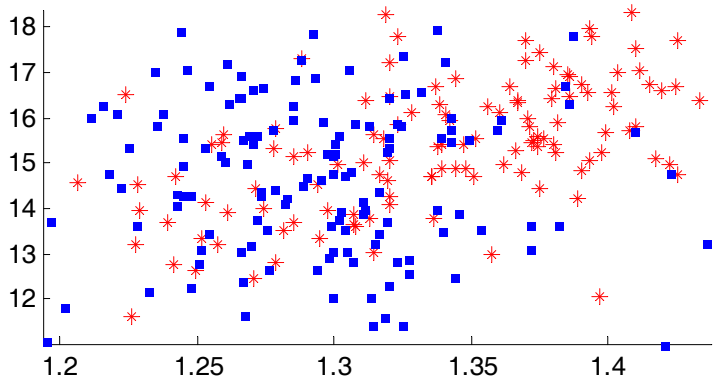


Рис. 12.1.6. Фрагмент результата `ShowFs(mean(abs(diff(X'))), std(X'),Y)`.

Одно из признаковых пространств, в котором каждый признак показывает «скорость изменения сигнала», приведено на рис. 12.1.6. Это очень типичная картина! Признаки не коррелируют, но точки располагаются на плоскости не хаотично, а вдоль определённых линий. Глядя на такую картинку, можно сделать много гипотез о природе данных. Например, что точки расположенные последовательно вдоль одной линии соответствуют данным, снятым в течение одного отрезка времени. А может, совсем наоборот, подобные расположения зависят не от времени, а от типа ментального действия (каждый класс имеет несколько типов).

Покажем ещё один стандартный приём, применяемый при визуальном анализе данных: «чуть-чуть» изменить найденный признак. Например, вместо модуля использовать квадрат:

$$\frac{1}{n-1}\sum_{i=1}^{n-1}(u_{i+1}-u_i)^2.$$

Новый признак, как правило, сильно коррелирует с исходным, но в проекции на эти два признака можно увидеть интересные закономерности, см. рис. 12.1.7.

На рис. 12.1.7 виден небольшой зазор между объектами первого и второго классов[1]. Самое интересное, что контрольная выборка тоже распадается на две группы с зазором, хотя лежит в стороне от обучающей[2] (в этом и состоит специфика нашей задачи: из-за разных условий формирования выборок они лежат в разных частях признакового пространства). Отсюда уже ясно, что надо использовать такое «распадение» контроля для его классификации.

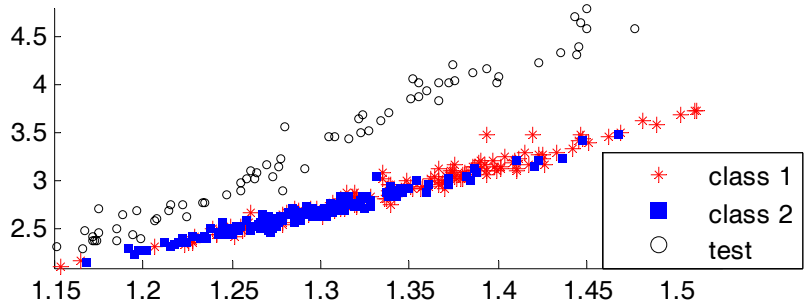


Рис.12.1.7. Фрагмент проекции обучения и контроля на пространство (mean(abs(diff(X'))), mean((diff(X')).^2)).

При решении задачи (участвуя в соревновании в 2003 году) пришлось существенно улучшить найденный признак. Хотя классификация проводилась пороговым правилом лишь по единственному признаку, сам признак уже был не совсем тривиальный. Листинг для его вычисления приведён ниже. **ДЗ** *Попробуйте разобраться, как «работает» такой признак?* Отметим лишь, что сначала происходит сглаживание сигнала (для устранения шумов), а затем вычисляется «обобщённая скорость изменения сигнала» (уже не по соседним точкам, а по целой окрестности).

```
% хороший признак для задачи BCI-3
function m = makef(X, Lusr, Lok)
    % X - матрица сигналов (по строкам)
    % Lusr - радиус окрестности для сглаживания
    % Lok - диаметр окрестности для обобщ. Производной

    L3000 = size(X,2); % 3000 точек
    X = X';

    %% сначала усредняем по окрестности радиуса Lusr
    X2 = X; % новое описание сигналов
    for u = 1:Lusr % попробуйте улучшить этот фрагмент кода
        X2 = X2 + circshift(X,u) + circshift(X,-u);
    end;
```

[1] Точнее между двумя «облаками точек». В первом облаке преобладают точки первого класса, а во втором – второго.

[2] На рис. 12.1.7 точки контрольной выборки немного смещены (параллельным переносом) в сторону точек обучающей выборки. На самом деле, контрольная выборка лежит «совсем в стороне».

```
    X2 = X2./(2*Lusr+1); % просто нормировка
    X2 = X2(((Lusr+1):(L3000-Lusr)),:); % устранение кр.эффектов

    %% нахождение min и max значений в каждой окрестности
    %% диаметра Lok
    Xmax = X2;
    Xmin = X2;
    for u = 1:Lok
        TMP = circshift(X2,-u);
        Xmax = max (Xmax, TMP);
        Xmin = min (Xmin, TMP);
    end;

    %% Скачки функции в окрестностях
    X = Xmax(1:(end-Lok),:) - Xmin(1:(end-Lok),:);

    %% Усреднение всех скачков
    m = (mean (X))';
end
```

§12.2. Классификация сигналов работы механизма

Рассмотрим теперь задачу классификации сигналов другой природы: используем данные Ford_A с международного соревнования «Ford Classification Challenge» 2008 г. [Ford]. Здесь матрица **X** имеет уже размеры 3271×500, т.е. в обучающей выборке 3271 сигнал, причём каждый задаётся 500 измерениями (хотя размеры матрицы **X** по-прежнему непринципиальны). Несколько сигналов обучающей выборки изображено на рис. 12.2.1.

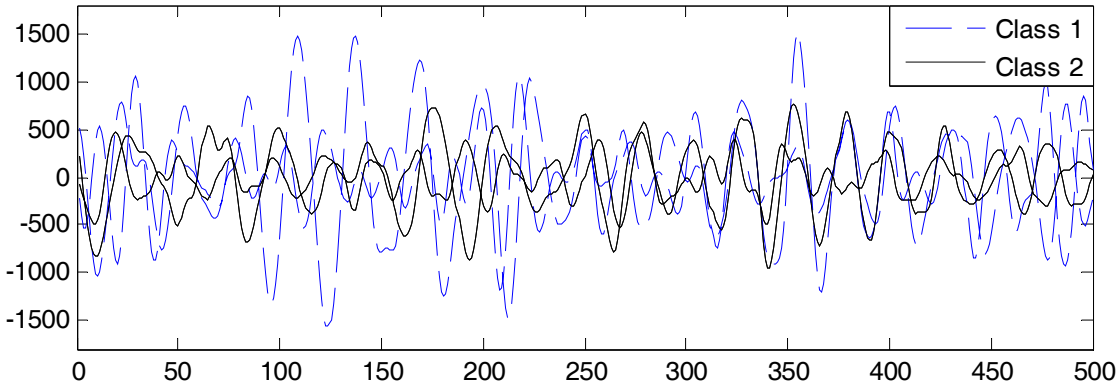


Рис. 12.2.1. Сигналы задачи [Ford].

Интересно, что в этой задаче сигналы уже «существенно неоднороны»: среднее значение второй половины сигнала не зависит от среднего значения первой половины, см. рис. 12.2.2.

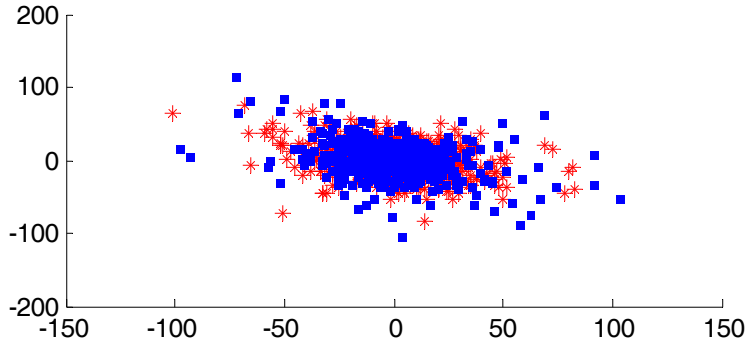


Рис. 12.2.2. Результат `ShowFs(mean(X(:,1:250)'), mean(X(:,251:end)'),Y)`.

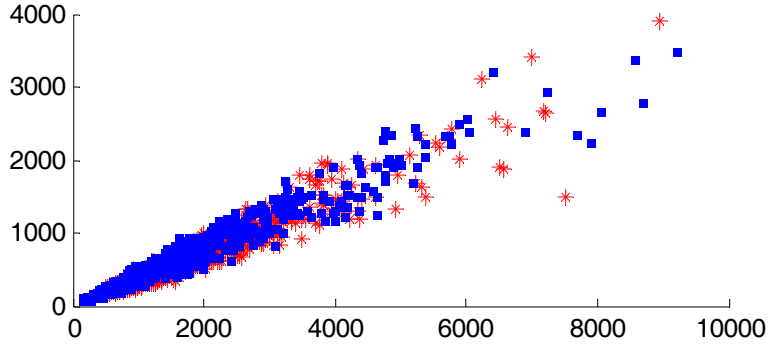


Рис. 12.2.3. Результат `ShowFs(max(X'),std(X'),Y)`.

Интересна также зависимость (и даже корреляция) между выборочной дисперсией и максимальным значением сигнала, см. рис. 12.2.3.

Хотя более явно коррелируют максимальные и минимальные значения, см. рис. 12.2.4 (что, кстати, бывает достаточно часто на реальных данных).

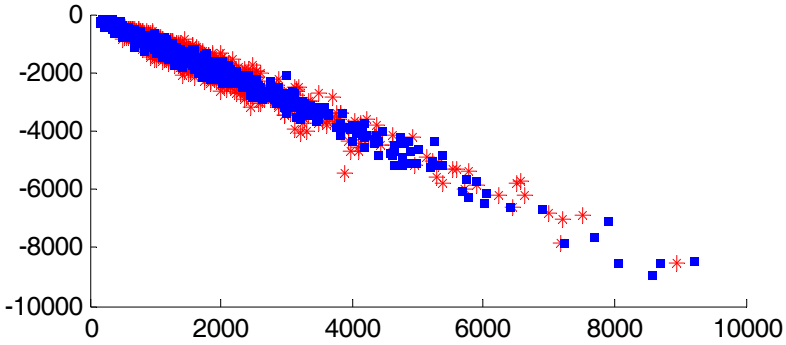


Рис. 12.2.4. Результат `ShowFs(max(X'),std(X'),Y)`.

На первый взгляд подобные картинки не раскрывают никаких закономерностей, но если внимательно посмотреть рис. 12.2.4, то видно, что точки одного класса слегка «окружают» точки другого, а если её увеличить (см. рис. 12.2.5), то видно, что часть точек одного из классов образует плотный сгусток. Эвристика **`max(X')<=350`** уже позволяет безошибочно отнести к одному из классов 622 сигнала обучения.

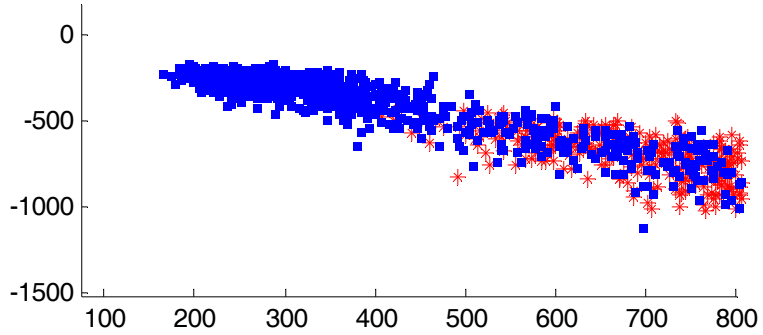


Рис. 12.2.5. Увеличение результата `ShowFs(max(X'),std(X'),Y)`.

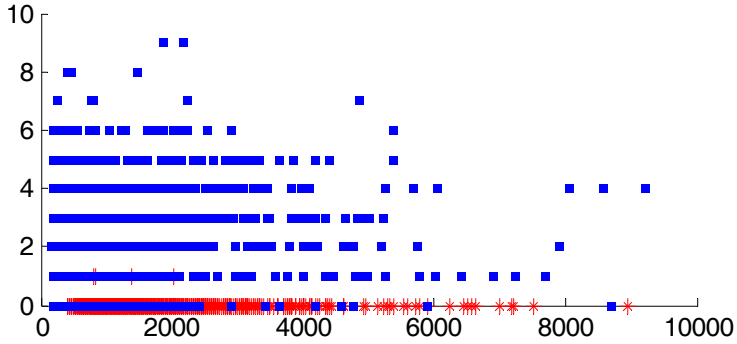


Рис. 12.2.6. Результат `ShowFs(max(X'), sum(diff(X')==0),Y)`.

На рис. 12.2.6 видно, что также неплохим признаком оказывается

`sum(diff(X')==0)`,

т.е. число точек с нулевой производной[1]. Этот признак можно обобщить. По сути, мы посчитали число точек, в которых для сигнала $\tilde{u} = (u_1,\ldots,u_n)$ значения u_i и u_{i+1} совпадают, т.е.

$$|\{i \in \{1,2,\ldots,n-1\} \,|\, u_i = u_{i+1}\}|.$$

Первое естественное обобщение – число незначительно отличающихся соседних точек:

$$|\{i \in \{1,2,\ldots,n-1\} : |u_i - u_{i+1}| \le \varepsilon\}|.$$

Второе – число повторов значений в сигнале. Это можно по-разному формализовать. Например, в векторе (1,2,0,1,3,3,1,4,1) четыре раза повторяется значение 1, два раза значение 3, остальные по одному. Если его отсортировать, то получим вектор (0,1,1,1,1,2,3,3,4), разностная производная (оператор **`diff`** в MatLabe) от которого даёт вектор (1,0,0,0,1,1,0,1), т.е. нулями помечены наши повторы. Поэтому простая MatLab-команда, которая формализует второе обобщение найденного хорошего признака, – **`sum(diff(sort(X'))==0)`**.

[1] На самом деле, конечно, разностной производной.

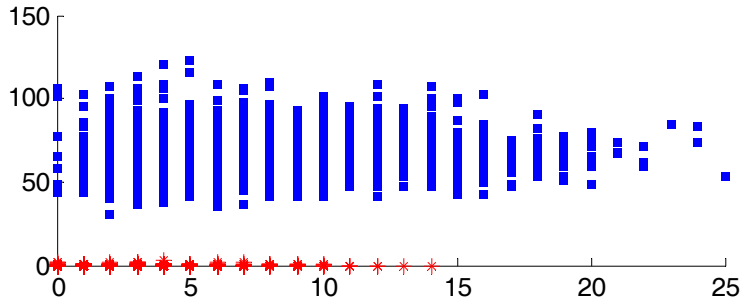


Рис. 12.2.7. Результат `ShowFs(sum(abs(diff(X'))<=1), sum(diff(sort(X'))==0),Y)`.

Как видно на рис. 12.2.7, второе обобщение «сработало», причём на 100%! Алгоритм, который разделяет по этому признаку, единственный из всех участников показал стопроцентный результат верной классификации на контрольной выборке Ford_A соревнования «Ford Classification Challenge» [Ford], хотя вектор классификации получается простой MatLab-командой

`2*(sum(diff(sort(X'))==0)<20)'-1`).

Это и есть «шаманство в анализе данных», когда ответ задачи кроется в 33 символах, при этом не надо строить ядра, восстанавливать плотности, настраивать нейросеть и придумывать метрику. Надо просто посмотреть на данные…

§12.3. Поиск хороших признаков в задаче классификации сигналов

Теперь покажем, как автоматизировать поиск «хороших» признаков. Рассмотрим упрощённую задачу: поиск одного признака, поскольку часто в задачах классификации сигналов удаётся найти именно такой признак и порог (показательный пример приведён выше). Как мы строили наши признаки? В виде выражений вида

`sum(diff(sort(X(:,1:250)'))<=3)`.

Рассмотрим конкретно это выражение. Сначала мы выделили подсигнал `X(:,1:250)`, потом отсортировали значения этого сигнала функцией `sort()`, т.е., по сути, от одного временного ряда перешли к другому (как и при выделении подсигнала), затем взяли производную функцией `diff()` (опять перешли к новому временному ряду), потом посчитали, сколько в полученном сигнале значений не превосходит числа 3: `sum()<=3` (здесь уже по временному ряду получаем число).

Пусть теперь признак, который является функцией «описание сигнала – число», представ***и***м в виде

$$C_{i_0}(B_{i_k}(\ldots(B_{i_1}(\tilde{x})))), \qquad (12.3.1)$$

где $k \in \{1,2,\ldots\}$, $B_{i_k},\ldots,B_{i_1} \in B^*$, B^* – множество операторов первого типа, $C_{i_0} \in C^*$, C^* – множество операторов второго типа. Операторы первого типа переводят сигнал в сигнал. Примеры таких операторов:

1) сглаживание (замена значения в каждой точке средним значением в окрестности этой точки) и фильтрация сигнала,
2) взятие производной (конечной разности),
3) сортировка значений,
4) удаление из сигнала точек, обладающих некоторым свойством («прореживание сигнала», выделение подсигналов),
5) взятие поточечной функции от значений сигнала (в частности логической, типа **`abs()<=10`**).
6) различные преобразования сигнала типа дискретного преобразование Фурье, вообще, разложение сигнала по базису и «порождение» сигнала со значениями, равными коэффициентам этого разложения.

Операторы второго типа получают по сигналу число. Примеры:
1) значение сигнала, обладающее определённым свойством (в том числе, максимальное значение сигнала, минимальное значение, среднее значение),
2) число значений сигнала, обладающих некоторым свойством (в том числе, число локальных максимумов, число значений из заданного интервала и т.д.),
3) некоторая статистика значений сигнала (например выборочные моменты).

Теперь задача поиска хорошего признака свелась к
1) созданию библиотек операторов первого и второго типа,
2) перебору операторов в выражении (12.3.1) и оценки качества признаков с помощью функционала качества признаков.
В качестве такого функционала подойдёт ROC-AUC. Перебор можно осуществлять методами глобальной оптимизации (см. главу 8), в том числе генетическими алгоритмами. Отметим, что перебор надо вести «разумно». Каждый оператор имеет некоторые параметры. Например, оператор **`diff( ,k)`** взятия *k*-й производной имеет параметр *k*. Если настраивать значения этих параметров (а это следует делать), то суперпозиции вида **`diff(diff( ,s) ,k)`** не следует рассматривать, поскольку они эквивалентны выражениям вида **`diff( ,s+k)`**. Кроме того, некоторые суперпозиции операторов первого и второго типа априорно более осмыслены, чем остальные. Например, после разложения сигнала по базису в качестве признаков используют значения коэффициентов разложения, а не применяют к последовательности коэффициентов операторы типа **`diff`**.

Создание подходящей библиотеки не является тривиальной задачей, поскольку она должна быть проблемно-ориентированной, достаточно большой, чтобы «было из чего выбирать», и не очень большой, чтобы можно было осуществить «почти полный» перебор и не пропустить хороший признак. Данный подход легко обобщается и для поиска признакового пространства.
ДЗ *Какие функционалы качества лучше использовать для оценки совокупности признаков?*

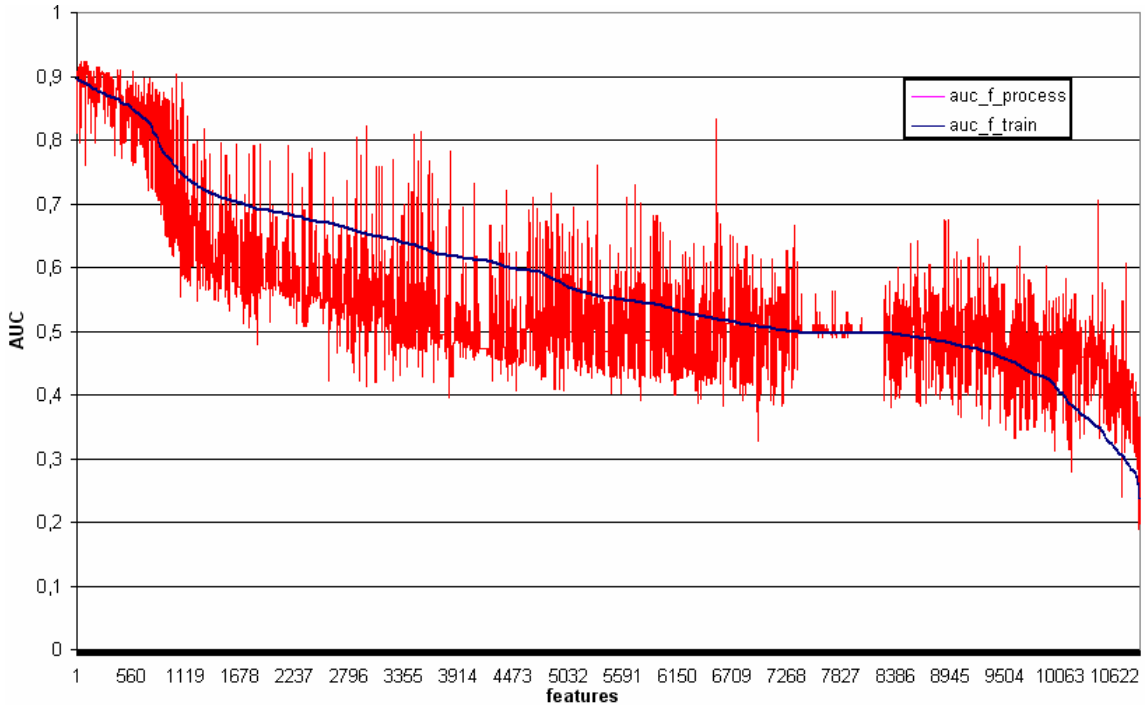


Рис. 12.3.1. Качество признаков при полном переборе.

Интересный эффект виден на рис. 12.3.1[1]. Для проблемно-ориентированной библиотеки в задаче §12.1 проведён полный перебор всех признаков (число операторов первого типа в признаковом выражении было не больше 4х). Для каждого признака вычислено качество на обучении и контроле с помощью ROC-AUC-функционала. На графике представлены кривые этих показателей, упорядоченные по значению качества на обучении (поэтому одна кривая монотонно убывает). Видно, что есть эффект переобучения: широкая (поскольку у неё большие скачки) кривая ошибки на контроле лежит ниже кривой ошибок на обучении. При этом кривая «настолько широкая», что у признаков с примерно одинаковым качеством на контроле может быть совершенно различное качество на обучении: разница достигает 35%! Более того, иногда (но очень редко) качество на контроле может быть выше (и это тоже переобучение).

Вообще, здесь два вида переобучения:
1) переобучение модели (кривая контроля лежит ниже кривой обучения),
2) переобучение конкретного алгоритма, в данном случае признака[2] (ширина кривой обучения).

Но самое интересное, что у «хороших» алгоритмов (здесь: признаков) эффект переобучения проявляется меньше. В левой части графика две кривые лежат на одном уровне и кривая обучения не такая широкая, т.е. если мы найдём какой-то хороший (для обучения) признак, то он останется хорошим и

[1] Взят из дипломной работы студентки ВМК МГУ 2009 г.в. Власовой Ю.В. «Генерация признаков в задаче классификации сигналов».
[2] Алгоритм будет задаваться ещё и значением порога.

на контроле. А найти хороший признак не так уж и сложно, поскольку их достаточно много (см. график) и даже случайным поиском можем на него наткнуться. Эти выводы могут показаться искусственными и справедливыми для одной конкретной задачи, но, как показывают эксперименты, для хорошо подобранной библиотеки операторов во многих задачах строятся аналогичные рис. 12.3.1 графики, хотя для некоторых задач эффект переобучения гораздо больше.

Подходы, основанные на переборе «кирпичиков», из которых строятся признаки или даже сами алгоритмы, достаточно часто применяются на практике (см., например, [Стрижов, Пташко, 2007]).

§12.4. Прогнозирование временных рядов

Рассмотрим задачу прогнозирования одномерных равномерных временных рядов. Задача состоит в продолжении набора $f_1,\ldots, f_n$ (чисел из $\mathbf{R}$), т.е. «предсказании» значений $f_{n+1},\ldots, f_{n+t}$. Специфику задачи очень просто понять на модельных примерах: $(\sin(j))_{j=1}^{n}$, $(aj^2+bj+c)_{i=1}^{n}$.

В качестве примеров реальных данных будем изображать ряды соревнования «Forecasting Competition for Neural Networks & Computational Intelligence» [NN3], [NN5].

Рассмотрим «здравый», но немного примитивный подход к прогнозированию… Если в ряде есть кусочек $f_s,\ldots, f_{s+l-1}$, который «очень похож» на кусочек $f_{n-l+1},\ldots, f_n$, тогда и его продолжение $f_{s+l},\ldots, f_{s+l+t-1}$ будет очень похоже на продолжение $f_{n+1},\ldots, f_{n+t}$. Это, так называемый, **локальный подход**, когда мы не ищем представление ряда в виде параметрической функции F_a:

$$\sum_{j=1}^{n} \| f_j - F_a(j) \| \to \min_a,$$

а пользуемся информацией только о маленьком участке ряда.

Осталось формализовать, что такое «очень похож». Если искать вектор $(f_s,\ldots, f_{s+l-1})$, который близок к вектору $(f_{n-l+1},\ldots, f_n)$ в евклидовой метрике, то наш подход будет применим к узкому классу рядов. Он даже «не справится» с линейным рядом $(aj+b)_{j=1}^{n}$. Поэтому близость надо искать с точностью до некоторого преобразования A:

$$\min_{s,A} \| A(f_s,\ldots, f_{s+l-1}) - (f_{n-l+1},\ldots, f_n) \|,$$

(A пробегает некоторый класс преобразований). Тогда прогноз запишется в виде $A(f_{s+l},\ldots, f_{s+l+t-1})$.

Класс преобразований, который следует рассматривать, зависит от прогнозируемых рядов. Самый простой класс – сжатия-растяжения по вертикали и параллельный перенос (по вертикали), т.е. линейные преобразования:

$$A_{a,b}(f_s,\ldots,f_{s+l-1}) = (a\cdot f_s + b,\ldots,a\cdot f_{s+l-1} + b).$$

Фактически мы ищем ближайшего соседа последнего отрезка ряда. Можно искать несколько соседей, а результат усреднять. Запишем соответствующий MatLab-код для экспериментов.

```
% 1й порядок kNN ДЛЯ ЭКСПЕРИМЕНТОВ по прогнозированию

% g - ряд
% l - размер окна (соседства)
% t - сколько (точек) прогнозировать
% K - какие соседи (K = [1] - ближайший, [2 3] - второй и третий)
% isGraph – построить график (или нет)

function z = forecNNK(f, l, t, K, isGraph)

if (size(f,1)~=1) % получаем строчку
    f = f';
    if (size(f,1)~=1)
        error('f - не строка');
    end;
end;

% очистить от последних NaNов
f(isnan(f)) = [];

if (isGraph)
    % отбросить последний кусочек
    endf = f(end-t+1:end);
    f(end-t+1:end) = [];
end;

n = length(f); % длина ряда

x = f(end-l+1:end); % то, что приближаем

A = ones([l 2]); % шаблон (для заполнения)

E = zeros([1 n - l - t]); % погрешности

% формируем ряд погрешностей приближений
for i = 1 : n - l - t %число строк X
    A(:,2) = f(i:i+l-1)';
    c = (A'*A)\(A'*x');
    e = (A*c - x')'*(A*c - x');
    E(i) = e;
end;
```

```
[E imax] = sort(E);

z = zeros([1 t]);
A2 = ones([t 2]);

for j = K
    A(:,2) = f(imax(j):imax(j)+l-1)';
    c = (A'*A)\(A'*x');

    A2(:,2) = f(imax(j)+l:imax(j)+l+t-1)';
    z = z + (A2*c)'; % прогноз
end;

z = z./length(K);

% графика
if (isGraph)
    clf;
    hold on;
    plot([f z],'k');
    plot([f endf],'b');
    plot(n-l+1:n, x, 'r','LineWidth',2); % то приближение чего
ищем...
    text(n-l+1, x(1), 'Window', 'FontSize', 10);
    text(n+1, x(end), 'Forecast', 'FontSize', 10);
    for j = K
        % что нашли
        plot(imax(j):imax(j)+l-1, f(imax(j):imax(j)+l-1), 'k',
'LineWidth',2);
        text(imax(j)+1, f(imax(j)), ['NN ' int2str(j)],
'FontSize', 10);
    end;
    xlabel('i');
    ylabel('f_i');
    % вывод ошибки
    title(['Accuracy = ' num2str(sqrt(sum((z - endf).^2)))]);
end;
```

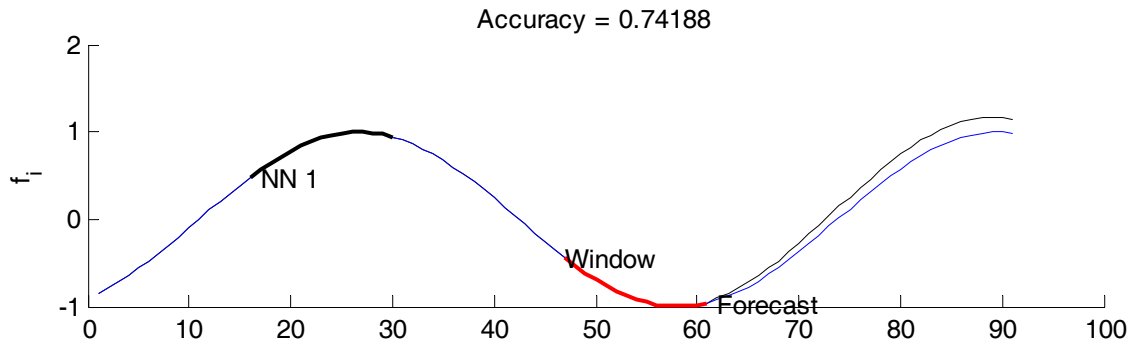


Рис. 12.4.1. Результат вызова `forecNNK(sin(-1:0.1:8),15,30,[1], true)`.

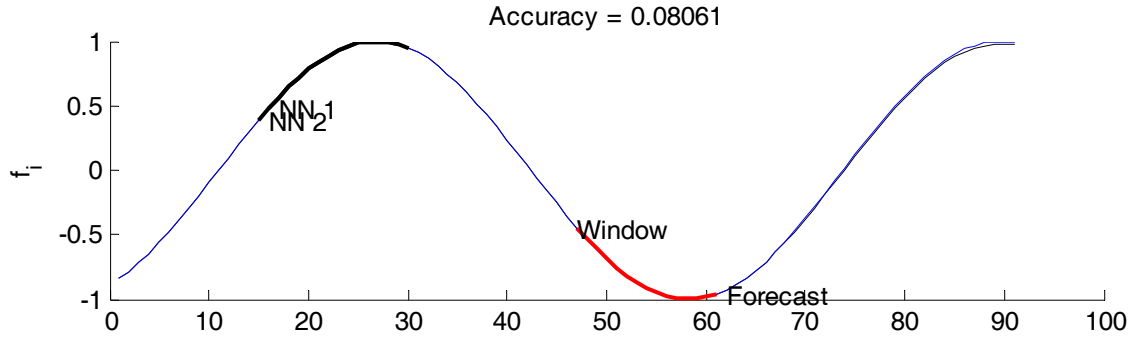


Рис. 12.4.2. Результат вызова `forecNNK(sin(-1:0.1:8),15,30,[1 2], true)`.

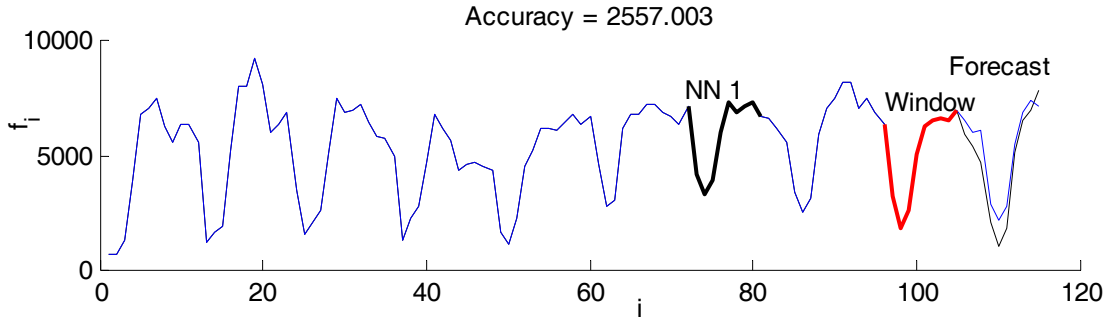


Рис. 12.4.3. Прогнозирование реального ряда с одним соседом.

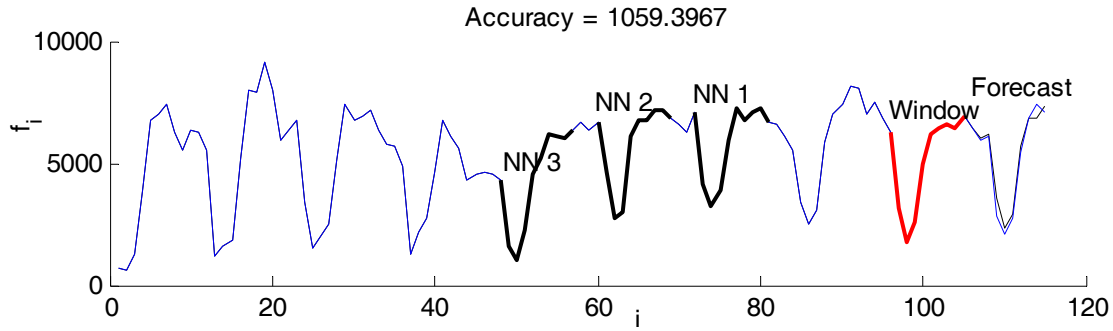


Рис. 12.4.4. Прогнозирование реального ряда с тремя соседями.

Заметим, что в процессе поиска «наиболее похожего» отрезка ряда происходит решение системы уравнений

$$\begin{bmatrix} 1 & f_s \\ \vdots & \vdots \\ 1 & f_{s+l-1} \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} f_{n-l+1} \\ \vdots \\ f_{s+l-1} \end{bmatrix}.$$

Метод можно обобщить, изменив матрицу в левой части. Её первый столбец отвечает за сдвиг по вертикали (величину сдвига определяет значение c_1), а на место второго столбца в процессе перебора подставляются всевозможные отрезки ряда. Можно искать приближение в виде линейной комбинации нескольких отрезков, т.е. перейти к системе

$$\begin{bmatrix} 1 & f_s & f_r \\ \vdots & \vdots & \vdots \\ 1 & f_{s+l-1} & f_{r+l-1} \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} f_{n-l+1} \\ \vdots \\ f_{s+l-1} \end{bmatrix}.$$

Интересно, что такое обобщение «здорово» работает на многих модельных рядах (см. рис. 12.4.5, обратите внимание на точность прогноза), но плохо работает на реальных (по крайней мере, данных [NN3]). **ДЗ** *Объясните этот эффект. Проведите эксперименты с бо**л**ьшим числом столбцов матрицы.* Также можно формировать столбцы этой матрицы специальным образом, например, вставляя столбец $(1,2,\ldots,l)^{\mathrm{T}}$ для удаления линейного тренда.

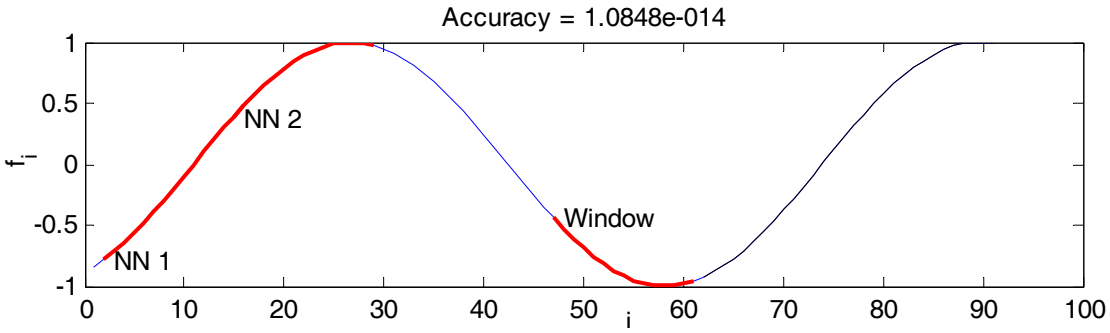


Рис. 12.4.5. Пример прогноза с помощью наилучшей линейной комбинации двух отрезков на модельном ряде.

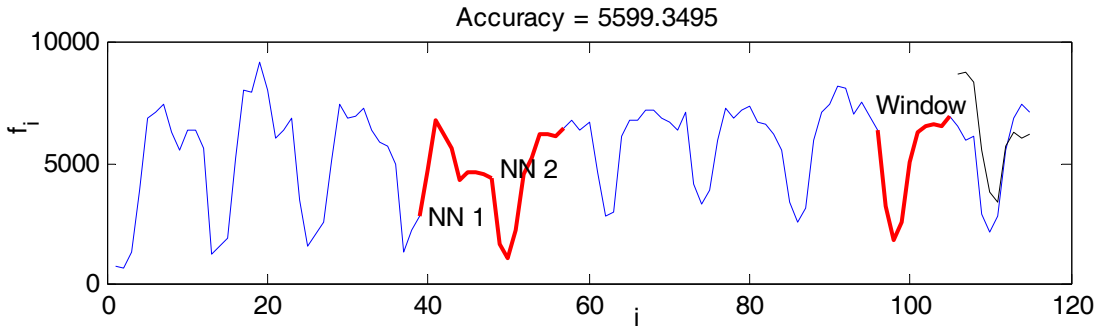


Рис. 12.4.6. Пример прогноза с помощью наилучшей линейной комбинации двух отрезков на реальном ряде.

Естественно, локальный подход имеет ограниченную область применения, но при анализе временного ряда именно локальные алгоритмы бывает очень полезно попробовать в первую очередь. Многие алгоритмы прогнозируют значения временного ряда, используя информацию только на последнем его отрезке[1], поэтому локальный подход помогает оценить, насколько это оправдано. Рекомендуется также проанализировать матрицу попарных близостей отрезков временного ряда, ij -й элемент которой имеет вид

$$\min_A \| A(f_i,\ldots,f_{i+l-1}) - (f_j,\ldots,f_{j+l-1}) \|$$

(матрица имеет размеры $(n-l+1)\times(n-l+1)$).

Такая матрица изображена на рис. 12.4.7 для ряда рис. 12.4.3. На ней хорошо видны диагональные полосы, которые расположены на одинаковом расстоянии друг от друга (это соответствует периоду ряда). Яркие

[1] Часто не только, но информация о начальных значениях берётся с меньшим весом.

вертикальные полосы соответствуют участкам, которые сложно приблизить с помощью других.

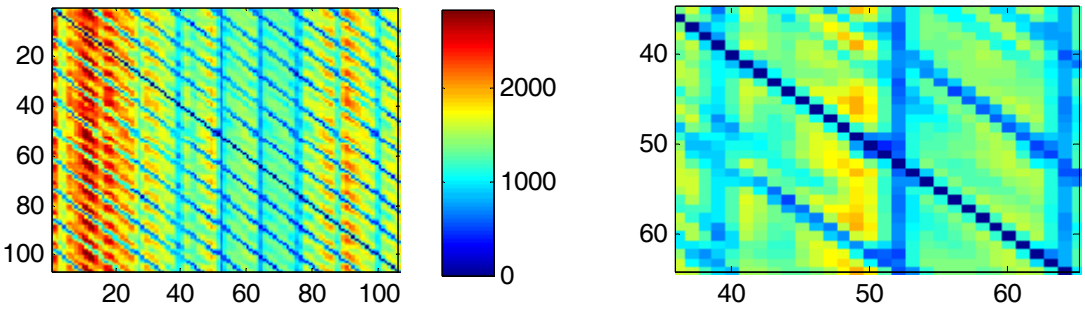


Рис. 12.4.7. Матрица попарных расстояний (результат imagesc) и её увеличенный фрагмент (справа).

Мы обозначили лишь идею локального подхода. Существует множество его обобщений:

1. Искать похожие участки можно не на самом графике, а в пространстве траекторий $(f_s,\ldots,f_{s+l})_{s=1}^{n-l}$ (или в другом пространстве).
2. Если в ряде есть выбросы, то разумно сначала их удалить (заменить соответствующие значения средним по окрестности) или измерять близость отрезков ряда с учётом выбросов (после нахождения преобразования A в векторе $A(f_s,\ldots,f_{s+l-1})-(f_{n-l+1},\ldots,f_n)$ найти координату с максимальным модулем и пересчитать преобразование A для векторов без этой координаты, т.е. устранить локальный выброс).

При использовании локального подхода необходимо исследовать его применимость для данного класса рядов. Часто он очень неустойчив относительно длины отрезка l, по которому мы определяем сходство. Кроме того, на многих реальных рядах лучше пользоваться сходством не с ближайшим соседом, а с другими (на данных [NN3] неплохо показало себя сходство с 3, 5 и 7 соседями!). Этот эффект можно проиллюстрировать следующим экспериментом. Для известного продолжения ряда $(f_{n+1},\ldots,f_{n+t})$ посчитаем ошибку приближения «предыстории»:

$$\min_A \| A(f_s,\ldots,f_{s+l-1})-(f_{n-l+1},\ldots,f_n)\|$$

(«ошибку на обучении») и «продолжения»

$$\| A(f_{s+l},\ldots,f_{s+l+t-1})-(f_{n+1},\ldots,f_{n+t})\|$$

для всех $s=1,2,\ldots,n-l+1$ («ошибка на контроле»). Графики этих ошибок представлены на рис. 12.4.8 (по горизонтали отложены $s=1,2,\ldots,n-l+1$). Даже когда графики «достаточно синхронны», не всегда минимальным значениям одной ошибки соответствуют минимальные ошибки другой (для многих классов рядов такие несоответствия типичны).

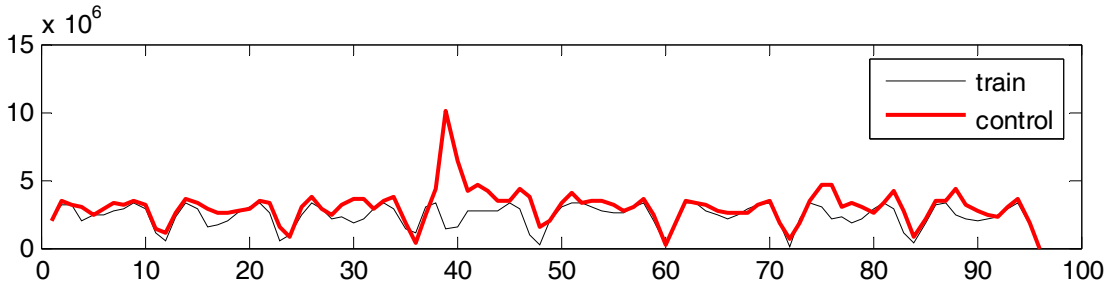


Рис. 12.4.8. Ошибка приближения отрезков (train) и их продолжений (control) для ряда рис. 12.4.3 – 12.4.4.

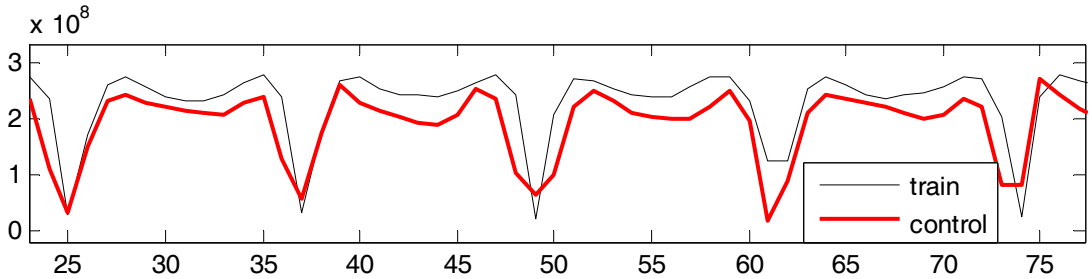


Рис. 12.4.9. Фрагмент графика ошибки приближения отрезков (train) и их продолжений (control).

ДЗ *Напишите функцию, которая строит графики таких ошибок, а также ошибки*

$$\min_{B} \| B(f_{s+l},\ldots, f_{s+l+t-1}) - (f_{n+1},\ldots, f_{n+t}) \|$$

(*ошибка нашего метода при знании оптимального преобразования*).

Общая рекомендация этой главы – в любой задаче анализа данных **необходимо сначала посмотреть на данные**, попытаться найти скрытые закономерности, чтобы определить метод решения. Конечно, для этого необходимо найти наилучший способ представления имеющейся информации. На соревновании [NN5] ряды, отражавшие объём снятых с банкоматов денег, не поддавались стандартным методам прогноза. Пример ряда показан на рис. 12.4.10. Основной проблемой, кроме «неочевидного поведения рядов», было наличие пропусков и нулевых значений (они, по смыслу, были выбросами, но функционал качества был составлен так, что эти выбросы желательно уметь прогнозировать).

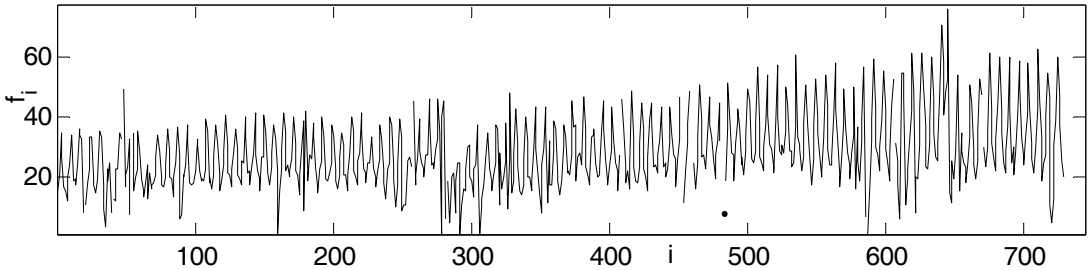


Рис. 12.4.10. Один из рядов соревнования [NN5].

Если разбить ряды [NN5] на подряды по семь точек (что соответствует снятиям в течение недели), то природа рядов станет понятной (см. рис. 12.4.11): все такие подряды лежат «в одной полосе», причём соседние недели очень похожи друг на друга.

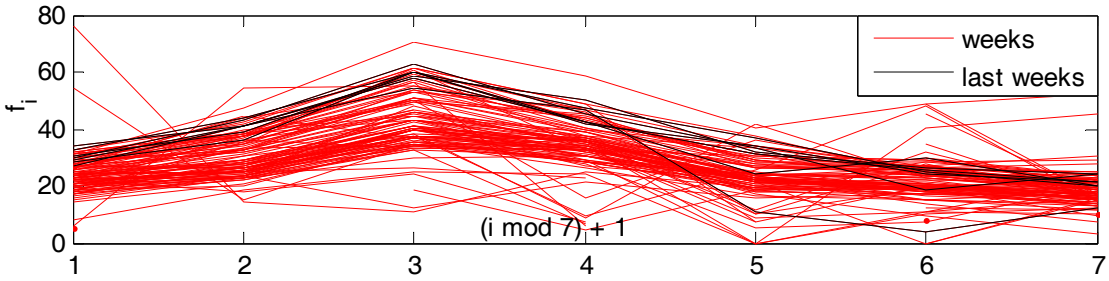


Рис. 12.4.11. Ряд рис. 12.4.10, записанный «по неделям».

Этот факт позволил изобрести подходящий алгоритм (который занял на соревновании [NN5] 2е и 3е места): усредняем последние недели и выдаём полученное в качестве прогноза, смотрим, насколько надо подкорректировать наш ответ, чтобы уточнить прогноз. Проходим нашим алгоритмом по всему ряду (пытаемся прогнозировать каждую неделю) и получаем ряды корректировок (значения параметров преобразований ответов нашего «примитивного» алгоритма). Теперь задача свелась к прогнозированию ряда корректировок (её уже удаётся хорошо решить стандартными методами).

Глава 13. ПРОГРАММА ДЛЯ АНАЛИЗА ДАННЫХ WEKA

Программа **WEKA**[1] (см. [Weka], [Weka Wiki], [Weka, 2009]) написана на языке Java в университете Вайкато (Новая Зеландия), распространяется под лицензией GNU General Public License version 21 (GPLv2) и предоставляет пользователю возможность предобработки данных, решения задач классификации, регрессии, кластеризации и поиска ассоциативных правил, а также визуализации данных и результатов. Программа очень проста в освоении (пожалуй, имеет самый интуитивный интерфейс среди всех программ такого типа), бесплатна и может быть дополнена новыми алгоритмами, средствами предобработки и визуализации данных.

Ссылку на последнюю версию можно найти на официальном сайте [Weka]. С инсталляцией особых проблем возникнуть не должно. Для работы необходимо ПО J2SE Runtime Environment 5.0 Update 22 (см. [Sun], может поставляться вместе с программой). Хорошо откомментированный код программы можно найти в файле **weka-src.jar** (если Вы не знакомы с программированием на Java, то считайте, что это обычный zip-архив, который можно распаковать).

Для начала работы достаточно запустить файл **RunWeka.bat**. Появится окно **Weka GUI Chooser** (см. рис. 13.1), в котором будет предложено выбрать один из четырёх модулей программы. Рассмотрим их подробно.

Рис. 13.1. Основное окно программы WEKA.

§13.1. Модуль Explorer

Это основной модуль программы (см. рис. 13.1.1), который позволяет загрузить и предобработать данные (вкладка **Preprocess**), решить задачу классификации (**Classify**), кластеризации (**Claster**), поиска ассоциаций (**Associate**), селекции признаков (**Select Attributes**) и визуализации (**Visualize**).

[1] WEKA = Waikato Environment for Knowledge Analysis.

Каждая задача имеет свою вкладку в общем окне. Сначала доступна только вкладка **Preprocess**, поскольку для выполнения остальных задач нужны данные. Отметим, что последовательность вкладок не всегда соответствует этапам решения задачи. Например, после загрузки данных в память ЭВМ можно перейти к селекции признаков. Рассмотрим работу на соответствующих вкладках.

1. Вкладка **Preprocess** (рис. 13.1.1).

Изначально вверху окна активны четыре кнопки, которые позволяют загрузить данные из файла (**Open file**), из Интернета (**Open URL**), из базы данных (**Open DB**) и сгенерировать модельные данные (**Generate**). Чаще всего приходится пользоваться данными из файла. Нажав на кнопку «**Open file**», в списке расширений файлов можно посмотреть, какие форматы понимает WEKA (arff, CSV, C4.5, бинарные, libsvm). Основной формат – **arff** (файлы с расширением arff). В каталоге **data** можно посмотреть примеры arff-файлов (с программой поставляются данные нескольких задач). Пример содержания такого файла:

example.arff

```
% комментарий
@RELATION myproblem

@ATTRIBUTE firstfeature REAL
@ATTRIBUTE secondfeature REAL
@ATTRIBUTE noise REAL
@ATTRIBUTE class {A,B}

@DATA
0,0,1.1,A
0,0,3.2,A
0,0,4.7,A
0,0,3.1,A
0,1,2.2,B
0,1,4.3,B
1,0,1.1,B
1,0,2.5,B
1,1,3.7,A
1,1,4.2,A
1,1,3.9,A
1,1,5.1,A
```

Сначала идёт название задачи (секция **@RELATION**), потом описание признаков: их названия и типы (каждое описание после ключевого слова **@ATTRIBUTE**), затем по строкам перечислены сами данные (секция **@DATA**). Кроме номинальных и числовых признаков, могут быть также признаки типа **string**, **date**, **relational** (см. [WekaManual-3-7-1.pdf]). Пропуск в данных кодируется символом «?» (вопросительный знак). Есть также формат для

представления разреженных матриц данных. Обычный arff-формат и формат для разреженных матриц различаются только оформлением секции данных (**@DATA**):

`% в обычном arff-файле`
 	
`@data`
`0, 0, 11, A`
`12, 0, ?, B`
	`% при описании разреженных`
	`% матриц данных`
	`@data`
	`{ 2 11, 3 A }`
	`{ 0 12, 2 ?, 3 B }`

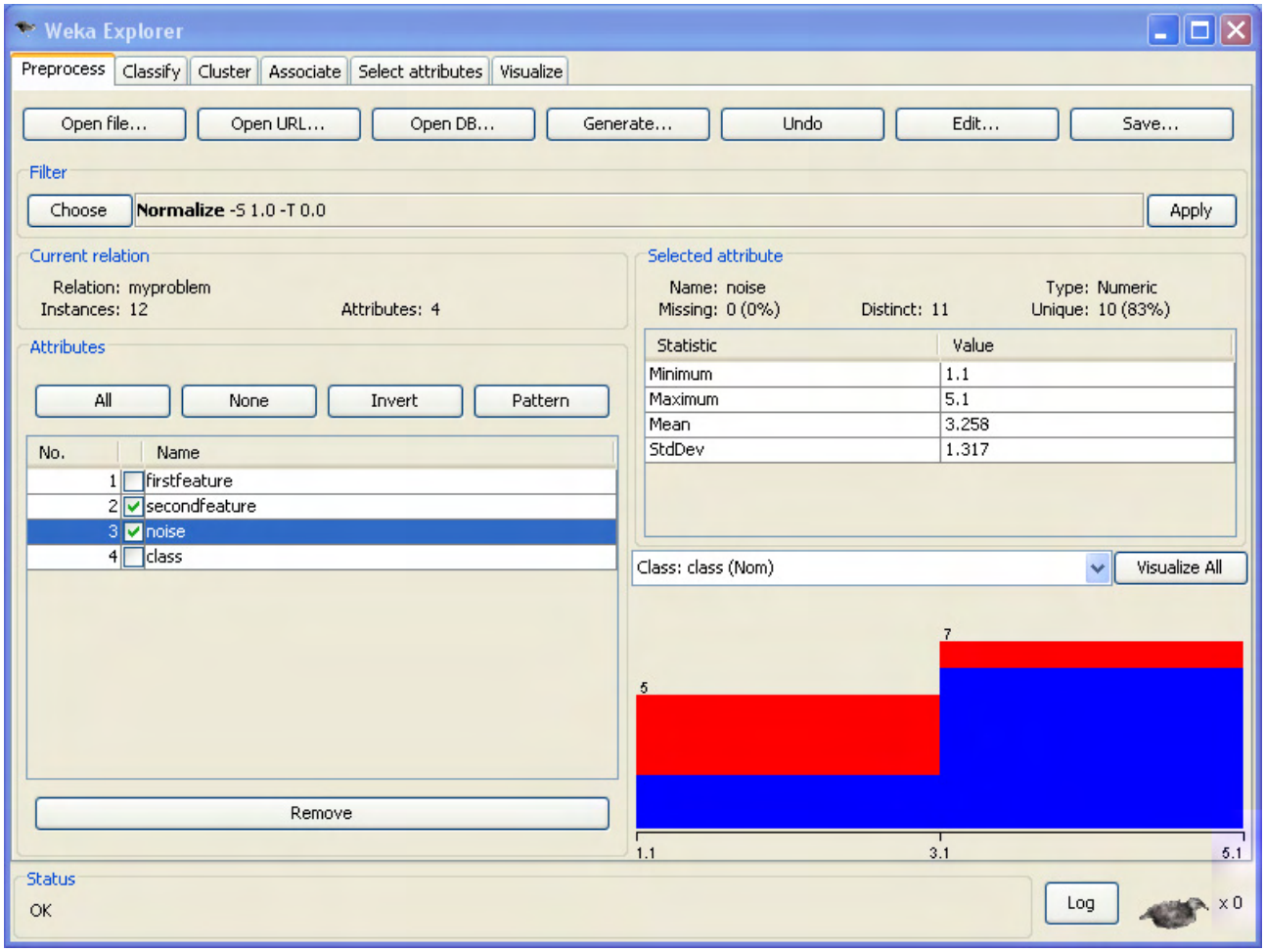


Рис. 13.1.1. Вкладка Preprocess модуля Explorer (выделены два признака: 2 и 3, для выбранного третьего признака построена гистограмма значений).

После загрузки файла на левой нижней панели **Attributes** можно выбрать признак (см. рис. 13.1.1), при этом на правой панели **Selected attribute** отображается гистограмма значений признака. Можно посмотреть гистограммы всех признаков нажатием кнопки **Visualize all**. Слева от этой кнопки находится компонент выбора целевого признака: какой признак считать классом при визуализации (выбор **NoClass** соответствует тому, что все объекты приписываются одному классу и для каждого признака показывается гистограмма распределения всех его значений). В верхней части панели **Selected attribute** отображается простая статистика: максимальное, минимальное и среднее значения признака, выборочная дисперсия, число

пропусков, тип признака, число уникальных значений. Нажатие на кнопку **Remove** приводит к удалению признаков, помеченных галочками на левой нижней панели **Attributes**. Нажатие на кнопку **Edit** позволяет редактировать значения признаков: появляется **Viewer** (см. рис. 13.1.2). Нажатие на кнопку **Save** сохраняет данные.

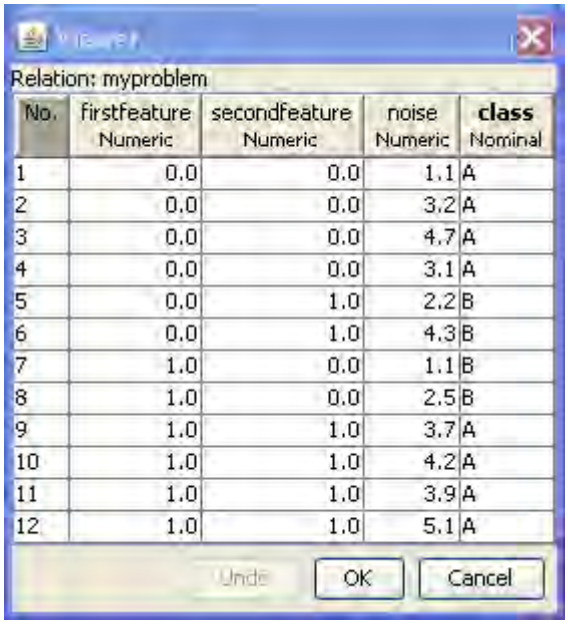


Рис. 13.1.2. Редактор значений признаков.

На верхней панели **Filter** можно выбрать преобразования данных. Например, нажатием кнопки **Choose** из иерархического списка выберем опцию

Weka/filters/unsupervised/attribute/Normilize/.

На панели отобразится название этого **фильтра** (так в системе WEKA называются процедуры предобработки данных), щёлкнув по нему правой клавишей мыши можно изменить его параметры. **Важно:** некоторые фильтры имеют параметры, определяющие, к каким признакам эти фильтры применять (т.е. область применения не всегда определяется пометками на панели **Attributes**). Нажатие кнопки **Apply** запускает фильтр. В данном случае происходит нормализация, эффект которой заметен только для третьего признака: его значения переводятся на отрезок [0,1].

На нижней панели **Status** отображается статус программы и изображение птички Киви[1]. Если она бегает, то программа производит вычисления, если сидит неподвижно, то программа находится в режиме ожидания. После значка «×» отображается число запущенных процессов.

2. Вкладка **Classify** (рис. 13.1.4).

На панели **Classifier** можно выбрать классификатор. Для этого нажимаем на кнопку **Choose** и выбираем из иерархического списка, например

[1] Птица Киви является символом Новой Зеландии. Она изображена также на главном окне программы Weka GUI Chooser.

weka/classifiers/functions/MultilayerPerceptron (для выбора многослойного персептрона). Как и для фильтров, нажатие правой кнопки мыши по появившемуся на панели **Classifier** названию позволяет выбрать параметры классификатора. Если для многослойного персептрона выбрать значение параметра **GUI** равным **true**, то появится возможность редактировать нейронную сеть (после нажатия кнопки Start, см. дальше и рис. 13.1.3).

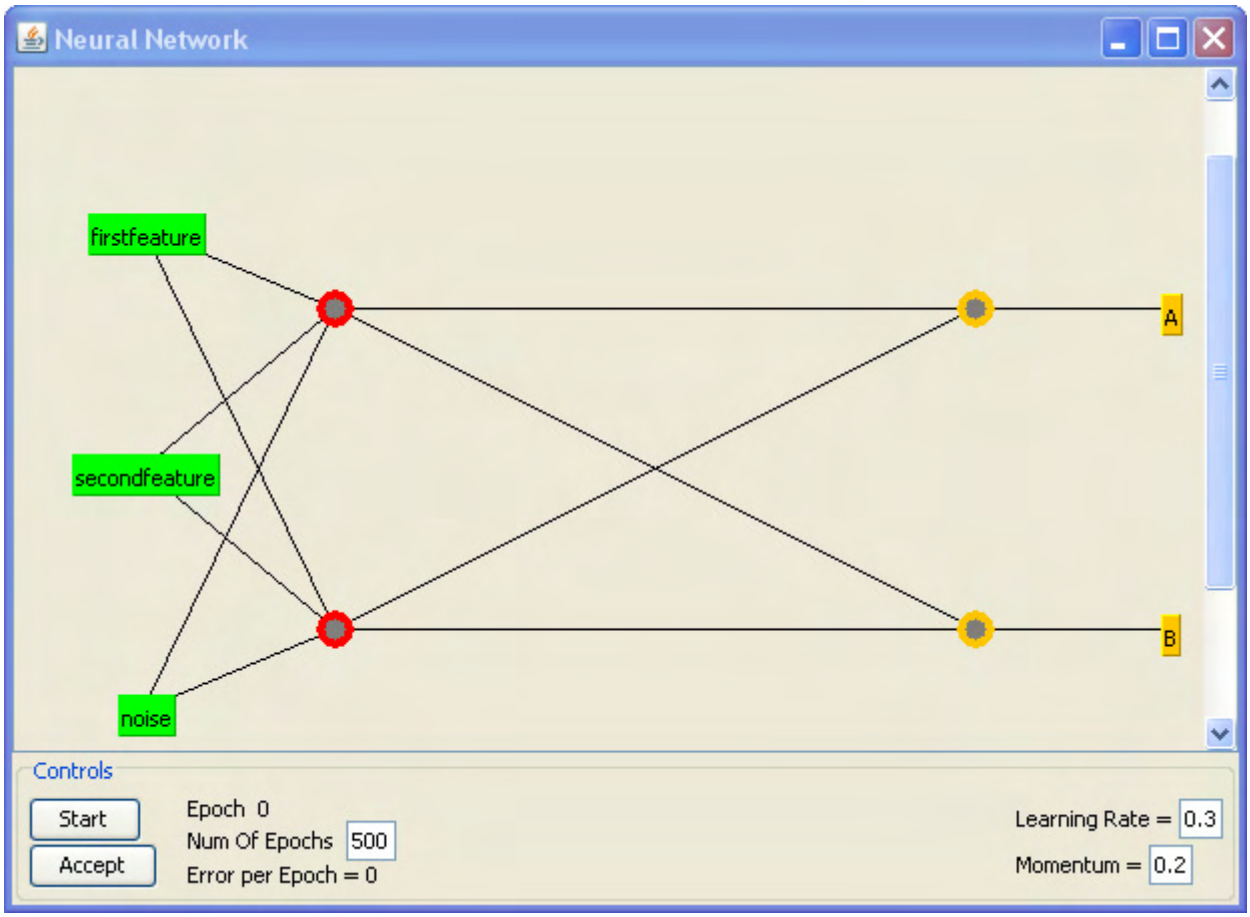


Рис. 13.1.3. Вид нейронной сети (его можно редактировать).

На панели **Test options** определено, как будет тестироваться классификатор: на обучающей выборке (**use training set**), на тестовой из отдельного файла (**supplied test set**), по блокам (**cross-validation**), с помощью разделения исходной выборки на обучение и контроль (**percentage split**). При выборе некоторых опций придётся указать параметры тестирования. Например, при выборе **cross-validation** надо указать, на сколько блоков (фолдов) разбивать выборку.

Очень полезная кнопка **More options**, которая позволяет выбрать вид отчёта об обучении классификатора. **Совет:** многие почему-то не знают, **как получить классификацию объектов** алгоритмом системы WEKA – достаточно просто включить опцию **More options/Output predictions**.

Ниже панели **Test options** находится компонент для выбора целевого признака (что будет считаться классом в задаче).

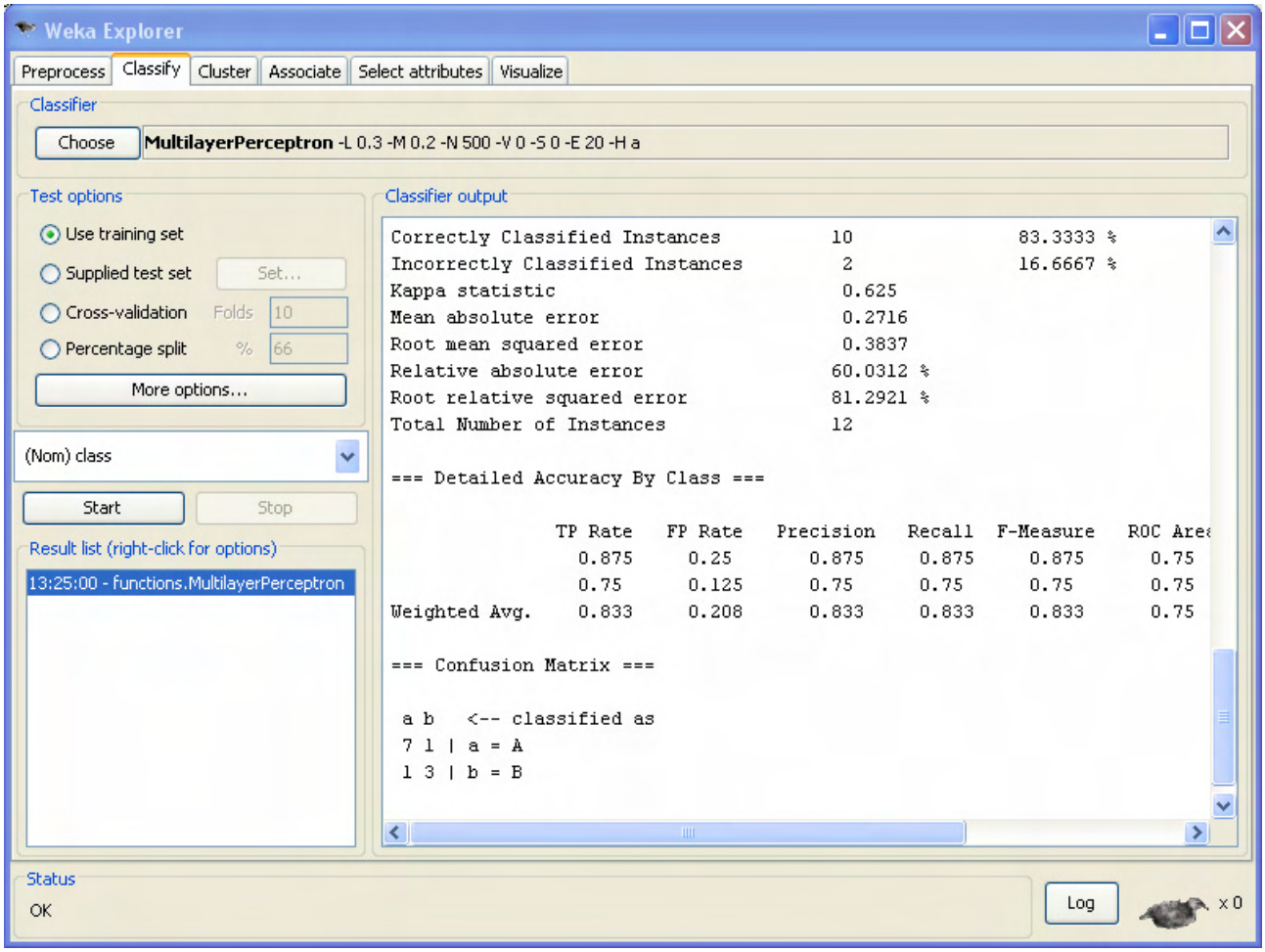


Рис. 13.1.4. Вкладка Classify модуля Explorer (выведен отчёт о настройке и тестировании алгоритма).

Нажатие на кнопку **Start** запускает обучение классификатора. На самой большой панели **Classifier output** отображается отчёт об обучении. Сначала отображается информация о задаче и алгоритме классификации.

```
   === Run information ===

   Scheme:       weka.classifiers.functions.MultilayerPerceptron -L
0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
   Relation:                                                  myproblem-
weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0
   Instances:    12
   Attributes:   4
                 firstfeature
                 secondfeature
                 noise
                 class
   Test mode:    evaluate on training data
```

Затем параметры настроенного классификатора:

```
=== Classifier model (full training set) ===

Sigmoid Node 0
    Inputs    Weights
    Threshold    -1.2925119736275923
    Node 2    2.3654729118347473
    Node 3    1.49677437820294
Sigmoid Node 1
    Inputs    Weights
    Threshold    1.2938640814197169
    Node 2    -2.3411075720959715
    Node 3    -1.5261235513997728
Sigmoid Node 2
    Inputs    Weights
    Threshold    0.3090319071556761
    Attrib firstfeature    -0.8186811880909343
    Attrib secondfeature    -0.2900149234184642
    Attrib noise    7.292295647103865
Sigmoid Node 3
    Inputs    Weights
    Threshold    0.13382796919598083
    Attrib firstfeature    -0.6828503739569654
    Attrib secondfeature    -0.3075255989524704
    Attrib noise    6.055013888651134
Class A
    Input
    Node 0
Class B
    Input
    Node 1
Time taken to build model: 0.03 seconds
```

Затем результаты классификации (если включена опция **More options/Output predictions**). В данном случае классификации обучения, поскольку выбран режим **use training set**. Обратите внимание, что отображается также полезная информация о «вероятности принадлежности классу[1]».

```
=== Predictions on training set ===

 inst#,    actual, predicted, error, probability distribution
     1       1:A       2:B      +   0.218 *0.782
     2       1:A       1:A          *0.874  0.126
     3       1:A       1:A          *0.929  0.071
     4       1:A       1:A          *0.851  0.149
     5       2:B       2:B           0.278 *0.722
     6       2:B       1:A      +  *0.927  0.073
```

[1] Точнее «оценка принадлежности классу», термин «вероятность» здесь символический.

```
     7       2:B       2:B        0.216 *0.784
     8       2:B       2:B        0.283 *0.717
     9       1:A       1:A       *0.843  0.157
    10       1:A       1:A       *0.914  0.086
    11       1:A       1:A       *0.887  0.113
    12       1:A       1:A       *0.928  0.072
```

Затем идёт различная статистика работы классификатора, включая процент верных ответов на контроле (см. рис. 13.1.4). Обратим внимание только на следующую информацию, которая традиционна для подобных систем.

```
=== Confusion Matrix ===

 a b   <-- classified as
 7 1 | a = A
 1 3 | b = B
```

Выводится матрица размера $l \times l$, где l – число классов, ij-й элемент матрицы равен числу объектов из i-го класса, которые были отнесены к j-му. Число верно классифицированных объектов равно сумме элементов, стоящих на главной диагонали. В выведенной статистике значение **True Positive (TP) rate** или **Recall (**для рассматриваемого класса**)** равно проценту верно классифицированных объектов класса (получается делением диагонального элемента на сумму элементов в его строке). Значение **False Positive (FP)** равно проценту объектов других классов, которые по ошибке занесены в рассматриваемый класс (если из матрицы вычеркнуть строку рассматриваемого класса, то значение равно сумме элементов столбца этого класса, делённое на сумму всех элементов). Значение **Precision** равно проценту верно классифицированных объектов из объектов, отнесённых алгоритмом к рассматриваемому классу (отношение диагонального элемента к сумме элементов столбца). Значение **F-Measure** вычисляется по формуле

2*Precision*Recall/(Precision+Recall)

(т.е. это среднее гармоническое Precision и Recall).

На левой нижней панели **Result List** выводится перечень всех запусков классификаций данных. Новый элемент в этом списке появляется после завершения настройки и тестирования классификатора (вызванных нажатием кнопки **Start**). В списке указывается время окончания классификации и название классификатора. Если по элементу списка **щёлкнуть правой кнопкой мыши**, то появится дополнительное меню, с помощью которого Вы сможете посмотреть отчёт в отдельном окне (**View in separate window**), загрузить и сохранить модель, т.е. тип и параметры классификатора (**Load model, Save model**), визуализировать ошибки классификатора (**Visualize classifier errors**, см. также работу с вкладкой **Visualize**), посмотреть различные

кривые отступов и ошибок. Для деревьев классификации доступна опция **Tree View**, результат см. на рис. 13.1.5.

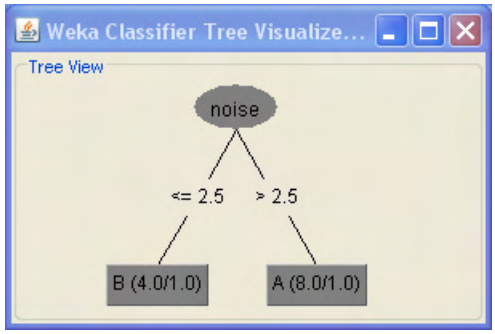


Рис. 13.1.5. Результат визуализации дерева (в задаче XOR с шумовым признаком дерево построилось с одним ветвлением по этому признаку!).

Внимание! Визуализация ошибок (специальная пометка неверно классифицированных объектов в проекциях на пары признаков) доступна только при выборе опции **Visualize classifier errors**. На вкладке **Visualize** можно будет посмотреть данные, их истинную классификацию, но не ошибки.

3. Вкладка **Visualize** (рис. 13.1.6).

Эта вкладка позволяет визуализировать выборку. Кнопка **Select Attributes** позволяет выбрать признаки для визуализации: будут построены картинки-проекции на всевозможные пары этих признаков. Ползунок **PlotSize** выбирает размер картинок, **PointSize** – размер точек, изображающих объекты, **Jitter** – уровень шумов, которые специально добавляются к признакам. Последний ползунок очень важен, поскольку для пары признаков, которые принимают небольшое число значений (например *k*-значных) целая группа объектов сливается в одну точку, а ползунок позволяет рассеять эту точку в облако.

Если кликнуть по одной из картинок, то она отобразится в более удобном для просмотра окне, см. рис. 13.1.7.

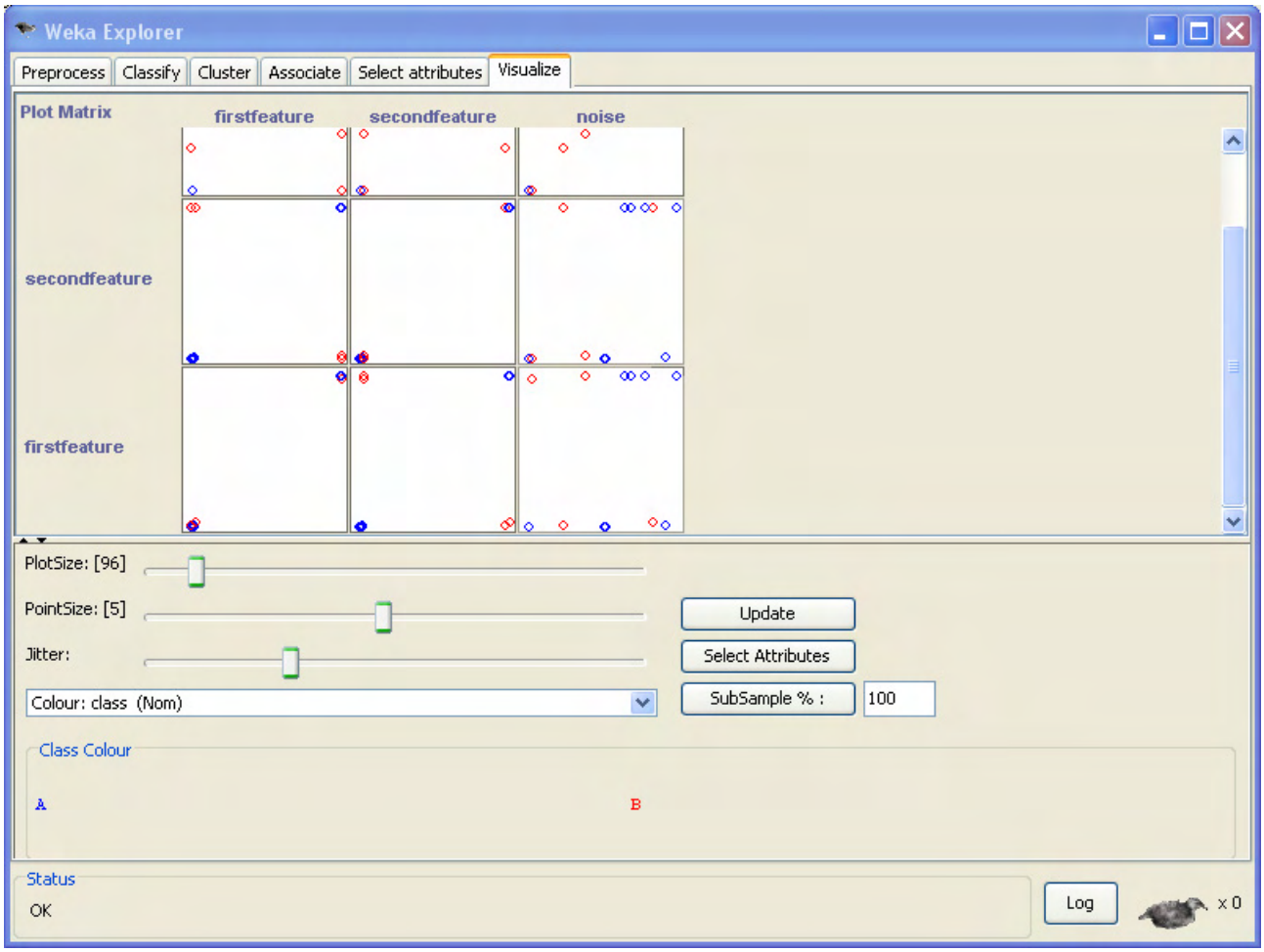


Рис. 13.1.6. Вкладка Visualize модуля Explorer.

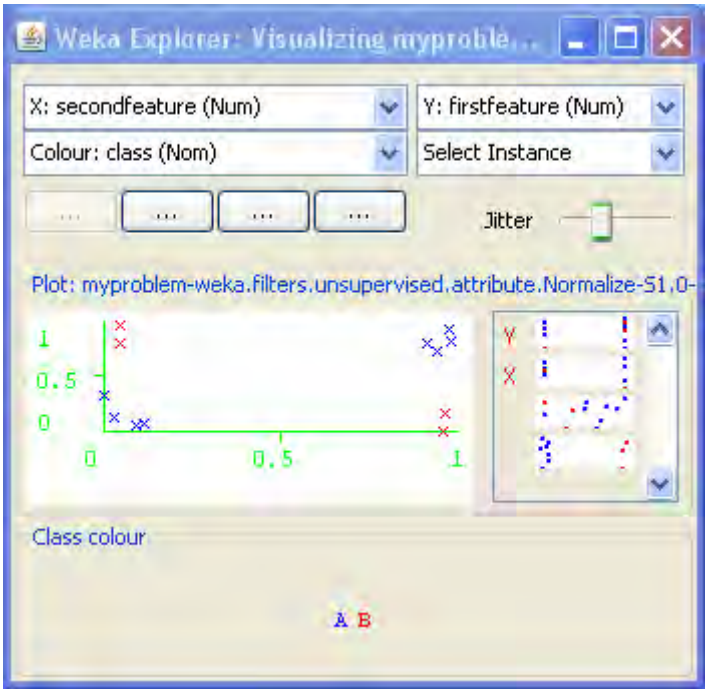


Рис. 13.1.7. Модуль для просмотра пары признаков.

4. Вкладка **Select Attributes** (рис. 13.1.8).

Вкладка **Select Attributes** позволяет выбрать признаки, для последующей классификации (или кластеризации) объектов. На панели **Attribute Evaluator** выбирается функция оценки качества признаков, а на панели **Search Method** – метод поиска оптимального признакового пространства (например генетический алгоритм, случайный поиск или полный перебор, см. главу «Методы глобальной оптимизации[1]»). Схема работы с этими панелями аналогична схеме выбора фильтра или классификатора. На панели **Attribute Selection Mode** выбирается режим селекции: по всей выборке (**use full training set**) или с помощью контроля по блокам (**cross-validation**). Есть компонент для выбора целевого признака (класса) и кнопка **Start** для запуска процедуры селекции признаков, результаты которой отобразятся потом на самой большой панели **Attribute selection output**.

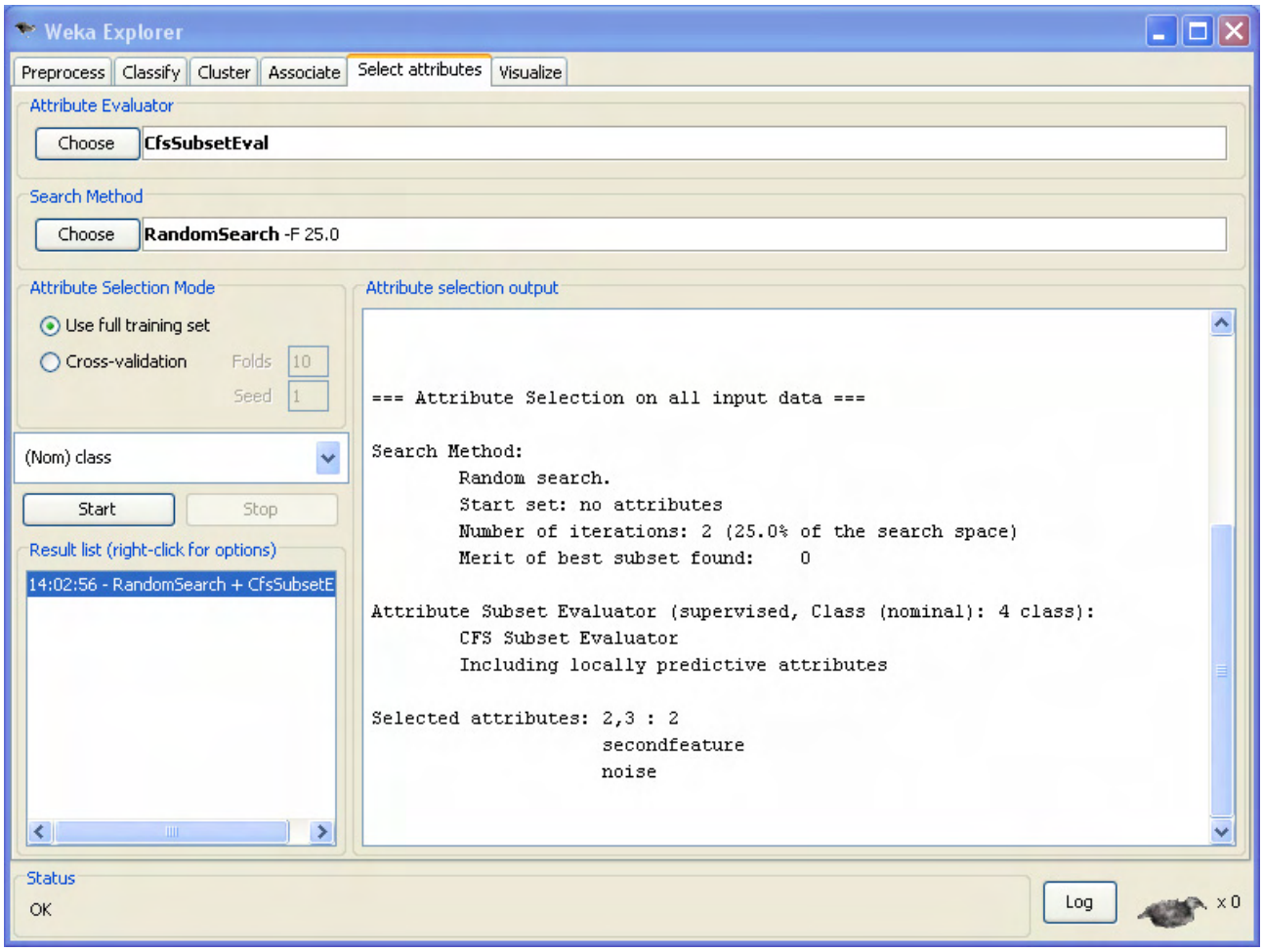


Рис. 13.1.8. Вкладка Select Attributes модуля Explorer (выведен отчёт процедуры селекции признаков).

[1] Глава может быть в этом же учебном пособии или в его первой части (зависит от версии пособия).

5. Вкладки **Claster** и **Associate**.

Работа на вкладках кластеризации и поиска ассоциаций (эту тему мы стараемся не затрагивать в данном учебном пособии) аналогична работе при классификации, а сами вкладки имеют аналогичную структуру (даже более простую).

§13.2. Модуль Experimenter

Этот модуль позволяет проводить эксперименты: запускать несколько алгоритмов на нескольких задачах и получать сводный отчёт.

1. Вкладка **Setup** (рис. 13.2.1).

На вкладке изначально активны две кнопки: **Open** (открыть файл эксперимента) и **New** (создать новый эксперимент), а также две опции списка: **Simple** (простой), **Advanced** (для «продвинутых» пользователей). При дальнейшем описании считаем, что выбрана опция **Simple**.

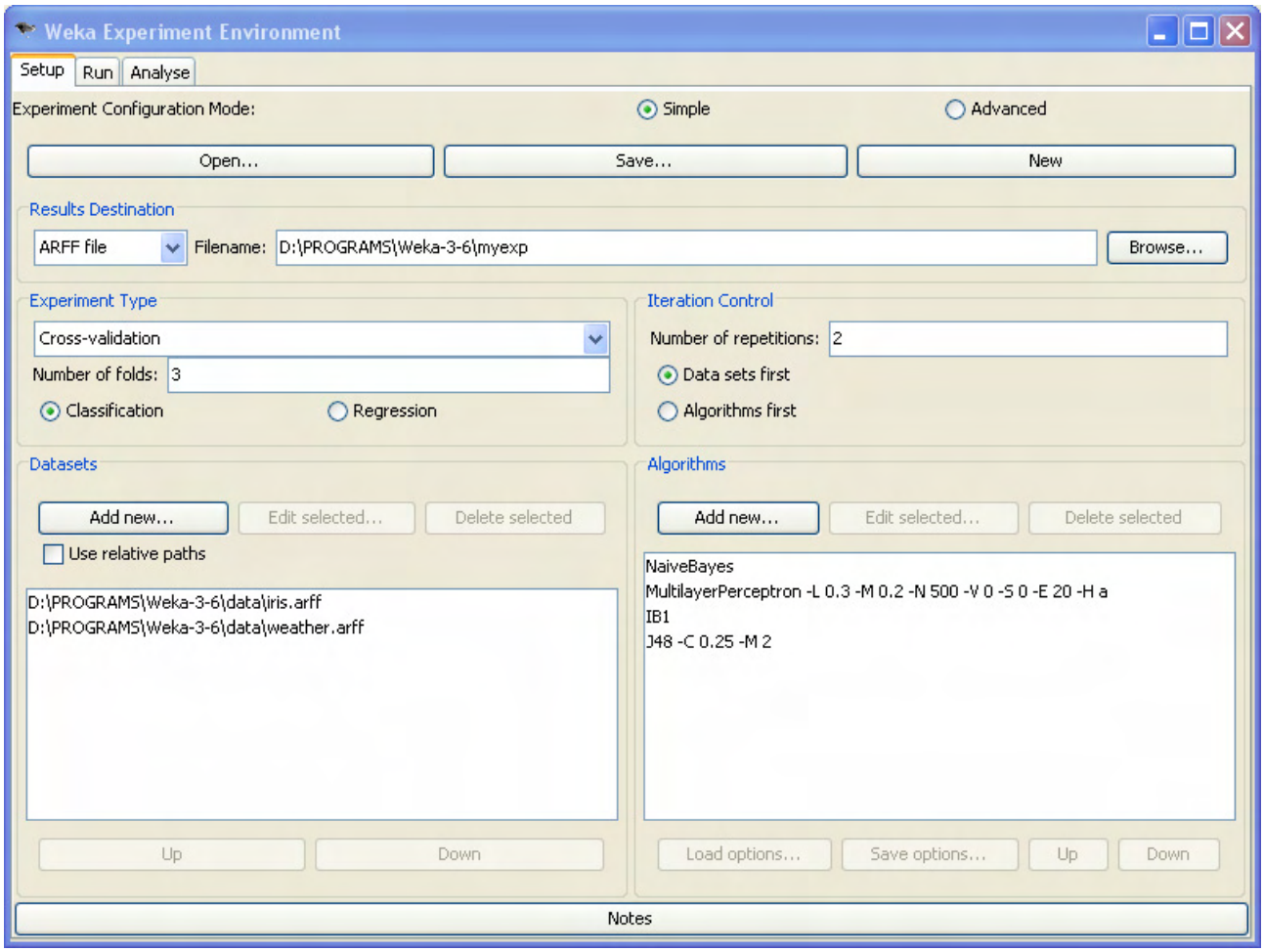


Рис. 13.2.1. Вкладка Setup модуля Experimenter (выбраны 2 задачи, 4 алгоритма, контроль с 3 фолдами и 2 итерациями).

Создадим новый эксперимент. При нажатии на кнопку **New** активируются все опции. Первым делом лучше на панели **Result Destination** выбрать файл для записи отчёта. **Внимание!** Файл, как правило (можно

выбрать и другие форматы), имеет расширение **arff**. Не затрите файлом результатов какой-нибудь файл данных!

На панели **Experiment Type** выбирается тип эксперимента: проводить верификацию методом контроля по блокам/фолдам или использовать разделение выборки на контроль/обучение. **Внимание!** Чем больше фолдов Вы укажете, тем дольше будут идти эксперименты.

На панели **Iteration Control** выбирается число итераций (повторений экспериментов). При повторениях происходят новые разбиения на обучение/контроль и на фолды. Также здесь указывается порядок проведения эксперимента (перебирать сначала все задачи или все алгоритмы). На панели **Datasets** можно выбрать задачи, на панели **Algorithms** – алгоритмы (параметры их выбираются аналогично тому, как это делалось в **Explorere**).

2. Вкладка **Run**.

На этой вкладке только одна «полезная» кнопка **Start**, которую и надо нажать для запуска экспериментов.

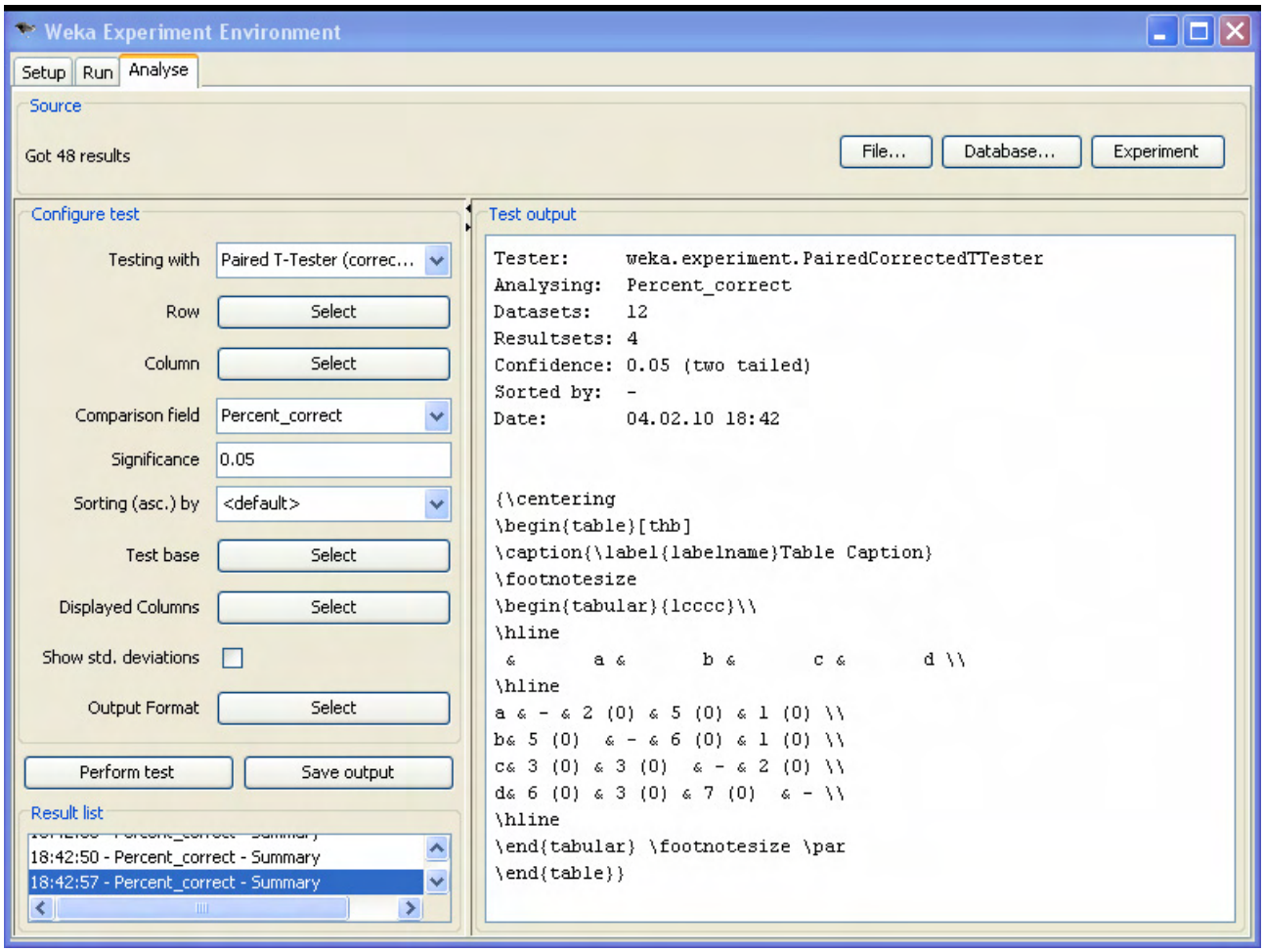


Рис. 13.2.2. Вкладка Analyse модуля Experimenter (выведен TeX-отчёт об экспериментах).

3. Вкладка **Analyse** (рис. 13.2.2).

Эта вкладка позволяет анализировать результаты экспериментов. На панели **Source** выбирается, какой эксперимент анализировать (только что проведённый или записанный в файле).

На панели **Configure test** определяется вид статистики для анализа. Кнопка **Row** позволяет выбрать, что будет записано по строкам выводимой матрицы (для примера выберем из открывшегося списка **Dataset**, **Run**, **Fold**), а кнопка **Column** – что будет записано по столбцам (выберем **Scheme**). В поле **Comparison field** выбирается, чем будет заполнена таблица (выбор **Percent_correct** обеспечивает заполнение процентами верной классификации). Остальные кнопки и опции также позволяют уточнить вид отчёта. Например, при выборе **Output Format/Output Format/LaTeX** будет произведена генерация TeX-файла (по умолчанию выбрана опция **Plain Text**).

Для генерации отчёта нажмите кнопку **Perform test**. Вид отчёта показан ниже

```
Tester:     weka.experiment.PairedCorrectedTTester
Analysing:  Percent_correct
Datasets:   12
Resultsets: 4
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       04.02.10 18:24

Dataset         (1) bayes.Nai | (2) funct (3) lazy.I (4) trees
----------------------------------------------------------------
iris 1 1         (1)    98.00 |   98.00      96.00     96.00
iris 1 2         (1)    92.00 |   92.00      92.00     92.00
iris 1 3         (1)    94.00 |   92.00      96.00     92.00
iris 2 1         (1)    96.00 |   98.00      96.00     88.00
iris 2 2         (1)    96.00 |   98.00      96.00     98.00
iris 2 3         (1)    96.00 |   92.00      94.00     96.00
weather 1 1      (1)    60.00 |   60.00     100.00     60.00
weather 1 2      (1)   100.00 |   80.00      80.00     60.00
weather 1 3      (1)    75.00 |   75.00     100.00     75.00
weather 2 1      (1)    80.00 |   60.00     100.00     60.00
weather 2 2      (1)    80.00 |   60.00     100.00     60.00
weather 2 3      (1)    50.00 |   50.00      50.00     50.00
----------------------------------------------------------------
Average                 84.75 |   79.58      91.67     77.25
----------------------------------------------------------------
                      (v/ /*) |  (0/12/0)   (0/12/0)  (0/12/0)

Key:
(1) bayes.NaiveBayes
(2) functions.MultilayerPerceptron
(3) lazy.IB1
(4) trees.J48
```

Здесь показана статистика работы на двух задачах **iris** и **weather** (их данные поставляются вместе с программой) четырёх алгоритмов: наивного байесовского классификатора (**bayes.NaiveBayes**), многослойного персептрона (**functions.MultilayerPerceptron**), ближайшего соседа (**lazy.IB1**) и одного из алгоритмов построения решающего дерева (**trees.J48**). Запись «**weather 1 3**» означает, что в соответствующей строке собрана статистика работы на задаче **weather** в первом запуске на третьем фолде, поскольку кнопкой **Row** были выбраны пункты **Dataset**, **Run**, **Fold**. Строка **Average** выведется, если была активирована опция **Output Format/Show Average**.

§13.3. Остальные модули и функции программы WEKA

Модуль **KnowledgeFlow** позволяет представить весь процесс решения задачи в виде графа: указать этапы решения в виде вершин, соединить их дугами (направленными рёбрами), а затем запустить этот процесс на исполнение. В последнее время подобные модули становятся очень популярными[1]. Для изучения работы с модулем лучше посмотреть соответствующую главу файла помощи [WekaManual-3-7-1.pdf], в которой на примере показан сценарий работы с модулем.

Модуль **Simple CLI** просто предоставляет интерфейс текстовой строки для ввода команд. При использовании командной строки у пользователя появляются возможности, которых не было в других модулях (см. подробнее в [WekaManual-3-7-1.pdf]), кроме того, экономится память. Рассмотрим пример классификации с помощью командной строки:

```
> java weka.classifiers.trees.J48 -t data\weather.arff -p 0

=== Predictions under cross-validation ===

 inst#     actual  predicted error prediction
     1       2:no      1:yes   +   0.75
     2      1:yes       2:no   +   0.75
     1       2:no      1:yes   +   0.75
     2      1:yes      1:yes       1
     1       2:no       2:no       1
     2      1:yes      1:yes       1
     1       2:no      1:yes   +   1
     2      1:yes      1:yes       1
     1       2:no       2:no       1
     1      1:yes       2:no   +   0.75
     1      1:yes      1:yes       1
     1      1:yes      1:yes       1
```

[1] В виде графа представлен процесс анализа и обработки данных в системе «RapidMiner», которой посвящена отдельная глава данного учебного пособия.

```
     1      1:yes      1:yes       1
     1      1:yes      1:yes       1
```

Перечислим некоторые классификаторы, которые можно указывать в модуле **Simple CLI** (полный перечень см. в файлах справки программы):
`trees.J48` – разновидность деревьев решений с алгоритмом C4.5,
`bayes.NaiveBayes` – наивный Байес (ключ **`-K`** – применять непараметрическое восстановление плотности для численных признаков),
`functions.SMO` – метод SVM (по умолчанию с линейным ядром, для полиномиального ядра степени 5 с коэффициентом 10 использовать **`-E 5 –C 10`**),
`lazy.IBk` – алгоритм kNN.

Перечислим также некоторые ключи, которые указываются при вызове классификатора:
`-t` – файл с обучающей выборкой (в формате arff),
`-T` – файл с тестовой выборкой (если параметра нет, то будет выполнен контроль по блокам – с 10 блоками по умолчанию),
`-x` – число блоков (фолдов) в контроле по блокам,
`-c` – устанавливает целевой признак (класс),
`-d` – сохраняет модель (тип алгоритма и параметры, настроенные по всей обучающей выборке),
`-l` – загружает модель,
`-p #` – показывает классификацию и указанное число признаков (если указан тестовый файл),
`-i` – детальное описание качества классификации,
`-o` – отключает вывод информации о модели.

При появлении ошибки **Out of Memory** следует увеличить область динамически распределяемой памяти (heap) для Java с помощью ключа

`-Xmx1024m`

(для 1 Гб)[1].

В вызове

```
java -Xmx1024m weka.classifiers.trees.J48 -t data.arff -i -k \
-d J48-data.model >&! J48-data.out &
```

увеличивается heap, запускается классификатор «решающее дерево», выводятся значения **precision** и **recall** для всех классов (**`-i`** и **`-k`**), сохраняется модель (**`-d`**). Символ бэкслэш («\») указывает, что строка переносится (команда не завершена).

[1] Дополнительную информацию можно найти по адресу http://java.sun.com/docs/hotspot/VMOptions.html.

В меню окна **Weka GUI Chooser** есть несколько полезных функций. На вкладке **Tools** размещён редактор arff-файлов (**ArffViewer**), простой модуль просмотра баз данных (**SqlViewer**) и программа для построения и обучения байесовских сетей (**Bayes net editor**). На вкладке **Visualization** различные средства визуализации: визуализация данных, которая есть и в Explorere (**Plot**), визуализация сохранённых ROC-кривых (**ROC**), визуализация деревьев решений (**TreeVisualizer**), визуализация графов формата XML BIF или DOT (**GraphVisualizer**) и визуализация поверхностей решения (**BoundaryVisualizer**), о которой мы расскажем подробнее.

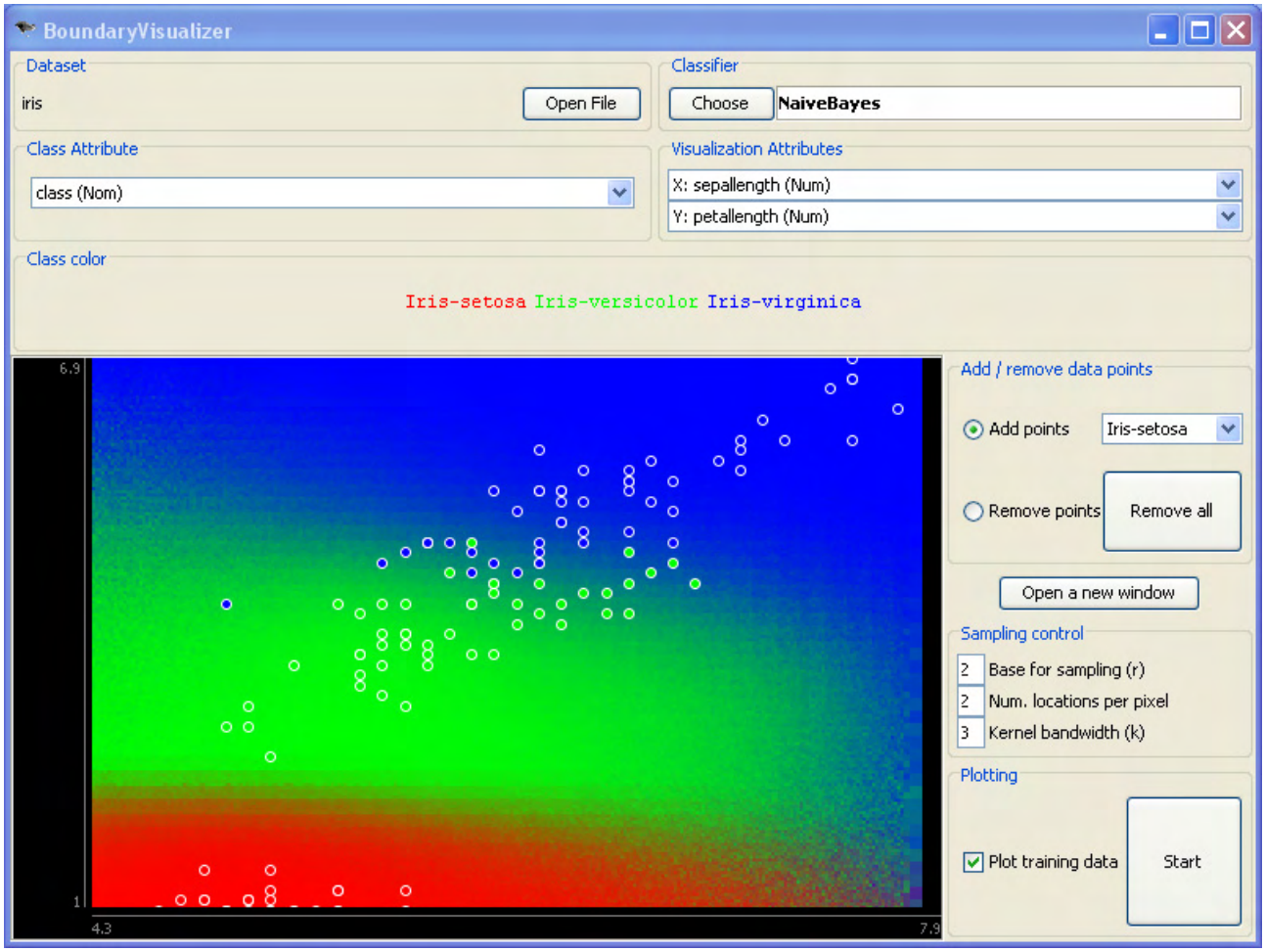


Рис. 13.3.1. Визуализация разделяющих поверхностей метода наивного Байеса в известной задаче «Ирисы».

Функция **Visualization/BoundaryVisualizer** позволяет посмотреть на разделяющую поверхность классификатора (естественно, в проекции на пару признаков). При выборе данной опции появляется окно **BoundaryVisualizer** (рис. 13.3.1), в котором с помощью кнопки **Open File** можно загрузить данные, на вкладке **Class Attribute** выбрать целевой признак (класс), на вкладке **Visualization Attributes** – пару признаков, которые определяют плоскость визуализации, на вкладке **Classifier** – классификатор. При нажатии на кнопку **Start** начинается процедура визуализации (последовательно уточняется

классификация точек пространства). Результат в виде закрашенных областей выводится на экран, см. рис. 13.3.1.

Напоследок опишем решение часто возникающих проблем с запуском программы WEKA и выдачей сообщений типа **classes are not found**. Проблемы могут быть вызваны неверно настроенной переменной среды CLASSPATH. Эта переменная подсказывает Java, где искать классы, поэтому должна включать путь к файлу weka.jar. В DOS это можно исправить командой типа

```
set CLASSPATH=c:\weka-3-6\weka.jar;%CLASSPATH%,
```

а в UNIX/Linux-е – командой типа

```
export CLASSPATH=/home/weka/weka.jar:$CLASSPATH
```

В ОС Windows XP/Vista надо кликнуть правой кнопкой мыши на ярлык **Мой компьютер** (My Computer) и выбрать **Свойства** (Properties). На вкладке **Дополнительно** (Avanced) нажать на кнопку **Переменные среды** (Environment Variables). Создать новую переменную с именем CLASSPATH и значением, которое указывает путь к файлу weka.jar.

Подробный файл справки [WekaManual-3-7-1.pdf] поставляется вместе с программой (или может быть скачен с сайта [Weka]). Кроме того, постоянно поддерживается справочный Wiki-ресурс [Weka Wiki]. Отметим, что библиотеки программы WEKA можно использовать другими программами (что многие и делают), подробности узнавайте на перечисленных выше ресурсах.

Глава 14. ПРОГРАММА ДЛЯ АНАЛИЗА ДАННЫХ RAPIDMINER

§14.1. Возможности программы RapidMiner

Программа **RapidMiner** (первое название «Yale») является средой для машинного обучения и анализа данных, в которой пользователь ограждён от всей «черновой работы» [RM KDD, 2006]. Вместо этого ему предлагается «нарисовать» весь желаемый процесс анализа данных в виде **цепочки** (графа) **операторов** и запустить его на выполнение. Цепочка операторов представляется в RapidMinere в виде интерактивного графа и в виде выражения на языке XML (eXtensible Markup Language, основного языка системы). Система написана на языке Java и распространяется под лицензией AGPL version 3. Ко всем основным функциям имеется доступ через Java API и версию программы для командной строки (а не только через общий пользовательский интерфейс).

Сейчас в системе реализовано более 400 операторов. Из них

1. Операторы **обучения по прецедентам** (machine learning algorithms), в которых реализованы алгоритмы классификации, регрессии, кластеризации и поиска ассоциаций, а также мета-алгоритмы (типа бустинга).

2. Операторы **системы WEKA**.

3. Операторы **предобработки** (дискритизация, фильтрация, заполнение пропусков, уменьшение размерности и т.д.)

4. Операторы **работы с признаками** (селекция и генерация признаков).

5. **Мета-операторы** (например, оператор оптимизации по нескольким параметрам).

6. Операторы **оценки качества** (скользящий контроль и т.д.).

7. Операторы **визуализации** (это «конёк» системы, поскольку способов визуализации достаточно много и все графики смотрятся очень эффектно).

8. Операторы **загрузки и сохранения данных** (включая работу со специальными форматами: arff, C4.5, csv, bibtex, базы данных и т.д.).

Имеются также богатые возможности по добавлению в систему своих операторов.

Последняя версия системы представлена для свободного скачивания на сайте [RM]. Система поставляется в виде инсталлятора для Windows или в виде Java-версии (для последней необходимо ПО Java Runtime Environment, version

5.0). Запуск программы осуществляется кликом мыши по ярлыку (Windows-версия) или по файлу **rapidminer.jar** (набором **`java -jar rapidminer.jar`**). Можно запустить версию программы для командной строки

`scripts/rapidminer.bat`

или с пользовательским интерфейсом

`scripts/RapidMinerGUI.bat` (в Windows).

Для установки плагинов достаточно скопировать их в папку **`lib/plugins`** (если нет специального инсталлятора).

Начало работы

При запуске RapidMiner предложит одну из следующих возможностей:

1. Начать новое определение процесса (New).
2. Открыть существующее (Open recent).
3. Открыть (Open).
4. Открыть шаблон (Open Template).
5. Запустить demo (Online Tutorial).

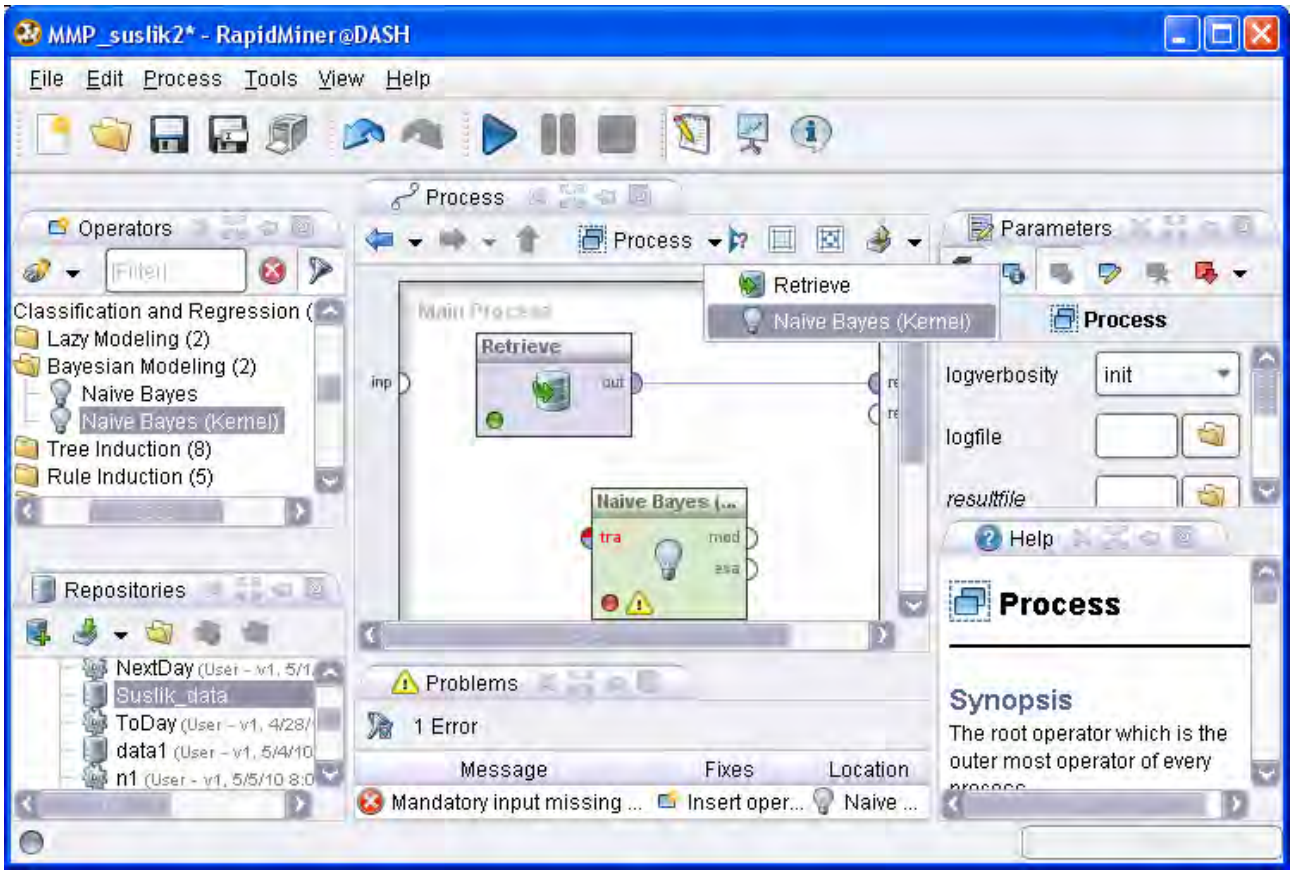


Рис. 14.1.1. Основное окно программы RapidMiner.

При выборе опции «**New**» появляется окно **Repository Browser**, в котором надо выбрать название файла для хранения определяемого **процесса обработки данных**. Далее появится основное окно программы (см. рис. 14.1.1). На

верхней панели справа[1] находятся три кнопки (см. рис. 14.1.2), которые позволяют переключаться между тремя основными режимами: построение (клавиша **F8**), просмотр результатов (**F9**) и переход к приветствию (выбору пяти опций, см. выше). Также переключение возможно через меню: **View/Perspectives/**.

Рис. 14.1.2. Кнопки переключения между режимами работы.

Через опцию меню **View/Show View** можно выбрать видимые панели, которые можно расположить на основном окне по своему вкусу. **Внимание!** Не всегда по умолчанию выбраны панели, которые нужны для работы! В частности, пользователю нужны панели **Process**, **Operators**. Перечислим панели программы:

1. **Process.** На этой панели пользователь «рисует» основной процесс, перетаскивая на неё операторы и соединяя их связями. Многие полезные функции вызываются через контекстное меню (щелчок по панели правой кнопкой мыши).
2. **Overview.** Отображается «нарисованный» процесс (панель нужна для навигации по панели Process, на которой весь граф может не помещаться, поэтому отображается только та часть, которая выделена на панели Overview).
3. **Operators.** Здесь в системе каталогов «хранятся» операторы, которые можно перетаскивать на панель Process. При выборе оператора на панели Help отображается основная информация.
4. **Tree.** Представление нарисованного процесса в виде дерева.
5. **XML.** Представление нарисованного процесса в виде XML-кода.
6. **Parameters.** При выделении оператора на панели Process здесь отображаются его параметры. Внесение изменений в параметры часто необходимо для нормальной работы (в частности, оператору загрузки данных необходимо указать имя файла с данными).
7. **Problems.** Здесь отображаются ошибки, которые не позволяют запустить процесс.
8. **Repositories.** На этой панели осуществляется навигация по файлам процессов и данных.
9. **Context.** Определение входа и выхода процесса.
10. **Help.** Отображение необходимой информации.
11. **Comment.** Панель «для пометок».
12. **Log.** Журнал событий.
13. **Remote Processes.** Работа с «удалёнными процессами» (для начинающего пользователя данная панель не нужна).

[1] Их положение может быть и другим (зависит от версии программы и конфигурации).

14. **System Monitor.** Отображаются системные ресурсы (используемая память).
15. **Result Overview.** Отображение результатов работы процесса. В режиме **Results** создаются также дополнительные панели для отображения графики.

§14.2. Загрузка и визуализация данных

Отметим сразу, что RapidMiner поддерживает не все «желаемые» форматы. Для загрузки данных можно

1. Перетащить файл мышкой на панель **Repositories**.
2. Выбрать в меню **File/Import Data.**
3. На панели операторов выбрать подходящий оператор.

Последний вариант самый универсальный. На панели **Operators** (рис. 14.2.1) изображено дерево операторов, каждый узел помечается числом (количество операторов в потомках). Выберем **Import / Data / Read ARFF**. Для этого можно кликнуть по нему два раза или перетащить его на панель **Process**.

Рис. 14.2.1. Панель Operators.

Панель **Process** является основной при работе. При выборе оператора он появляется на ней и автоматически включается в граф процесса (например, на рис. 14.2.2 появилась дуга из выхода оператора в узел результата).

Каждый оператор изображён пиктограммкой, с левой стороны которой расположены узлы входа (если есть), а с правой – узлы выхода (если есть). Выходы одного оператора можно соединять с входами другого, тогда результат его работы будет поступать на вход другому оператору. У общего поля **Main Process** (основной процесс) есть также узлы входа и выхода. В частности, выход оператора **Read ARFF** автоматически соединился с общим выходом основного процесса. **Внимание!** Для визуализации результатов работы операторов надо, чтобы соответствующие операторы были соединены с выходами основного процесса. В левом нижнем углу пиктограммы каждого оператора (на панели Process) есть индикатор состояния:

зелёный цвет – оператор отработал,
жёлтый цвет – оператор готов к работе,
красный – оператор не готов к работе (есть ошибки).

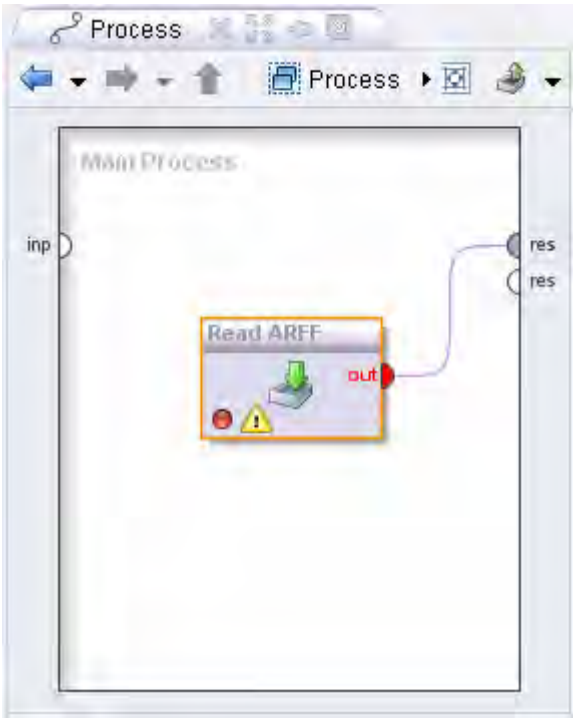


Рис. 14.2.2. Оператор Read ARFF на панели Process.

В нашем случае (рис. 14.2.2) оператор не готов к работе, поскольку не указан файл с данными. Если щёлкнуть по пиктограммке оператора, то на панели **Parameters** отобразятся его параметры (см. рис. 14.2.3). В частности, у оператора **Read ARFF** есть параметр **data file**, который надо заполнить.

Рис. 14.2.3. Панель Parameters.

Исправлять ошибки можно также пользуясь панелью **Problems** (рис. 14.2.4), на которой они перечислены. В списке три графы: описание проблемы (Message), способ устранения (Fixes) и источник (Location). Щёлкнув два раза

левой кнопкой мыши по полю Fixes, переходим к исправлению ошибки (можно также щёлкнуть правой кнопкой и выбрать нужный пункт контекстного меню).

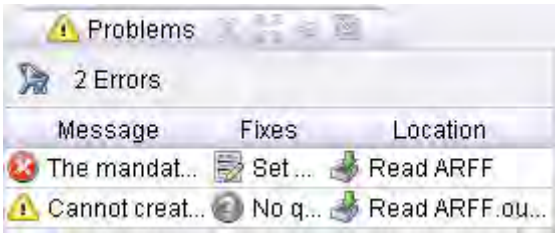


Рис. 14.2.4. Панель Problems.

Основной процесс запускается кнопкой Run  на верхней панели (клавиша **F11**). После окончания работы программа попросит разрешения переключиться в режим просмотра результатов **Results**.

На панели **Result Overview** приведён список результатов процессов. Щелчок по соответствующему пункту сворачивает или разворачивает отчёт (см. рис. 14.2.5).

Рис. 14.2.5. Панель Result Overview.

В нашем примере с процессом **Read ARFF** на панели **Example Set** (см. рис. 14.2.6) можно увидеть загруженные данные. Доступны три режима просмотра: **Meta Data View** (общая статистика), **Data View** (в виде таблицы) и **Plot View** (графическая визуализация).

Рис. 14.2.6. Панель Example Set.

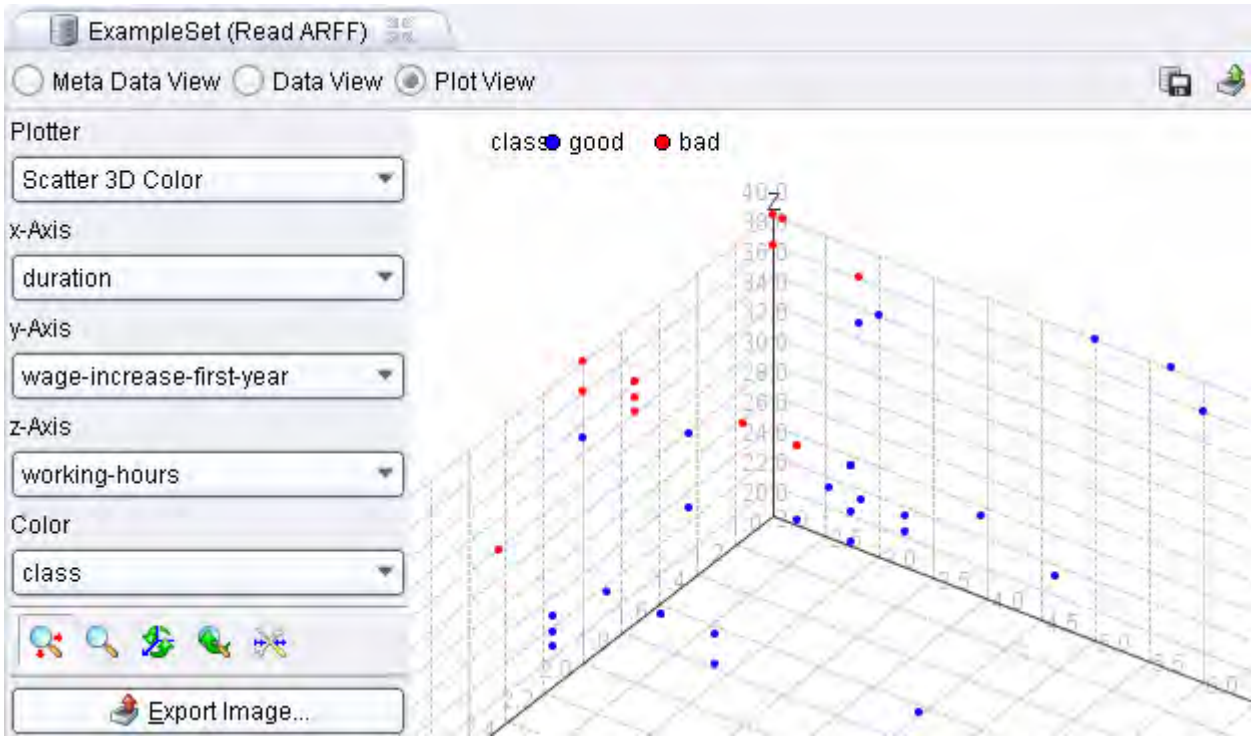


Рис. 14.2.7. Визуализация на панели Example Set (режим Scatter 3D Color).

В системе RapidMiner реализовано огромное число различных способов визуализации данных. Способ визуализации выбирается на панели **Example Set** в режиме **Plot View** кнопкой с пометкой **Plotter**. Перечислим некоторые виды визуализации, реализованные в системе:

Scatter – обычные 2х и 3х мерные проекции точек (см. рис. 14.2.7),

Bubble – каждый объект изображается кругом, радиус которого соответствует значению одного из признаков,

Parallels – каждый объект изображается линией (которая проходит через точки, соответствующие значениям признаков),

Series – изображение рядов,

SOM – карты Кохонена (см. рис. 14.2.8)

Block – объекты изображаются прямоугольниками,

Density – изображение плотности (см. рис. 14.2.9),

Pie, Ring – секторные диаграммы (типа «пирог»),

Bars – гистограммы,
Surface – поверхность,
и т.д.

Изображения можно увеличивать, перемещать, менять масштаб осей (необходимые кнопки расположены в левой нижней части панели). Нажатие на кнопку **Export Image** позволяет сохранить изображение на диск (поддерживаемые форматы: eps, swf, pdf, ps, svg, emf, gif, raw, ppm, bmp, jpg, png, gif и т.д.)

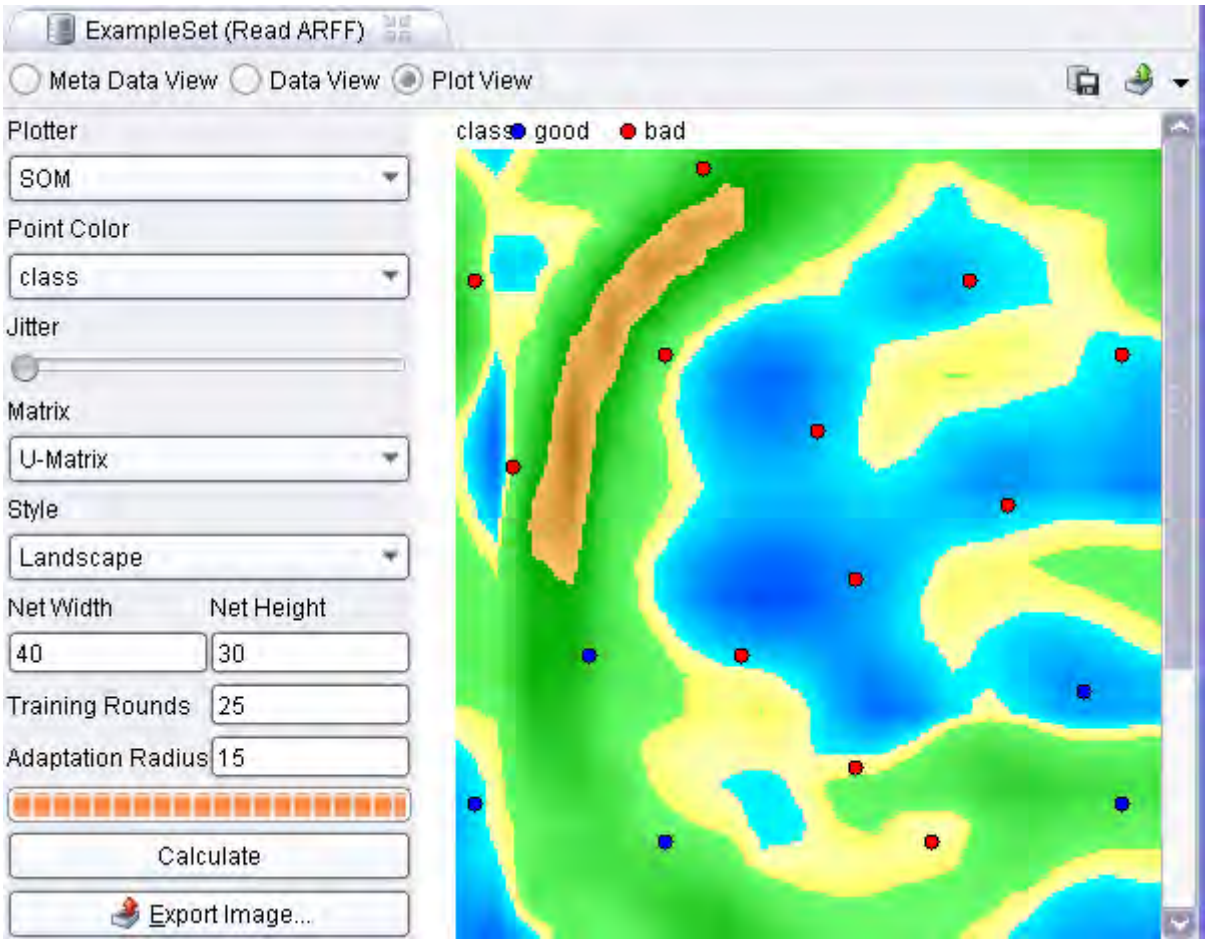


Рис. 14.2.8. Построение карты Кохонена в программе RapidMiner.

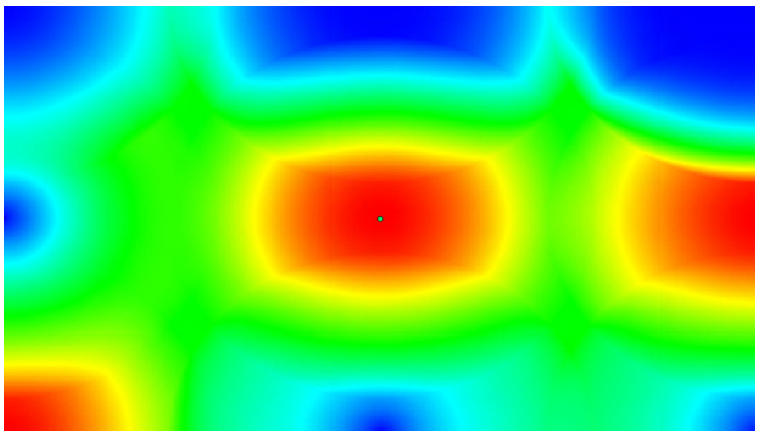


Рис. 14.2.9. Изображение типа Density.

§14.3. Решение задач классификации в системе RapidMiner

Для иллюстрации работы классификаторов перетащим с панели **Operators** на панель **Process** какой-нибудь классификатор. Например, байесовский (расположение **Modeling/Classification and Regression/Bayesian Modeling**). Выход оператора загрузки данных соединим с входом оператора классификации, а выход оператора классификации с выходом основного процесса (рис. 14.3.1). Заметим, что у оператора **Naïve Bayes** два выхода: **model** (описание классификатора, его надо соединить) и **example set** (выборка, при желании её можно передать на вход другому классификатору). Оператор классификации по-прежнему не готов к работе (горит красный индикатор), поскольку не указан целевой признак. Щёлкнув по полю **Fixes** в строке этой ошибки на панели **Problems**, выбираем нужный признак. В итоге на панели **Process** появляется новый оператор **Set Role**, в параметрах которого прописан целевой признак, см. рис. 14.3.2.

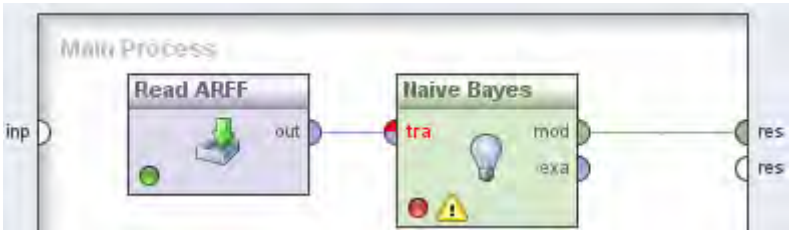


Рис. 14.3.1. Цепочка «загрузка данных» – «байесовский классификатор».

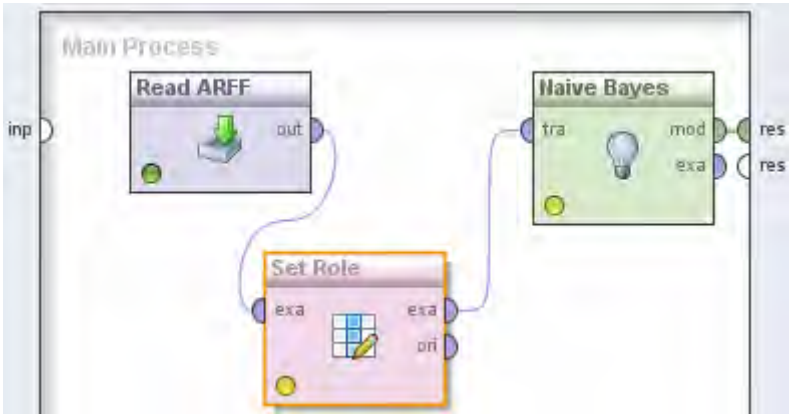


Рис. 14.3.2. Цепочка «загрузка данных» – «назначение класса» – «байесовский классификатор».

Запуск процесса приводит к построению байесовского классификатора, см. рис. 14.3.3.

Заметим, что мы построили классификатор, но пока «не запустили» его на классификацию. Покажем, как организовать процедуру контроля по блокам (фолдам). Для этого к процессу загрузки данных добавляем оператор **Evaluation/Validation/X-Validation**. В принципе, на панель основного процесса больше на надо добавлять операторы (остальные операторы добавляются «внутрь оператора Validation»), поэтому на панели основного процесса оставим только оператор загрузки данных и оператор **Validation.** Выход оператора **Read**

ARFF соединяется с входом оператора **Validation**, ave-выход которого соединяется с выходом основного процесса (рис. 14.3.4).

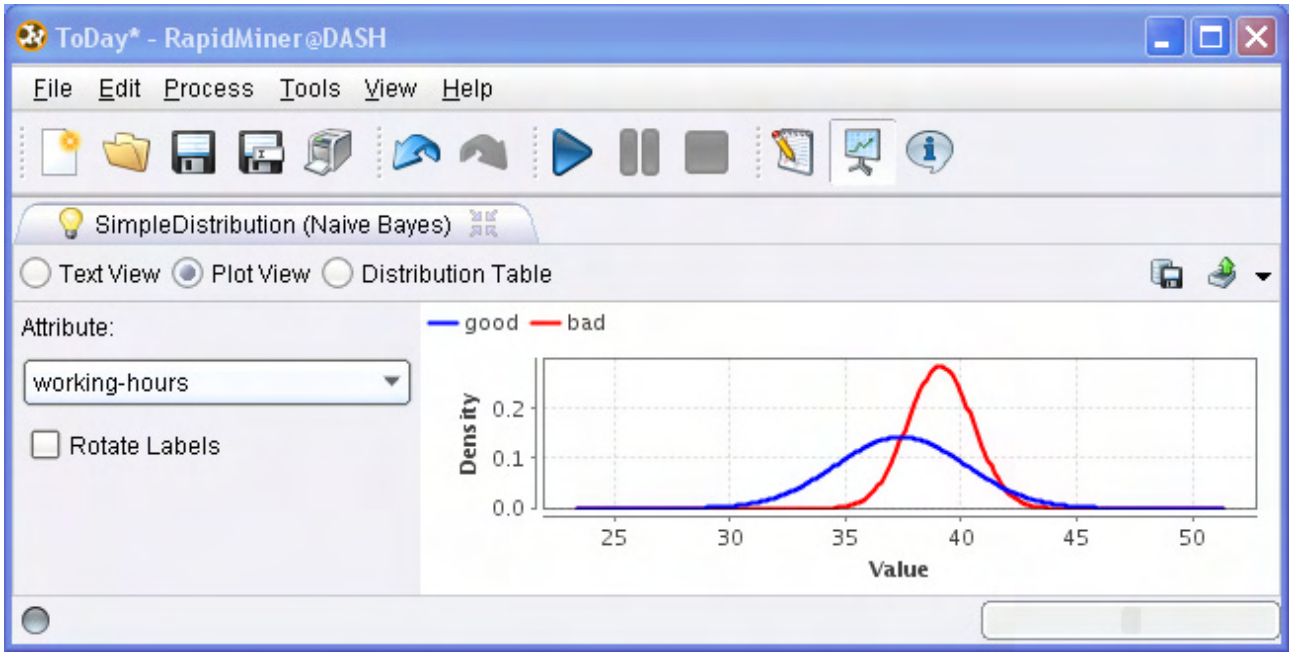


Рис. 14.3.3. Построение байесовского классификатора.

Пиктограммка добавленного оператора **Validation** имеет в правом нижнем углу пометку «два синих окошка», которая означает, что этот оператор можно «открыть» и «наполнить содержимым». Щёлкнем два раза мышкой по пиктограммке (**внимание!** Это нетривиальный ход.). Откроются два поля: обучение (training) и тестирование (testing), см. рис. 14.3.5. В первое окно перетащим оператор классификации, например нейросеть (**Modeling/Classification and Regression/Neural Net Training/Neural Net**), а во второе – оператор оценки качества классификации, например **Performance Measurement/Performance**. Остальные нужные операторы можно добавить, исправляя ошибки. Обязательно понадобятся операторы **Set Role** для пометки целевых признаков и оператор **Apply Model** для запуска построенного классификатора. Кроме того, некоторые классификаторы требуют предварительной обработки данных: устранения пропусков, перевод номинальных признаков в числовые и т.д.

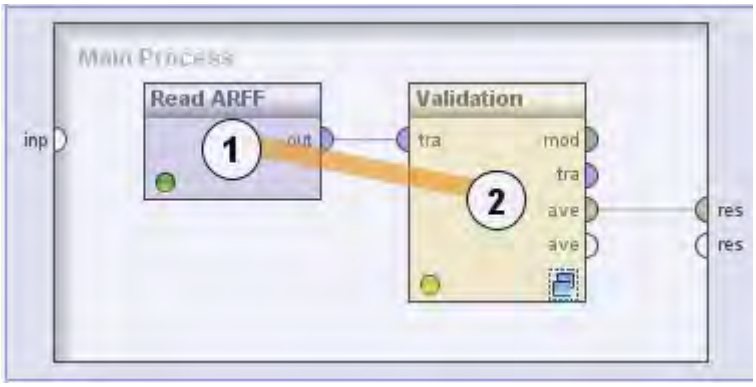


Рис. 14.3.4. Организация контроля по блокам (фолдам).

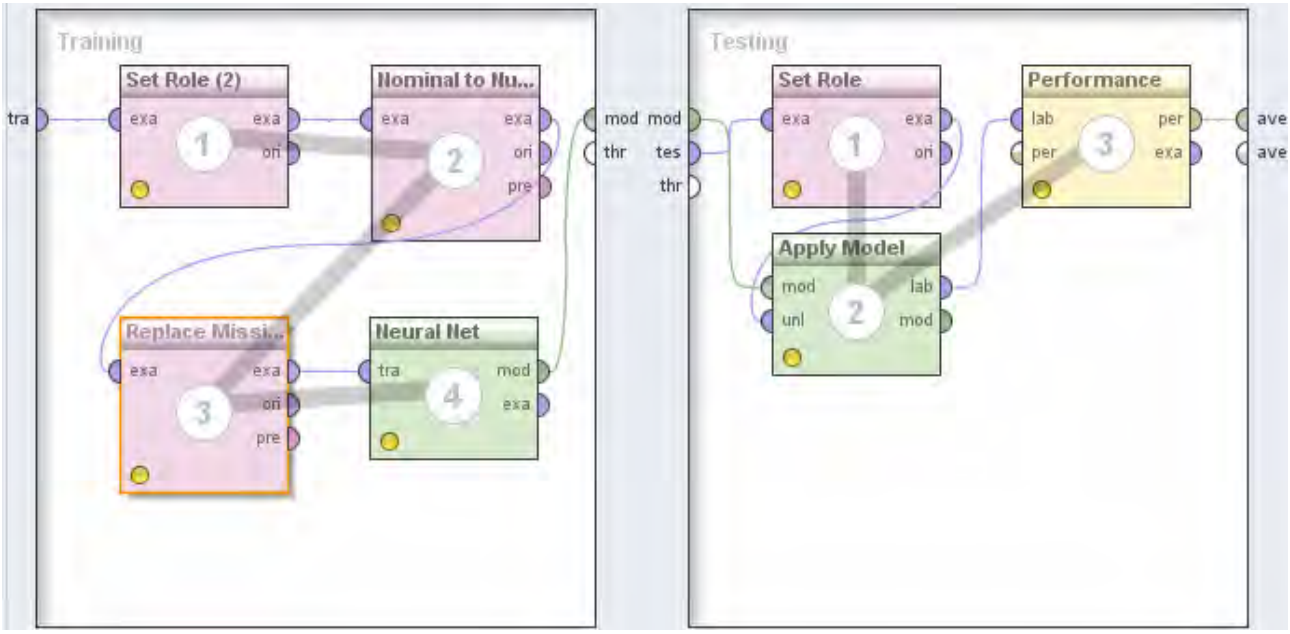


Рис. 14.3.5. «Заполнение» оператора Validation содержимым.

Обратим внимание на важный оператор **Apply Model** (применение модели). Операторы классификаторов имеют выход **mod**, на который поступает полученная модель – настроенный в результате обучения классификатор[1]. Оператор применения модели позволяет этот классификатор запустить для классификации конкретных данных.

На панели **Parameters** можно редактировать параметры каждого оператора (надо сначала его выделить на панели **Process**). В частности, можно менять параметры классификатора. Можно посмотреть описание и основные свойства классификатора. Можно изменить число блоков (фолдов) в параметрах оператора Validation. Нажав на кнопу RUN (или **F11**), запускаем процедуру контроля по блокам.

В системе RapidMiner реализована очень удобная процедура замены операторов. Например, пусть вместо контроля по блокам необходимо организовать другую процедуру контроля качества. Щёлкаем правой кнопкой мыши по пиктограммке оператора **X-Validation**, выбираем опцию **Replace Operator** и название оператора, которым мы хотим заменить наш оператор контроля по блокам, например **Split Validation** (разбиение выборки на обучение и контроль). При этом начинка оператора сохранится! Фактически изменится только его название: вместо **X-Validaton** – **Split Validation**, т.е. новый оператор не надо будет наполнять содержимым… оно достанется ему от оператора, замещённого им.

---

[1] Если навести курсор мыши на выход (или вход) оператора, то появится подсказка, в которой указаны свойства этого выхода (входа). Для просмотра их в отдельном окне надо нажать **F3**.

§14.4. Решение задач кластеризации в системе RapidMiner

Покажем на примере решение задачи кластеризации. Пусть имеется текстовый файл с данными вида

```
0.29 0.37
0.39 0.30
0.50 0.85
0.72 0.76
```
и т.д.

Для загрузки такого файла подойдёт оператор **Import/Data/Read CSV**. Заметим, что файл не удастся перетащить мышкой в директорию репозитория (панель **Repositories**), поскольку эта операция не поддерживается для текстовых файлов. Обойти это препятствие можно следующим образом:

1) открыть файл в программе Excel,
2) сохранить как «Книга Microsoft Excel»,
3) перетащить созданный файл на панель **Repositories** в нужный каталог.

При третьем действии откроется помощник переноса файлов (рис. 14.4.1), который, в частности, позволяет поставить некоторым признакам метки (метки идентификатора, целевого, обычного признака и т.д.)

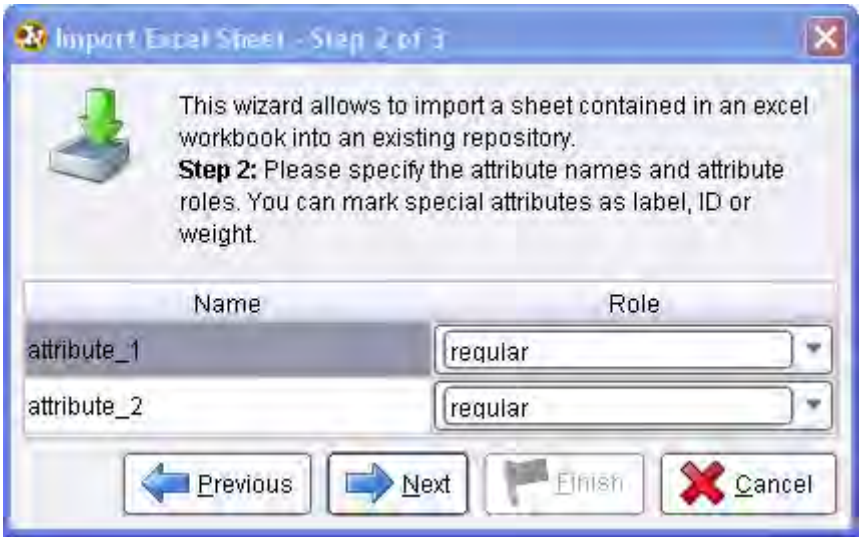


Рис. 14.4.1 Помощник переноса файлов.

Созданный файл можно перенести мышкой на панель **Process** (из репозитория), при этом автоматически создастся оператор **Retrieve**, который загружает нужные данные (рис. 14.4.2).

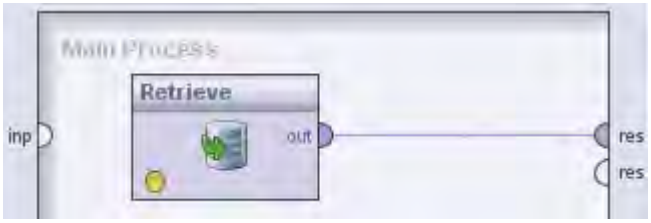


Рис. 14.4.2. Оператор Retrieve.

Перетащим несколько операторов кластеризации на панель **Process**. Выбрав на панели **Process** опцию **Rewire Operators (Recursively)**, получаем автоматическое связывание операторов (рис. 14.4.3). Это полезная опция: часто с помощью неё операторы связываются «правильно» и пользователь освобождается от «лишней» работы. Обратите внимание, что каждый оператор (кластеризации) должен подавать свой выход на выход основного процесса (иначе нельзя будет посмотреть результат – доступно только то, что подаётся на выход). В данном случае связывание «оказалось не совсем правильным»: на выход подаётся описание настроенной модели кластеризации, но не сама выборка для визуализации – она подаётся на вход следующему оператору. Если вручную указать, что выход какого-то оператора должен подаваться ещё куда-то (соединив мышкой выход с нужным входом), то система запросит, что ей делать со старой связью: разорвать или разветвить выход (рис. 14.4.4). При выборе «разветвить» создаётся специальный оператор разветвления **Multiply**.

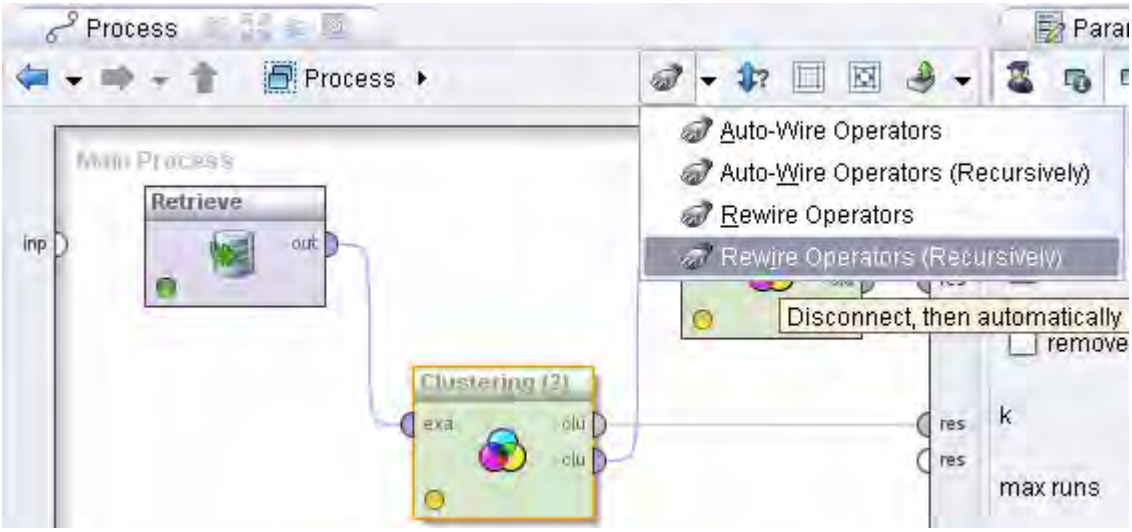


Рис. 14.4.3. Автоматическое соединение операторов.

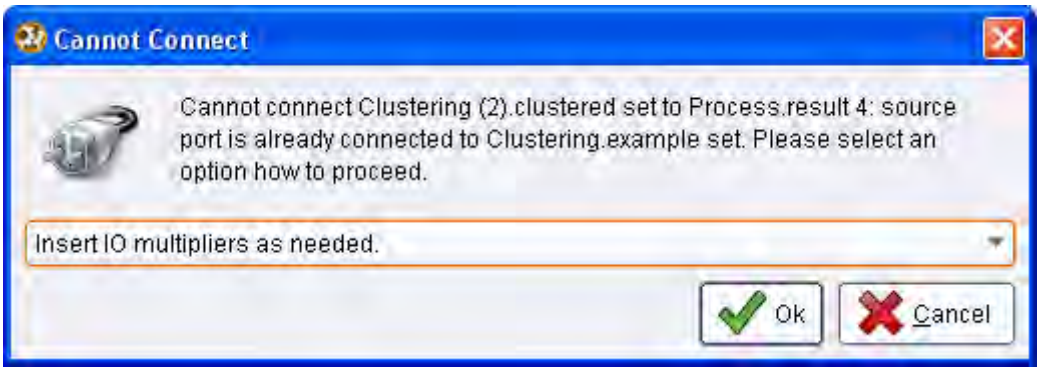


Рис. 14.4.4. Разветвление связей.

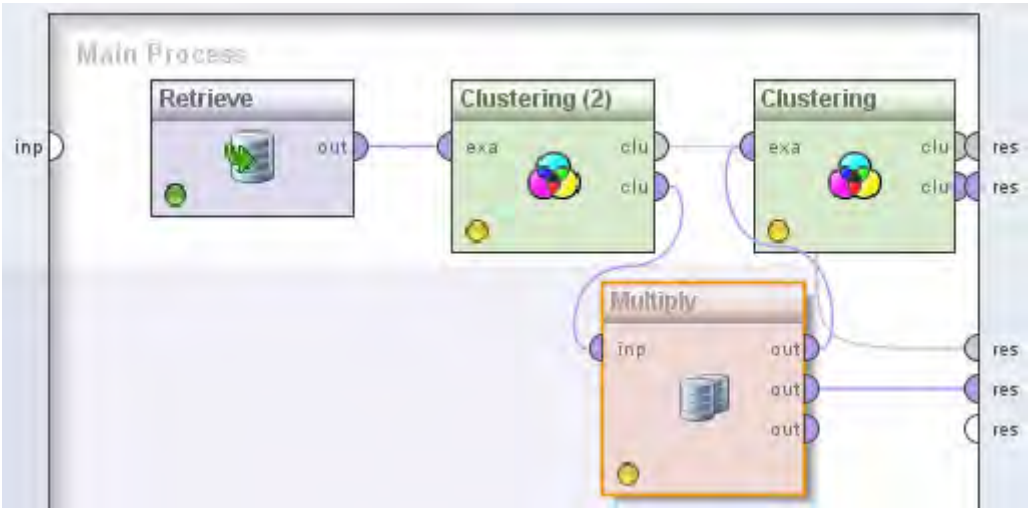


Рис. 14.4.5. Цепочка операторов кластеризации.

В результате создана цепочка операторов кластеризации. Запустив её на исполнение кнопкой **RUN**, получаем результат кластеризации: описание обученной модели – на панели **Cluster Model**, визуализацию кластеризации – на панели **Example Set** (рис. 14.4.6).

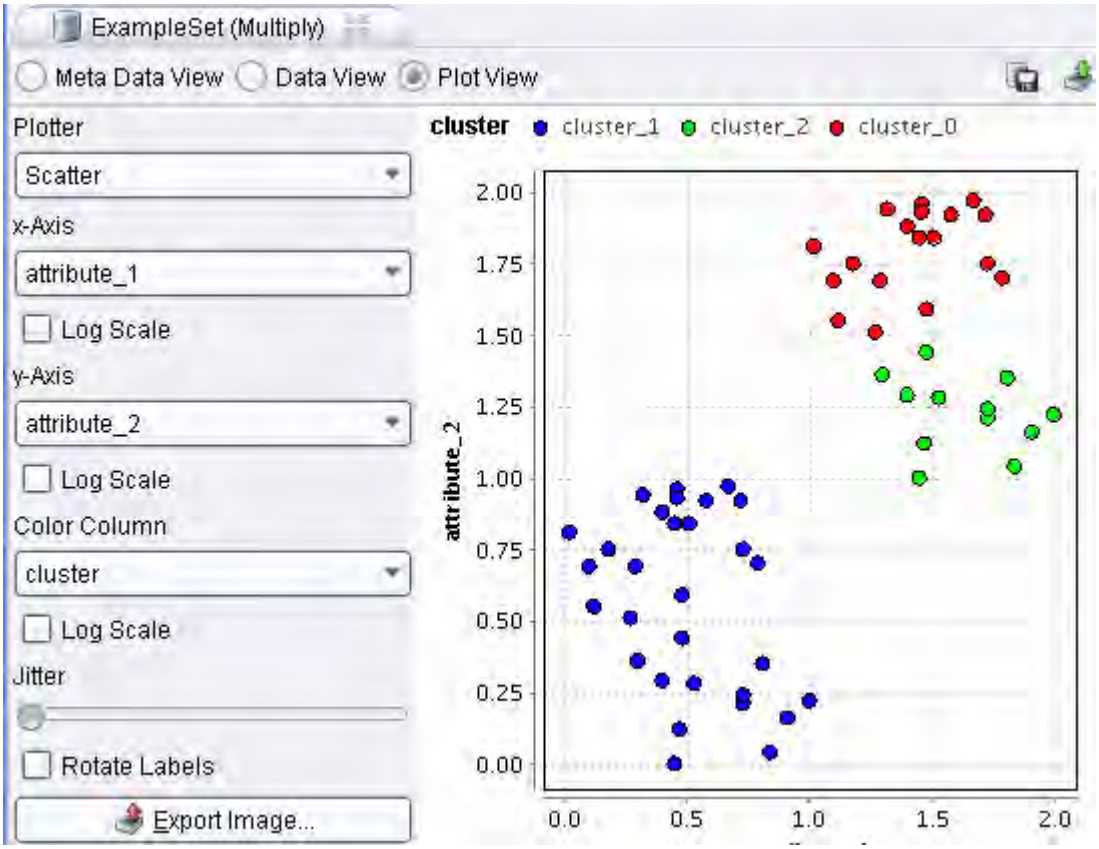


Рис. 14.4.6. Визуализация кластеризации.

В RapidMinere реализованы «потрясающие» средства для визуализации моделей: дендрограмм, каталогов кластеров и т.д. (а в классификации – деревьев решений). Здесь мы не приводим все иллюстрации, но советуем начинающему пользователю изучить графические возможности системы.

Глава 15. СРЕДА ДЛЯ ВЫЧИСЛЕНИЙ И ВИЗУАЛИЗАЦИИ MATLAB

§15.1. Знакомство с MatLab

MatLab – это программный продукт компании «The MathWorks, Inc.» [MathWorks], предназначенный для инженерных, научных и прикладных вычислений, а также визуализации и анализа их результатов. MatLab предлагает мощный С-подобный язык[1] (который удобнее С++ и FORTRANa для реализации численных методов), прекрасную графику и большое число различных библиотек (**toolbox**ов). Среда MatLab идеально подходит для решения задач машинного обучения, обработки сигналов и изображений и т.д. Пользователю предоставляется инструмент, с помощью которого можно загрузить и предобработать данные, запустить алгоритмы анализа, визуализировать результат, составить отчёт об экспериментах и получить исполняемый файл, который проводит нужные операции над этими данными. Слово MatLab означает **matrix laboratry** (матричная лаборатория). Все **вычисления** здесь **матричные,** и, в определенном смысле, только один (полезный) тип данных: матрица.

Так получилось, что самый лучший источник информации по MatLab – её справочная система, поэтому дальнейшее изложение построено в виде примеров, с помощью которых проще и быстрее изучить особенности среды MatLab. Мы остановимся только на основах программирования, принципах «правильного написания кода», эффективном использовании стандартных функций, оставив за рамками изложения подробное описание всевозможных библиотек[2].

При запуске системы появляются следующие окна:

1. Command Window (правое нижнее на рис. 15.1). В этом окне вводятся команды (после значка-приглашения **>>**). В нём же отображаются результаты выполнения. MatLab – это **интерпретатор**! Команды можно также записать в М-файл (текстовый файл *.m) в виде скрипта (последовательность команд) или функции (получает аргументы и выдаёт значения), тогда они будут запускаться при наборе в командном окне имени этого файла. Для создания такого файла вызовите редактор командой **edit** – появится окно редактора **Editor** (правое верхнее окно на рис. 15.1). Если нажать клавишу «стрелка вверх» после значка-приглашения **>>**, то отобразится предыдущая набранная команда (очень удобно, когда следующая команда незначительно отличается от предыдущей).

2. Command History (левое нижнее на рис. 15.1). В этом окне отображаются все команды, которые запускали на исполнение вводом в командном окне.

3. Workspace (левое верхнее на рис. 15.1). В этом окне отображаются все переменные, которые использует система в данный момент. Если набрать команду

[1] Этот язык называют также **MatLab**ом или М-языком (M-code).

[2] Некоторые особенности системы, например подготовка отчётов в MatLabe, изложены ранее (в предыдущих главах или первой части пособия – в зависимости от версии издания).

a=1, то в окне появится новая переменная: матрица **a** размера 1×1. Если набрать команду **b=2;**(точка с запятой), то новая переменная **b** появится в окне рабочего пространства, но в командном окне её результат не выводится. Это эффект действия «точки с запятой». Её используют, когда не требуется промежуточный вывод результатов, который может занять много времени или просто не нужен.

4. Help (окно помощи, вызываемое нажатием клавиши **F1**).

5. Current Directory (на рис. 15.1 «закрыто» окном Workspace, для активации необходимо щёлкнуть по вкладке с названием окна). Отображает **текущую директорию**. В этой директории система MatLab ищет файлы данных, которые Вы пытаетесь открыть и М-файлы, имена которых Вы набираете. Она также ищет их во всех каталогах, перечисленных в списке, доступном через меню **File / Set Path…** (его можно изменить вручную или пополнить командой **addpath**, см. ниже). В верней части основного окна MatLab есть специальный компонент для выбора текущей директории (также для смены текущей директории можно использовать команду **cd**, см. ниже).

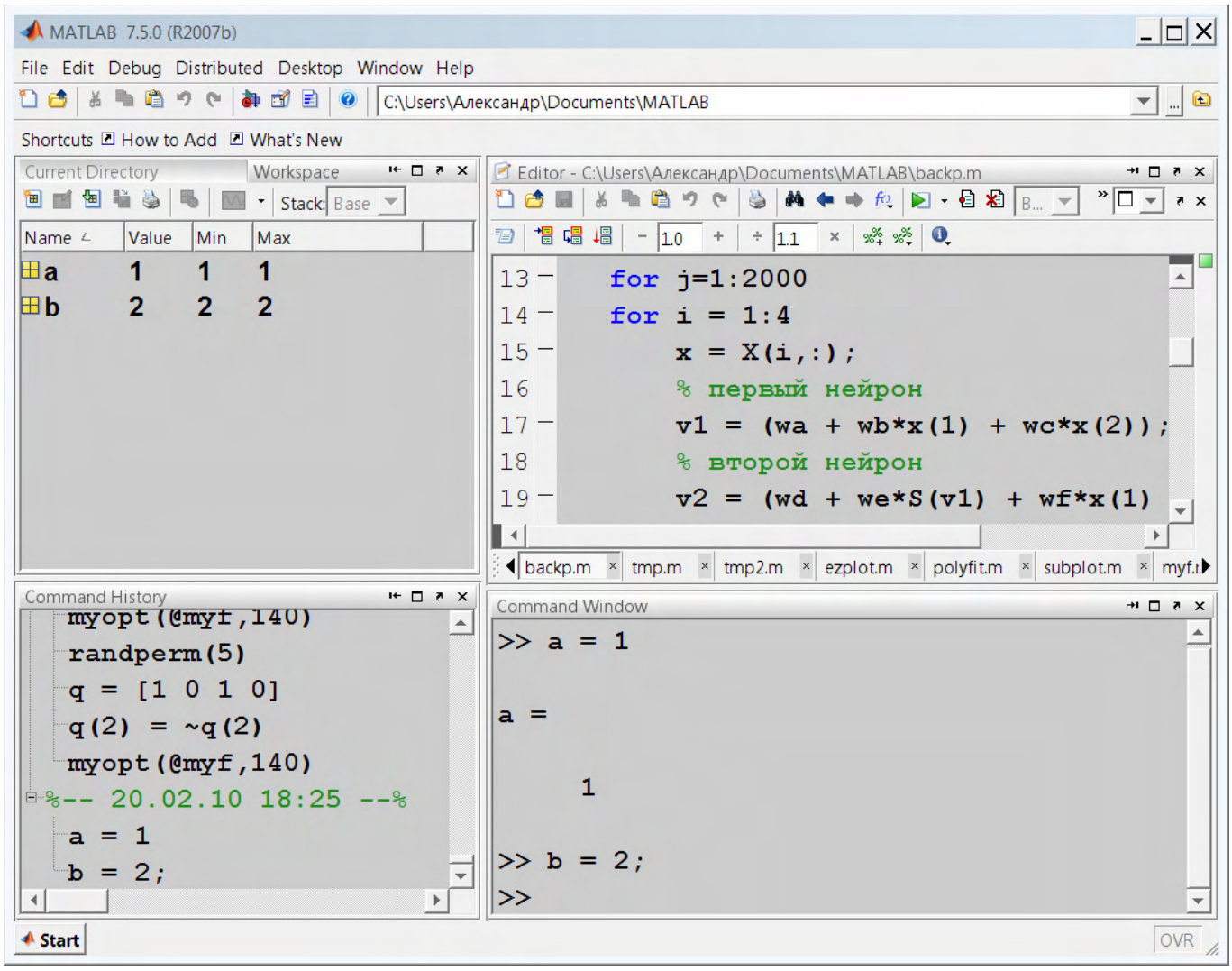


Рис. 15.1. Вид основного окна системы MatLab.

Назначение остальных окон будет ясно из контекста (например **Figure**) или файлов помощи (например **Profiler**).

При названии переменных **важен регистр**. Для начала работы можно набрать некоторые переменные, которым MatLab уже присвоил значения, хотя это не отображено в рабочей области:

i (мнимая единица),
j (мнимая единица),
pi (число пи),
Inf (бесконечность),
NaN («не число», чтобы понять – наберите **0/0**).
eps (порог чувствительности),
realmax (наибольшее число),
realmin (наименьшее число),
ans (результат последней операции),

Можно присваивать этим переменным новые значения, но тогда они теряют старые (попробуйте выполнить присваивание **pi=1**, команда **clear pi** вернёт исходное значение). Считается, что основное удобство при работе с системой MatLab – отсутствие необходимости определять переменные и распределять память, а также «краткость языка»: благодаря векторным (матричным) вычислениям и логическим массивам одна строчка в этой среде заменяет несколько строчек кода на языке C++. Для этого, правда, необходимо научиться некоторым тонкостям работы с системой, например избегать по возможности оператора **for**.

Далее текст разбит на две колонки. В левой приводятся команды системы MatLab и результат их выполнения, в правой – необходимые пояснения[1]. В каждой ячейке левого столбца таблицы приводятся команды, которые следует выполнять последовательно (следующие команды могут зависеть от результатов предыдущих).

§15.2. Работа с системой

>> help pi **PI 3.1415926535897....** **PI = 4*atan(1) = imag(log(-1)) = 3.1415926535897....** **Reference page in Help browser** **doc pi**	*Помощь. Попробуйте набрать* **sin(pi).**
>> help ans **ANS Most recent answer.** **ANS is the variable created automatically when expressions** **are not assigned to anything else.** **ANSwer.** **Reference page in Help browser** **doc ans**	*Кстати, с помощью* **help elfun** *можно посмотреть список реализованных элементарных функций.* **lookfor cat** – поиск в помощи слова «cat».
>> a = [1,2], b = a'	*Порождение двух векторов:*

[1] Данный материал использовался на занятиях «Практикума на ЭВМ» для студентов 3 курса кафедры Математических методов прогнозирования ВМК МГУ. В электронной версии пособия можно копировать команды левой колонки, вставлять их в командное окно и запускать.

a = 1 2 b = 1 2	*вектор-строки и вектор-столбца. Штрих обозначает транспонирование. Элементы векторов заключаются в* ***квадратные*** *скобки!*
>> a*b ans = 5	*Скалярное произведение (строка на столбец).*
>> b*a ans = 1 2 2 4	*Внешнее произведение (столбец на строку).*
>> a.*b' ans = 1 4	*Поэлементное произведение. Все* ***поэлементные операции «помечаются» точкой*** *перед знаком операции.*
>> clear b	*Удаление переменной* **b** *из памяти.*
>> whos Name Size Bytes Class Attributes a 1x2 16 double ans 1x2 16 double	*Вывод всех переменных в памяти. Список переменных текущей рабочей области выводит команда* **who**.
>> path	*Вывод путей доступа (определены в pathdef.m).*
>> addpath c:\matlab7	*Добавление нового пути*
>> cd C:\MATLAB7\work >> cd .. >> cd C:\MATLAB7	*Вывод текущего каталога (изменение его).* *Для вывода списка файлов в каталоге наберите* **what C:\MATLAB7**
>> !dir	**!** *или* **dos** *– вызов команды DOS.*
>> diary 1.txt >> diary off	*Писать в файл 1.txt историю команд (вести дневник).* *Отключить ведение дневника.*
>> 1+1+1+... % переход	*Три точки подряд – переход*

`+1+1 % комментарий`
 	
`ans =`
`5`
	на следующую строку. Знак процента – комментарий.

`>> format short; pi, format long; pi, format short e; pi, format long e; pi, format rat; pi`
 	
`ans =`
`3.1416`
`ans =`
`3.141592653589793`
`ans =`
`3.1416e+000`
`ans =`
`3.141592653589793e+000`
`ans =`
`355/113`
	Форматы вывода чисел. См. помощь **help format.** *Последний формат – приближение рациональной дробью. Команда* **format** *влияет только на формат вывода, а не на внутреннее представление.*

`>> which plot`
 	
`C:\MATLAB\toolbox\matlab\graph2d\plot.bi`
	Путь к файлу, в котором определена функция. В новых версиях выводится что-то типа **built-in (C:\Program Files\MATLAB\R2007b\toolbox\matlab\graph2d\plot).**

`>> edit`	*Запуск редактора-отладчика.*
`>> quit`	*Завершить работу с Matlab.*
`>> clear a*`	*Удалить из рабочей области все переменные, которые начинаются на букву* **a.**
`>> clc`	*Очистить командное окно.*

`>> [Inf, Inf, Inf, NaN, NaN, 0].*[Inf 0 -Inf Inf 0 Inf]`
 	
`ans =`
`Inf NaN -Inf NaN NaN NaN`
	Иллюстрация работы с «бесконечностью» (причём со знаком «плюс» или «минус») и «не числом».
	
	Обратите внимание, что элементы вектор-строки можно разделять пробелом, а можно – запятой.
	
	Попробуйте набрать **-1/0** *и* **1/0.**

`>> conj(i)`
 	
`ans =`
`0 - 1.0000i`
	Пример вызова функции (комплексного сопряжения).

`>> demo`	*Запуск демонстрации.*

§15.3. Теоретико-множественные операции

```
>>  A = [1,3,5,3]; B = [4,2,2,3];
>> [A, B]

ans =
     1   3   5   3   4   2   2   3

>> union(A, B)

ans =
     1     2     3     4     5

>> setdiff(A, B)

ans =
     1     5

>> intersect(A, B)

ans =
     3

>> ismember(2, B)

ans =
     1

>> ismember(A, B)

ans =
     0     1     0     1


>> setxor(A, B)

ans =
     1     2     4     5


>> unique(B)

ans =
     2     3     4


>> issorted(ans)

ans =
     1

>> sort(B)

ans =
```

Задание двух множеств. Конкатенация.

Объединение (обратите внимание на упорядоченность в ответе).

Разность.

Пересечение.

«Членство» (входит ли элемент 2 во множество B, 0 *– нет,* 1 *– да, см. дальше «Логические массивы»).*

«Членство». Особенность Matlaba: применение функции сразу к массиву (множеству). Сравните с **sin([0,1,pi/2])**.

Симметрическая разность. Сравните с **union(setdiff(A, B), setdiff(B,A))** *и* **setdiff(union(A,B), intersect(A,B))**.

Уникальные элементы. Выдаёт массив (множество) без повторений. Сравните действие **union(A,B)** *с* **unique([A,B])**.

Отсортированность: является ли множество отсортированным.

Сортировка.

```
      2     2     3     4

>> [1 4 -2] + 1

ans =
      2     5    -1
```

Увеличить все элементы множества на единицу. Обратите внимание, что допустимо также сложение **[1 4 -2] + [1 1 1]**, *но не* **[1 4 -2] + [1 1]** (*сложение матриц разных размеров*).

§15.4. Двоичные представления

```
>> a = bitget(19, 1:8)

a =
    1   1   0   0   1   0   0   0

>> b = bitget(24, 1:8)

b =
    0   0   0   1   1   0   0   0

>> bitand(a,b)

ans =
      0   0   0   0   1   0   0   0

>> dec2bin(1:8)

ans =
     0001
     0010
     0011
     0100
     0101
     0110
     0111
     1000
```

Представление в двоичном виде (чисел 19 *и* 24 *с помощью* 8 *бит).*

Конъюнкция (логическая операция И). *Есть еще* **bitor** (*ИЛИ*), **bitxor** (*исключающее ИЛИ*).

Вывести все двоичные представления чисел от **1** *до* **8**. *При наборе команды на ЭВМ обратите внимание на тип ответа!*

§15.5. Порождение матриц и работа с ними

```
>> A = [1,2; 3,4]

A =
     1     2
     3     4

>> A(1, 2)

ans =
      2

>> cat(3, A, A+1)
```

Запятая (и пробел) является горизонтальной конкатенацией, «точка с запятой» – вертикальной.

Обращение к элементу матрицы. ***Сначала указывается строка, потом столбец. Нумерация от единицы!***

Конкатенация по третьему (!)

```
ans(:,:,1) =
             1     2
             3     4


ans(:,:,2) =
             2     3
             4     5
```

направлению матрицы **A** *и матрицы* **A+1**. ***в MatLabe матрицы многомерные!*** *К матрице можно прибавлять скаляр или матрицу такого же размера.*

```
>> D = [zeros(3) 2*ones(3,4) ...
       3*eye(3,2), diag([4,5,1])]

D =
    0  0  0  2  2  2  2  3  0  4  0  0
    0  0  0  2  2  2  2  0  3  0  5  0
    0  0  0  2  2  2  2  0  0  0  0  1
```

zeros – *матрица из нулей,* **ones** – *матрица из единиц,* **eye** – *единичная матрица,* **diag** – *диагональная матрица. Многоточие – перевод команды на другую строку*[1] (*команда пока не завершена*).

```
>> D(1:2, 7:8)

ans =
     2     3
     2     0
```

Подматрица матрицы. **a:b** – *вектор с элементами* **a,a+1,...,b**.

```
>> D(:, 2*(1:6)) = []

D =
     0     0     2     2     0     0
     0     0     2     2     3     5
     0     0     2     2     0     0
```

Удаление всех чётных столбцов. **[]** – *пустая матрица.* **:**(*двоеточие*) – *все элементы в данном направлении.* **2*a** – *поэлементное умножение вектора.*

```
>> D(:)'

ans =

 0 0 0 0 0 0 2 2 2 2 2 2 0 3 0 0 5 0
```

Вывести все элементы матрицы. «Штрих» – транспонирование (*чтобы выводилось в строку*).

MatLab выводит элементы по столбцам! *Попробуйте выполнить* **D(7).**

```
>> B = fix(10*rand([2 3]))

B =
     2     9     1
     5     9     9
```

rand – *случайная матрица* (*см. также* **randn**), **fix** – *отбросить дробную часть. См. также* **round** (*округление к ближайшему целому*), **floor** (*округление к наименьшему целому*).

[1] По возможности будем переводить длинные команды на новую строку, но в некоторых случаях такой перевод будет неудобен и длинные команды MatLaba займут несколько строк (с обычным переносом «по пробелам»). Такие «вынужденные переносы» будут легко идентифицироваться, тем более что начало каждой команды (кроме некоторых комментариев) помечаются символами **>>**.

	Какие элементы могут быть в матрице **B***?*
>> [max(B); min(B); sum(B)] **ans =** **5 9 9** **2 9 1** **7 18 10**	*Максимальные и минимальные элементы в столбцах. Суммы столбцов. См. также* **cumsum** *(кумулятивная сумма),* **prod** *(произведение),* **cumprod** *(кумулятивное произведение),* **sort** *(сортировка матрицы),* **mean** *(ср. арифметическое),* **median** *(медиана).*
>> C = reshape(1:2:11, [2 3]) **C =** **1 5 9** **3 7 11**	**reshape** *– изменение формы матрицы (форма меняется «по столбцам»).* **a:b:c** *– вектор из чисел* **a, a+b, ...** *Команды* **1:4, 1:1:4, linspace(1,4,4)** *эквивалентны (последняя «делит» отрезок* **[1,4]** *четырьмя точками). Попробуйте команду* **logspace(1,4,4)**, *которая «действует в логарифмической шкале».*
>> min(C, B) **ans =** **1 5 1** **3 7 9**	*Минимум (поэлементный) двух матриц.* *Кстати,* **min(C,[],2)'** *и* **min(C')** *– минимальные элементы по строкам.*
>> C(end, 1) **ans =** **3**	**end –** *последняя позиция:* **C(end,end)** *– элемент в позиции* (2,3), **C(end,:) –** *последняя строка,* **C(:,end)** *– последний столбец.*
>> B.*C **ans =** **2 45 9** **15 63 99**	*Поэлементное произведение матриц. Точка – указание того, что операцию надо произвести поэлементно.* **B.^2** *– поэлементное возведение в квадрат.*
>> a = reshape(repmat(1:3,2,1), 1, 6) **a =** **1 1 2 2 3 3**	*Получение строки* **1 1 2 2 3 3. repmat** *– повтор матрицы. Например,* **repmat([1,0;0,1],50,50)** *–* матрица типа шахматной доски. Попробуйте выполнить команду

```
>> A = [1 5 2; 3 2 1; 5 4 0];
>> [B I] = sort(A)

B =
     1     2     0
     3     4     1
     5     5     2

I =
     1     2     3
     2     3     2
     3     1     1
```

imagesc(repmat([1,0;0,1], 4, 4)).

Сортировка элементов столбцов(!) матрицы! Выводится результат сортировки и матрица перестановок индексов. Для вывода только результата наберите **B = sort(A)**. *Для сортировки элементов строк –* **sort(A,2)**. *Для сортировки в обратном порядке –* **B = sort(A,'descend').**

```
>> [B I] = sortrows(A, 2)

B =
     3     2     1
     5     4     0
     1     5     2

I =
     2
     3
     1
```

Сортировать строки по возрастанию элементов второго столбца. Выводится перестановка номеров строк.

```
>> B = rand([1 2 3 5 1]);
```

Порождение случайной матрицы размера 1×2×3×5×1.

```
>> size(B)

ans =
     1     2     3     5
```

Вывод её размера. Обратите внимание, что последняя (фиктивная!) размерность устранена. Для устранения всех фиктивных размерностей используйте **B = squeeze(B).**

```
>> size(permute(B, [2 1 4 3]))

ans =
     2     1     5     3
```

permute – *перестановка размерностей.*

```
>> size(shiftdim(B, 1))

ans =
     2     3     5
```

shiftdim – *сдвиг размерностей. Последняя фиктивная размерность всегда удаляется. Три способа транспонирования матрицы*: **[shiftdim([1,2;3,4],1) , permute([1,2;3,4],[2,1]) , [1,2;3,4]'].**

```
>> [length(B), ndims(B), size(B,4)]
```

length – *длина вектора. Для матриц эквивалентно*

```
ans =
     5     4     5
```

max(size(B)). **ndims(B)** – *число размерностей, эквивалентно* **length(size(B))**. **size(B,4)** – *показать длину* 4*-й размерности.*

```
>> A = [], A(end+2) = 2

A =
     []

A =
     0     2
```

Вектор (матрица) автоматически дополняется нулями. Команда **A(end+1) = val** *добавляет к вектору* **A** *ещё один элемент. Присваивание* **A = [val A]** *добавляет его слева. Обратите внимание, что команды можно записывать в одной строке через запятую (или через точку с запятой – тогда не будет выведен промежуточный результат).*

```
>> A = toeplitz([1 2 3])

A =
     1     2     3
     2     1     2
     3     2     1
```

Порождение матрицы специального вида. См. также функции «галереи». Например, порождение циркулянтной матрицы **gallery('circul',1:3)** *или случайной бинарной* **gallery('rando',5).**

```
>> A(:, 2:3) = A(:, [3 2])

A =
     1     3     2
     2     2     1
     3     1     2
```

Перестановка столбцов.

```
>> rot90(A)

ans =
     2     1     2
     3     2     1
     1     2     3
```

Поворот матрицы на 90 градусов.

```
>> x=A\[1 2 4]'

x =
    1.1250
   -0.3750
    0.5000
```

Решение уравнения **A*x=[1 2 4]'**. *Не надо думать! Просто формально поделить на матрицу* **A**. *Способ* **x=inv(A)*[1 2 4]'** *хуже, т.к. происходит инвертирование матрицы. Обратите внимание на направление деления, сравните* **1/2** *и* **1\2.**

```
>> A*x - [1 2 4]'

ans =
     1.0e-015 *
               -0.2220
                     0
                     0
```

Уравнение решено с погрешностью!

```
>> A^2

ans =
     13    11     9
      9    11     8
     11    13    11
```

Возведение матрицы в квадрат. Сравните с поэлементным возведением **A.^2**.

```
>> sub2ind([3,2], [1,2,3], [1,1,2])

ans =
      1     2     6
```

Перевод многомерной нумерации в последовательную. Например, в матрице размера 3×2 *элемент в позиции* (2,1) *соответствует второму элементу вектора всех элементов матрицы. См. также* **ind2sub**.

```
>> A = reshape(1:9, 3, 3)

A =
     1     4     7
     2     5     8
     3     6     9
```

```
>> A(:, end) = [1 2 3]

A =
     1     4     1
     2     5     2
     3     6     3
```

При присваивании строка автоматически конвертируется в столбец (главное совпадение числа элементов). Можно писать также **A(:,end) = [1 2 3]'**, *но нельзя* **A(:,end) = [1 2]'**.

```
>> A(end, :) = 5

A =
     1     4     1
     2     5     2
     5     5     5
```

Константа присваивается сразу всем указанным элементам.

```
>> A(1,2,1,1)

ans =
      4
```

На единицы в несуществующих размерностях MatLab не обращает внимание.

```
>> A(1, 2, 2)
??? Index exceeds matrix dimensions.
```

На не-единицы обращает...

```
>> A(1:8) = rand([2 4])

A =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    5.0000
```

При присваивании вектору (!) *происходит «подгонка» размеров. Нельзя сделать* **A(1:2,1:2) = rand([1 4])**.

```
>> A(2,2,2) = 3

A(:,:,1) =
          0     0
          0     0

A(:,:,2) =
          0     0
          0     3
```

Создаётся матрица размера 2×2×2, *все элементы которой, кроме элемента в позиции* (2,2,2) *равны нулю.*

```
>> accumarray([1 2 3 1 1 2]', [1 2 3 4 5 6])

ans =
      10
       8
       3

>> accumarray([1 2 3 1 1 2]', [1 2 3 4 5 6], [], @prod)

ans =
      20
      12
       3
```

Полезная функция **accumarray**. *Формирует вектор, состоящий из суммы элементов, помеченных* 1, *суммы элементов, помеченных* 2 *и т.д.*

Аналогично с произведением. Обратите внимание: **@prod** – *указатель на функцию* **prod**.

```
>> rank([A,b]) == rank(A)
```

Определяет, существует ли решение системы **A*x = b**(*матрица* **A** *может быть вырождена, т.е. её определитель* **det(A)** *равен нулю)?* **rank** – *ранг матрицы.*

```
>> [sum([]) sum([]+4) prod([])]

ans =

     0     0     1
```

Обратите внимание на значение суммы и произведения элементов пустого множества, а также на то, что **[]+4 = []**.

```
>> A = [i 2*i; 3*i 4*i]; A'

ans =
      0 - 1.0000i      0 - 3.0000i
      0 - 2.0000i      0 - 4.0000i

>> A.'

ans =
      0 + 1.0000i      0 + 3.0000i
      0 + 2.0000i      0 + 4.0000i
```

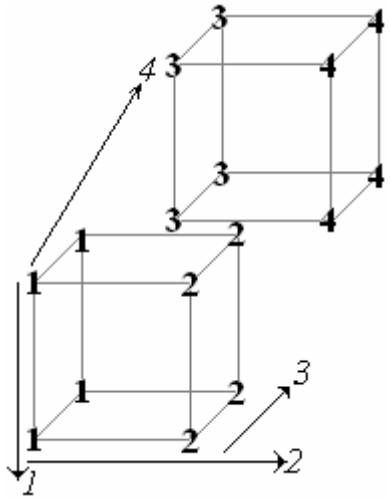
Отличие транспонирования и «транспонирования с комплексным сопряжением».

§15.6. Фокусы с размерностями

```
>> A = [1 1; 1 1]; % A = ones(2);
>> X = [A, 3*A; 2*A 4*A]

X =
     1     1     3     3
     1     1     3     3
     2     2     4     4
     2     2     4     4

>> Y = reshape(X, [2 2 2 2])

Y(:,:,1,1) =
             1     2
             1     2

Y(:,:,2,1) =
             1     2
             1     2

Y(:,:,1,2) =
             3     4
             3     4

Y(:,:,2,2) =
             3     4
             3     4

>> Y = permute(Y, [1 3 2 4]);
>> Y = reshape(Y, [2 2 4])

Y(:,:,1) =
             1     1
             1     1

Y(:,:,2) =
             2     2
             2     2

Y(:,:,3) =
             3     3
             3     3

Y(:,:,4) =
             4     4
             4     4
```

Создаём матрицу, состоящую из четырёх блоков. Следующий код иллюстрирует, как продублировать эти блоки в третьем направлении, т.е. реализовать команду **cat(3, A, 2*A, 3*A, 4*A)**.

Превращаем матрицу в четырёхмерную.



Делаем перестановку размерностей и изменяем форму. Задача решена.

Попробуйте реализовать команду **cat(3, A, 3*A, 2*A, 4*A)** *аналогичным способом.*

```
>> x = ones(1,2,1,0,0,1)

x =

   Empty array: 1-by-2-by-1-by-0-by-0

>> squeeze(x)
```

Обратите внимание, что порождён пятимерный массив. Последняя размерность фиктивная, но предпоследняя (нулевая!) не является фиктивной.

```
ans =

   Empty array: 2-by-0-by-0
```

Отбросили фиктивные (единичные) размерности.

```
>> Y = reshape(X,[m size(X,1)/m n ...
 size(X,2)/n]);
% сам поворот
>> Y(:,:,:) = Y(:,:,end:-1:1);
% игра с размерностями
>> Y = permute(Y,[1 4 3 2]);
>> Y = reshape(Y, [m*size(X,2)/n ...
 n*size(X,1)/m])

% Если убрать вторую команду,
% то получим обобщённую транспозицию
```

Даны двумерные матрицы **A, B, C, D, E, F, G, H, E** *одинаковых размеров* **[m n]**. *Матрица* **X** *имеет вид*

```
X = [A B C; ...
     D E F; ...
     G H E].
```

Этот код осуществляет «внешний» матричный поворот [Acklam]:

```
    [C F E; ...
     B E H; ...
     A D G].
```

§15.7. Полезные функции

```
>> x = 1; y = [1 2];
>> [x,y] = deal(y,x)

x =
       1     2

y =
       1

>> f = @(varargin) varargin{:};

>> [y x] = f(x,y)

y =
       1     2

x =
       1
```

Функция **deal** *позволяет «удобно» присваивать значения аргументам. Здесь показано, как поменять две переменные (даже разных типов). Заметим, что конструкция* **[x y] = [y x]** *не работает.*

Для этих целей можно использовать и анонимную функцию (см. также ниже).

```
>> [x y z] = deal(7)

x =
       7
y =
       7
z =
       7
```

Ещё одно использование функции **deal**. *Вместо записей*

```
x = 7;
y = 7;
z = 7;
```

```
>> row = [1 1 2 3 3 2 3]';
>> col = [1 2 1 2 2 1 2]';
>> val = [1 2 3 4 5 6 7]';

>> [row, col, val] = find(accumarray(...
[row, col], val, [], @sum, [], true))

row =
```

Решение следующей задачи. Допустим, у нас есть три массива одной длины **row, col, val**. *Элемент некоторой матрицы (потом мы увидим, что так реализуются в MatLabe разреженные матрицы) с номерами* **(row(i), col(i))**

```
       1
       2
       1
       3

col =
       1
       1
       2
       2

val =
       1
       9
       2
      16
```

имеет значение **val(i)**. *Если пара* **(row(i), col(i))** *одна и та же для нескольких* **i**, *то элемент равен сумме соответствующих* **val(i)**. *Требуется перевести данные в «сжатую форму», когда нет одинаковых пар* **(row(i), col(i))** *для разных* **i**. *Например,*

```
row = [1 1 2 3 3 2 3]';
col = [1 2 1 2 2 1 2]';
val = [1 2 3 4 5 6 7]';
```

в

```
row = [1 1 2  3]';
col = [1 2 1  2]';
val = [1 2 9 16]';
```

```
>> clear; exist('a')

ans =
       0

>> a = 10; exist('a')

ans =
       1

>> exist('pi')

ans =
       5
```

Существует ли переменная **a**?

«*Системная*» *константа* **pi**.

```
>> A = 1;
>> L = 10;
>> for i = 1:L
>> A(i+1) = 2*A(i) + 1;
>> end
% «эквивалентный» код
>> B = filter([1], [1 -2], ones(1,L+1))

B =
    1   3   7   15   31   63   127   255   511
1023  2047

>> isequal(A, B)

ans =
      1
```

См. справку по функции **filter**, *которая применяется при обработке сигналов, но может быть также полезна и для других задач.*

Сравнение на равенство.

§15.8. Операции с файлами

```
>> save b.mat a


>> load b.mat
```

Сохранить массив с именем «a» в файл «b.mat».

Загрузить данные из файла «b.mat».

`>> save b.txt a –ascii`	*Сохранить в формате* ***ascii.*** **-tabs** *– разделение символов табуляцией,* **-v4** *– создание файла для системы MATLAB4,* **-append** *– добавление в существующий файл.*

`>> fid = fopen('answer.txt', 'wt', 'n');`
`>> fprintf(fid, '%d\n', y);`
`>> fclose(fid);`
	На практике чаще всего используют такой способ сохранения (например, когда записывают в файл вектор-столбец ответов на задачу классификации). Поскольку команда **save answer.txt y -ascii** *число* 82 *записывает как* 8.2000000e+001.
	
	В MatLabe есть ещё следующие низкоуровневые функции для работы с файлами: **fread, fscanf, fgetl, fseek.**

`>> f = fopen ('st.txt');`
`>> a = fscanf(f, '%d:%d %d/%d',...`
`[4 100])'`
 	
`a =`
`12 1 5 7`
`13 6 3 2`
 	
`>> fclose(f);`
 	
`>> g = fopen('st2.txt','w');`
 	
`>> fprintf(g, ...`
`'%2.2d:%2.2d %2.2f/%2.2d\n', a');`
`>> fclose(g);`
	Чтение файла, в котором записано
	12:01 5/7
	13:06 3/2
	
	Большое значение «100» взято «на всякий случай».
	
	Попробуйте до закрытия файла повторить **fscanf.**
	
	Сохранение в файл в формате
	12:01 5.00/07
	13:06 3.00/02

`>> A = dlmread('1.txt', '+')`
 	
`A =`
`1 2 3`
	Загружаются данные из файла, который содержит единственную строку **1+2+3.**
	
	Очень удобная функция для загрузки текстовых файлов (особенно с таблицами данных) **textread.**

Загружать данные (бинарные, ascii-файлы, Excel-таблицы и т.д.) можно также выбрав в меню **File / Import Data**, что часто проще и удобнее.

§15.9. Разреженные матрицы

В анализе данных часто приходится иметь дело с матрицами больших размеров, почти все элементы которых нулевые. Для хранения таких матриц в

системе MatLab есть специальный тип – разреженные матрицы (sparse). При этом работа с разреженными матрицами происходит также как и с обычными.

```
>> A = fix(rand(3)*3)

A =
     2     1     0
     2     0     2
     2     0     0

>> S = sparse(A)

S =
   (1,1)        2
   (2,1)        2
   (3,1)        2
   (1,2)        1
   (2,3)        2

>> S(1,:)

ans =
   (1,1)        2
   (1,2)        1


A = speye([2 3])

A =
   (1,1)        1
   (2,2)        1

>> B = sparse([1 2 1], [2 3 1], ...
[4 7 2], 3, 4)

B =
   (1,1)        2
   (1,2)        4
   (2,3)        7


>> sparse(rand([2 2 2]))
??? Undefined function or method
'sparse' for input arguments of type
'double' and attributes 'full 3d real'.

>> spy(S)
```

Перевод матрицы в разреженную форму. Хранятся только ненулевые элементы. Обратный переход производится с помощью **full(S)**.

Первая строка разреженной матрицы.

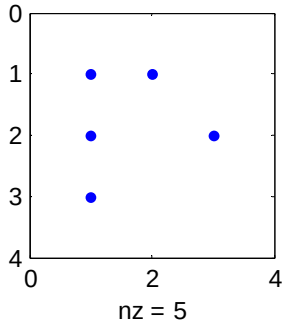
spones(A) *– заменяет все ненулевые элементы на единицы, а* **nnz(A)** *– число ненулевых элементов.*

Создание разреженной единичной матрицы. См. также **spdiags**, **sprand**, **sprandn**.

Создание разреженной матрицы. Перечисляются индексы элементов, их значения, указываются размеры матрицы. Попробуйте набрать без указания размеров матрицы: **sparse([1 2 1],[2 3 1],[4 7 2]**.

В MatLabe только двумерные разреженные матрицы!

Визуализация матрицы:




```
>> S = spalloc(2,3,2);
>> whos
  Name  Size  Bytes  Class  Attributes
  S     2x3   40     double sparse

>> S(1,1) = 1;
>> S(1,2) = 2;
>> whos
  Name  Size  Bytes  Class  Attributes
  S     2x3   40     double sparse

>> S(2,1) = 3;
>> whos
  Name  Size  Bytes  Class  Attributes
  S     2x3   88     double sparse
```

Выделить память для разреженной матрицы размера 2×3 *с двумя ненулевыми элементами.*

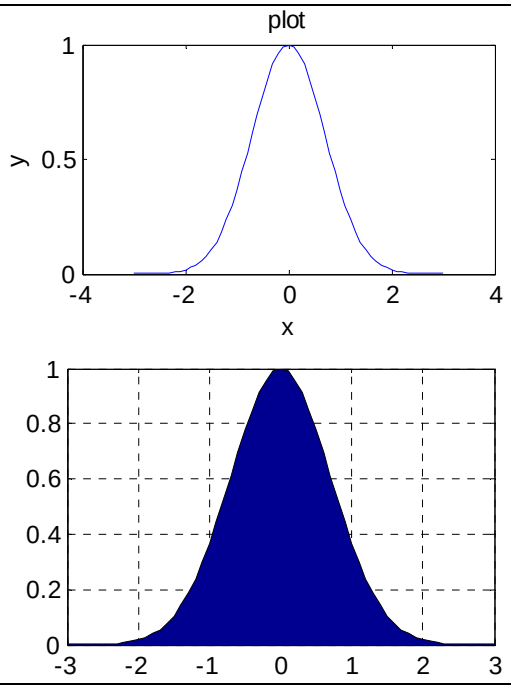
Памяти хватает...

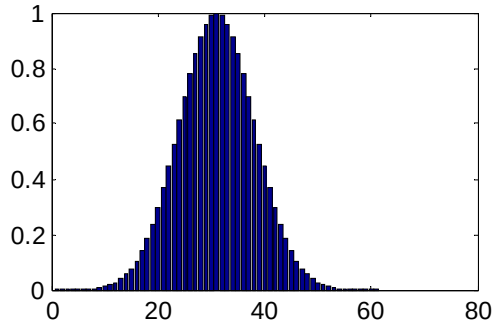
«Лишний» элемент вызывает перераспределение памяти. Посмотрите на результат команд **S = spalloc(2,3,3); whos**. ***Лучше сразу знать, сколько памяти потребуется!***

Если **S** – разреженная матрица, а **F** – обычная («полная»), то матрицы **S+S**, **S*S**, **S.*S**, **S.*F**, **inv(S)**, **diag(S)**, **max(S)**, **sum(S)** разреженные, а **S+F**, **S*F** – полные.

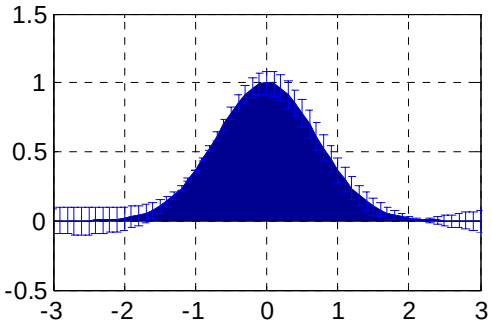
§15.10. Графика

```
>> x = -3:0.1:3;
>> y = exp(-x.*x);
>> plot (x,y);
>> title('plot'); % название
>> xlabel('x'); % метки осей
>> ylabel('y');
>> figure % создание нового окна
>> area(x, y); % закрашенный график
>> grid on; % нарисовать сетку
>> figure(1); % вывод в первое
>> bar(y)
>> figure(2);
>> hold on; % Рисовать в окне без
удаления старого рисунка
>> errorbar(x, y, sin(x+1)./10);
```





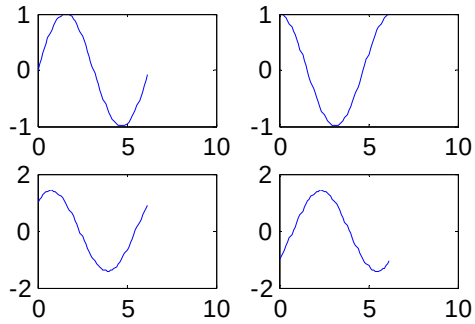
Пометки осей и название удалились!



Выполните эти команды последовательно.
Попробуйте команду **plot** *от двухмерной матрицы.*

```
>> clf % очистить окно вывода
>> t = (0:.1:2*pi)';
>> subplot(2, 2, 1)
>> plot(t, sin(t))
>> subplot(2, 2, 2)
>> plot(t, cos(t))
>> subplot(2, 2, 3)
>> plot(t, sin(t)+cos(t))
>> subplot(2, 2, 4)
>> plot(t, sin(t)-cos(t))
```

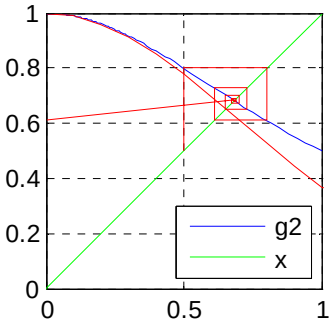
Построение нескольких графиков в одном окне.



Удалите команды **subplot.**
Поставьте на их место команды **hold on;**

```
>> g2 = @(x) 1./(x.^2+1);
>> fplot(g2, [0 1]);
>> hold on
>> straightLine = @(x) x;
>> fplot(straightLine, [0 1], 'g')
>> legend('g2', 'x', ...
          'Location', 'SouthEast')
>> grid on
>> axis equal, axis([0 1 0 1])
>> x(1) = 0.5;
>> y(1) = x(1);
>> x(2) = x(1);
>> y(2) = g2(x(2));
>> for n = 3:2:21
>> x(n) = y(n-1);
```

Нахождение неподвижной точки [Loren].



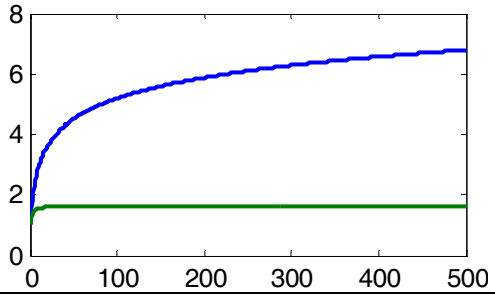
'r' – *красный цвет (возможно*

```
>> y(n) = y(n-1);
>> x(n+1) = x(n);
>> y(n+1) = g2(x(n+1));
>> end
>> plot(x,y,'r')
>> hold off
```

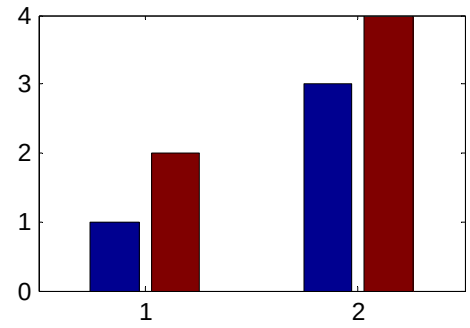
использование букв **b, g, r, c, m, y, k**).

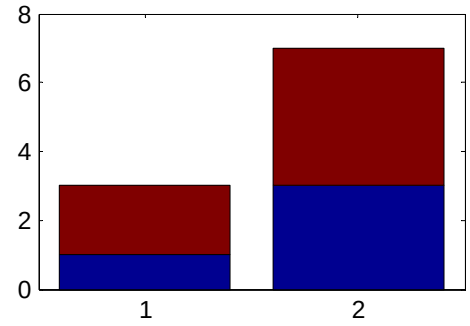
```
>> n = 1:500;
% кумулятивная сумма
>> s1 = cumsum(1./n);
>> s2 = cumsum(1./(n.*n));
>> plot([[s1;s2]]', 'LineWidth', 2)
```

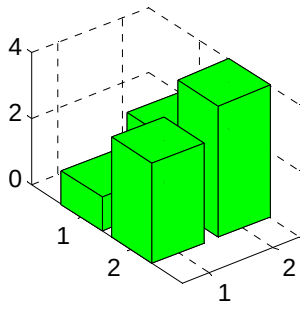
Иллюстрация сходимости и расходимости рядов.

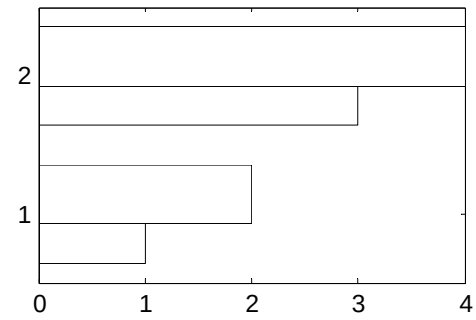



```
>> bar([1,2;3,4], 'grouped');
>> bar([1,2;3,4], 'stacked');
>> bar3([1,2;3,4], 'g');
>> barh([1,2;3,4], 1.5, 'w');
```

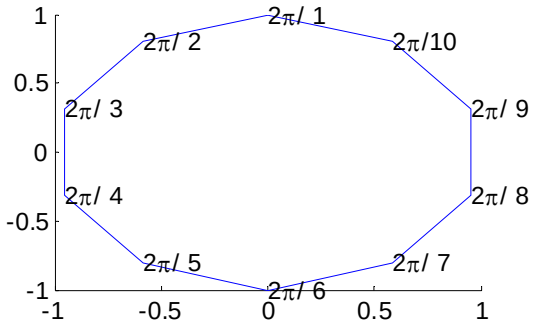






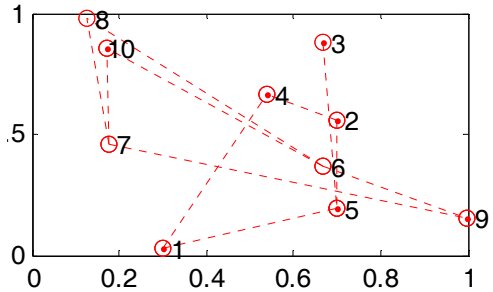


```
>> A = sin((0:10)*pi/5);
>> B = cos((0:10)*pi/5);
>> line(A, B); hold on;
>> text(A(2:11), B(2:11), ...
[repmat('2\pi/', 10, 1), ...
int2str(10-(0:9)')], 'FontSize', 10);
```




```
>> D = rand(10,2);
>> B = fix(triu(blkdiag(2*rand(5,5),...
                        2*rand(5,5))));
>> gplot(B,D,':og');
>> text(D(:,1),D(:,2),int2str((1:10)'));
```

Построение 2х случайных графов.



Кстати, **randn(m,n)** *– случайная матрица (элементы распределены по нормальному закону) размера* **m*n**.

```
>> saveas(gcf, 'myfig.fig')

>> openfig myfig

>> saveas(1, 'myfig.eps')
```

Сохранение рисунка.

Загрузка рисунка.

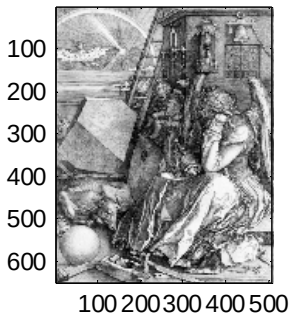
Сохранение **Figure 1** *в формате* **eps**.

```
>> set(gcf, 'paperpos', [0 0 3 2.25])
>> saveas(1, 'myfig.eps')
```

Попробуйте такой вариант сохранения. Посмотрите разницу. Попробуйте также **print -f -djpeg**.

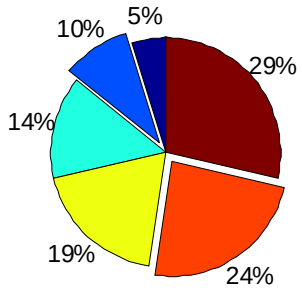
```
>> load durer
>> whos
  Name     Size      Bytes     Class
  X        648x509   2638656   double array
  ans      648x509   2638656   double array
  caption  2x28      112       char array
  map      128x3     3072      double array

Grand total is 660104 elements using
5280496 bytes

>> imagesc (X);
>> colormap(map);
>> axis image
```

Загрузка картинки.



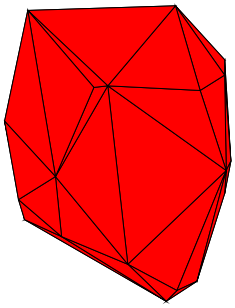

```
>> pie([1 2 3 4 5 6], [0 1 0 0 1 0])
```

Секторная диаграмма («пирог»), с пометкой, какие сектора отделять.



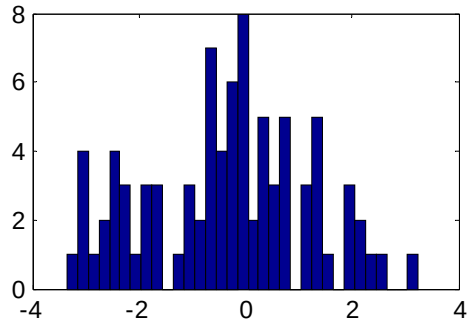

```
>> A = randn(100, 3);
>> C = convhulln(A);
>> for i=1:size(C, 1)
>> j = C(i, [1,2,3,1]);
>> patch(A(j,1), A(j,2), A(j,3), 'r');
>> end;
>> view(3), axis equal off tight vis3d;
>> camzoom(1.5)
>> camlight; lighting phong
```

Выпуклая оболочка точек.



```
>> a = [randn([1 10])+2 randn([1 50])...
randn([1 25])-2];
>> x = [min(a):0.2:max(a)];
>> hist(a, x)

% Кстати, очень полезная функция...
>> q = hist(a, x)

q =
     1  4  1  2  4  3  1  3  3  0  1  3
2  7  4  6  8  2  5  3  5  0  3  5  1  0
3  2  1  1  0  0  1
```

Гистограмма.



Обратите внимание, что

```
>> q = hist([0 0.5 1 1.5
2],[0 1 2])

q =
     2     2     1
```

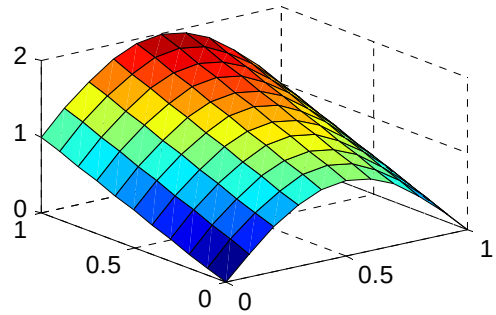
Для изменения цвета наберите

```
h = get(gca,'Children');
set(h,'FaceColor', 'r');
```

```
>> [A, B] = meshgrid([1,2], [3,4])

A =
     1     2
     1     2

B =
     3     3
     4     4
```

Действие функции **meshgrig**. *Эта функция нужна для построения трёхмерных графиков (функций от двух переменных).*

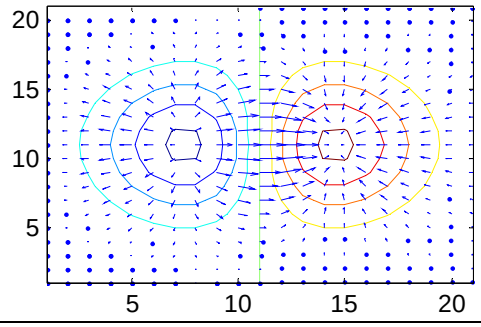
```
>> x = 0:0.1:1;
>> y = linspace(0,1,11);
% или y=0:0.1:1
>> [X,Y] = meshgrid(x,y);
% или [Y,X] = ndgrid(y,x)
>> Z = sin(X*pi) + Y;
>> surf(x, y, Z);

Можно попробовать также:
>> mesh(Z);
>> meshz(Z);
>> surfl(z); shading interp;
>> colormap(pink);
>> contour(Z);
```



Параметры для **shading: interp, flat.**
Параметры для **colormap: hsv, jet, cool, hot, gray, pink, cooper.**

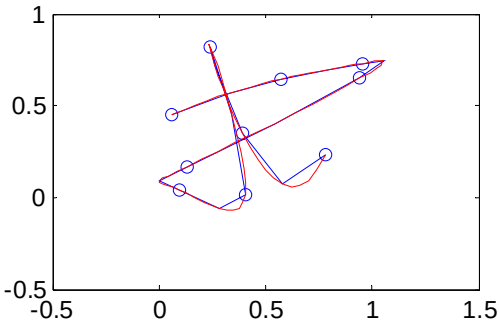
```
>> [x,y] = meshgrid(-2:0.2:2,-2:0.2:2);
>> z = x.*exp(-x.^2-y.^2);
>> [px,py] = gradient(z, 0.2, 0.2);
>> contour(z);
>> hold on;
>> quiver(px, py);
```

Построение поля направления функции.

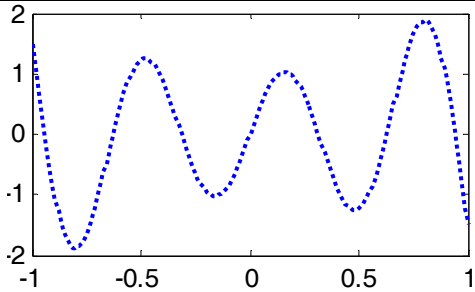


```
>> x = rand(10, 1); y = rand(10, 1);
>> plot(x, y, 'o');
>> hold on
>> xs = spline(1:10, x, 1:0.5:10);
>> ys = spline(1:10, y, 1:0.5:10);
>> plot(xs, ys, 'b');
>> xs = spline(1:10, x, 1:0.1:10);
>> ys = spline(1:10, y, 1:0.1:10);
>> plot(xs, ys, 'r');
```

Сплайны.

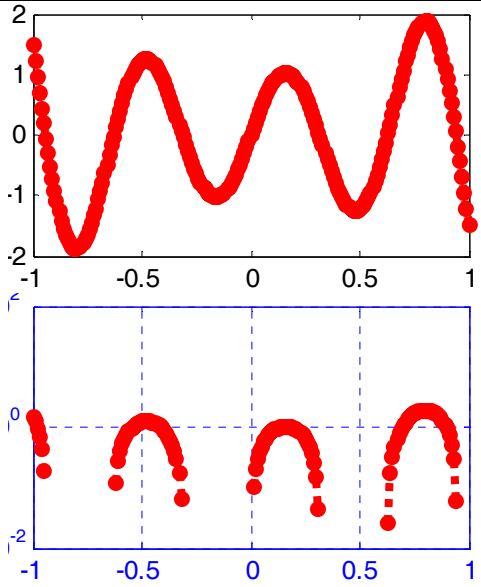


```
>> x = -1:0.01:1;
>> q = plot(x, sin(x*10).*exp(x.*x));
% «стиль» линии
>> set(q, 'LineStyle', ':');
% толщина
>> set(q, 'LineWidth', 2);


>> set(q, 'Marker', 'o');









>> set(q, 'Color', 'red');
>> set(q, 'LineWidth', 3, ...
```



Параметры для **LineStyle: '-', ':', '--', '-.'.**
Параметры для **Marker: 'o', 'x', '+', '*', '.', 's','d','^','v','h','p','>', '<'.**

```
          'MarkerSize', 3);



>> set(gca, 'XGrid', 'on', ...
            'YGrid', 'on');
>> set(gca, 'YScale', 'log');
Warning: Negative data ignored.
>> set(gca, 'XColor', 'blue', ...
            'YColor', 'blue');
```



Попробуйте применить команду **set(gca, 'fontsize', 10)** *и команду* **get(gca)**, *чтобы узнать, что ещё можно менять...*

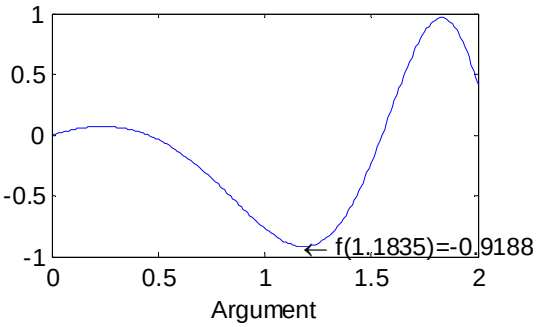
```
>> x = -5:0.01:5;
>> for j = 0:0.01:20
>> plot(x, sin(x+j))
>> drawnow
>> end

>> for j = 0:0.01:10
>> plot(x, cos(j)*exp(-x.*x));
>> set(gca, 'YLim', [-1 1]);
>> drawnow;
>> end
```

«*Плывущие волны*»

«*Вращающаяся петля*»

Задали диапазон изменения. Есть также функция **ylim**.

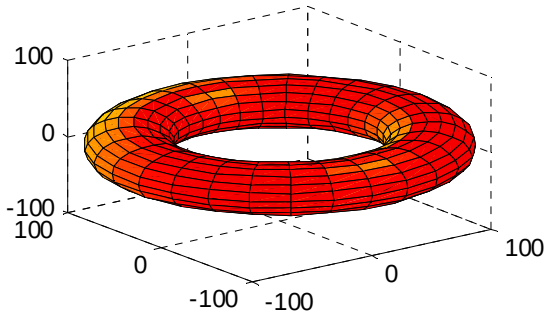
```
>> clf
>> f = @(t)cos(exp(t)).*sin(t);
>> T = 0:0.01:2;
>> [x,fm] = fminsearch(f,1);
>> s = strcat('\leftarrow f(',...
        num2str(x),')=',num2str(fm));
>> plot(T, f(T))
>> text(x, fm, s, 'Fontsize', 10);
>> xlabel('Argument')
```



Пример минимизации функции. Показан пример использования анонимной функции (см. ниже).

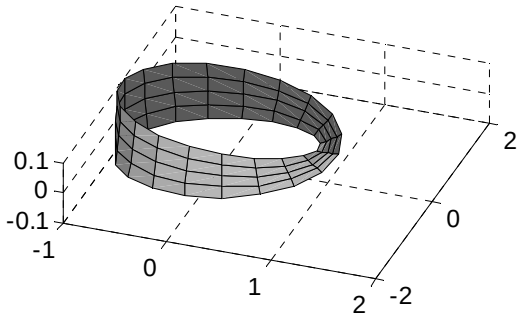
```
>> t = 0:0.1:30;
>> x = t.*sin(t);
>> y = t.*cos(t);
>> clf
>> plot3(x,y,t);
>> axis off % убрать оси
```

```
>> [A B] = meshgrid(linspace(0,2*pi,20),
linspace(0,2*pi,30)); % перенос!

>> X = (100+30*cos(A)).*cos(B);
>> Y = (100+30*cos(A)).*sin(B);
>> Z = 30*sin(A);
>> surfl(X, Y, Z);
>> axis([-100 100 -100 100 -100 100]);
>> colormap hot
```



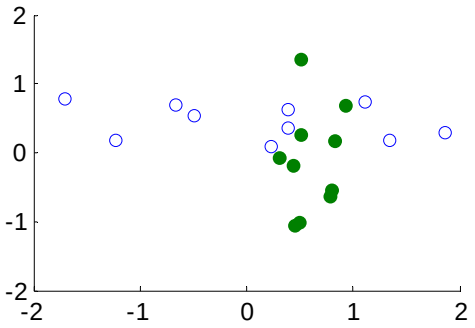
Тор.

```
>> [A B] = meshgrid(linspace(0,2*pi,20),
linspace(-0.1,0.1,5)); % перенос!

>> X = cos(A)+B.*cos(A/2).*cos(A);
>> Y = sin(A)+B.*cos(A/2).*sin(A);
>> Z = B.*sin(A/2);
>> surfl(X,Y,Z);
>> view(20,70);
>> colormap gray
```



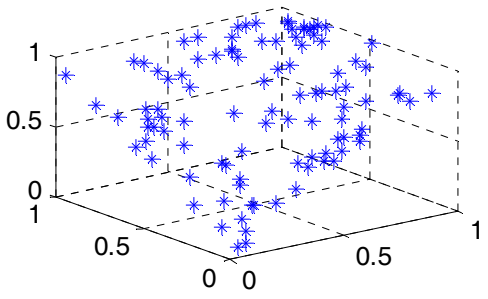
Лист Мебиуса.

```
>> x = randn([1 10]); y = rand([1 10]);
>> scatter(x, y);
>> hold on;
>> x = rand([1 10]); y = randn([1 10]);
>> scatter(x, y, 'filled');
```

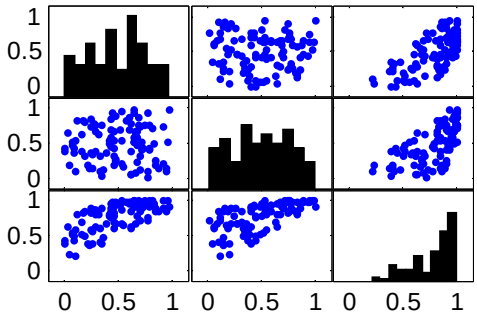


Визуализация данных (одна из самых полезных функций в анализе данных).

```
>> x = rand([1 100]);
>> y = rand([1 100]);
>> z = sin(x + y);
>> clf; scatter3(x,y,z);
```



Визуализация в трёхмерном пространстве. Попробуйте вращением картинки (**Rotate**

`>> figure` `>> plotmatrix([x; y; z]')`	**3D**) *убедиться, что данные лежат на «двумерной поверхности».* *Двумерные проекции множества точек.* 

`>> ezcontour('x^2-y^2', [-1 1], [-1 1])`
`>> ezsurf('1/(1+x^2+2*y^2)', ...`
`[-3 3], [-3 3])`
`>> ezplot('exp(3*sin(x)-cos(2*x))',...`
`[0 4])`
	Прямое построение графиков.

Сохранить графику можно, выбрав **File/Save** в меню окна **Figure**, при этом сохранение производится в специальном формате ***.fig**. Можно также выбрать **File/Generate M-File**, тогда будет сгенерирован код, который рисует изображение, но в коде не присутствуют данные (их придётся внести вручную). Для использования построенных графиков в документах редактора Word используйте буфер: в окне Figures выберите **Edit/Copy Figure**, затем переключитесь в Word-документ и произведите вставку из буфера (**Paste**). Для использования графики в TeX используйте команду **print**.

§15.11. Циклы

`>> A = 1; B = 2;`
`>> if A > B % если`
`>> '>'`
`>> elseif A < B % иначе если`
`>> '<'`
`>> else % иначе`
`>> '='`
`>> end % «если» завершается`
 	
`ans =`
`<`
	Пример условного оператора.
	
	В условном операторе следующие выражения эквивалентны:
	
	if (a>b)||(a>c),
	
	if or((a>b),(a>c)),
	
	if any([(a>b),(a>c)]).

`>> f = [1 1];`
`>> for n = 3:10 % что пробегает n`
`>> f(n) = f(n-1) + f(n-2);`
`>> end % конец цикла`
`>> f`
 	
`f =`
`1 1 2 3 5 8 13 21 34 55`
	Вывод чисел Фибоначчи. Попробуйте усовершенствовать этот код (см. дальше об эффективном программировании в MatLab).
	
	Переменная может пробегать столбцы матрицы. См., например, **for n = [1 2 3; 1**

	1 1], n, end; *Кстати, допустим такой цикл:* **for a=1:Inf, a, end;** *но присваивание* **a=1:Inf** *без цикла* **for** *не удастся. Цикл* **for a = [1:Inf], a, end;** *недопустим, поскольку квадратные скобки сообщают MatLaby, что надо сначала вычислить выражение в них.*
>> n **n =** **10**	*После выполнения цикла переменная-счётчик остаётся в памяти (это способ проверки, прошёл ли цикл до конца, правда, с одной тонкостью... какой?).*
>> S = 'qw12e3r4 56'; **>> n = 0;** **>> for s = S** **>> if isletter(s) continue;** **>> elseif s==' ' break;** **>> end; % конец if** **>> n = n + str2num(s);** **>> end; % конец for**	*Использование* **continue** *и* **break**. *Что делает этот код? Попробуйте его усовершенствовать.*
>> x = 10; **>> n = 0;** **>> while x > 1** **>> x = x/2; n = n+1;** **>> if n > 50, break, end** **>> end**	*Цикл* **while**.
>> switch isempty(A) **>> case 0** **>> 'непустой'** **>> case 1** **>> 'пустой'** **>> otherwise** **>> '?'** **>> end** **ans =** **непустой**	**switch** *Пример проверки на пустоту массива.*
>> x = 4; **>> switch x** **>> case {1,2}** **>> disp('1<=x<=2');** **>> case {3,4}** **>> disp('3<=x<=4');** **>> end** **3<=x<=4**	*Допустимы и такие конструкции...*
>> E = [];	*Посмотрите на действия*

`>> if E`
`>>  disp('Empty is true')`
`>> else`
`>>  disp('Empty is false')`
`>> end`
`Empty is false`
 	
`>> true || E`
 	
`ans =`
`  1`
 	
`>> true | E`
 	
`ans =`
`  []`
 	
`>> E || true`
`??? Operands to the || and && operators must be convertible to logical scalar values.`
	условного оператора с пустым множеством.

`>> [1 1 0 0] | [1 0 1 0]`
 	
`ans =`
`  1  1  1  0`
 	
`>> [1 1 0 0] || [1 0 1 0]`
`??? Operands to the || and && operators must be convertible to logical scalar values.`
	Разница между операциями **|** *и* **||**.
	
	Получаем логический массив.

`>> x = NaN;`
`>> if (x==x) disp('='); else disp('~');`
`end;`
`~`
	В MatLabe **(x==x)** *не всегда истина. Что будет при* **x = []** *и* **x = Inf***?*

В системе MatLab нет операторов безусловного перехода **goto** (как, впрочем, нет меток и номеров строк для перехода).

§15.12. Логические массивы

Логические массивы позволяют во многих случаях обходиться без операторов цикла. Для хорошего программирования в среде MatLab необходимо уметь применять этот тип данных (который отсутствует во многих других языках).

`>> A = [1,2,3;4,5,6;7,8,9];`
`>> L = logical([1,0,0;0,1,0;1,1,0]);`
 	
`>> A(L)'`
 	
`ans =`
`  1  7  5  8`
 	
`>> [i,j] = find(A>5 & A<9);`
`>> [i,j]'`
	Порождение логического массива.
	Вывод элементов, «помеченных» логическим массивом.
	
	Поиск индексов элементов, удовлетворяющих специальным

```
ans =
      3     3     2
      1     2     3
```

условиям. Сравните с командой **find(A>5 & A<9)** (*индексы в последовательной нумерации*). *Для перевода последовательной нумерации в обычную можно использовать* **[i,j] = ind2sub(size(A), find(A>5 & A<9))**.

```
>> find(isprime(1:20))

ans =
     2  3  5  7  11  13  17  19
```

Вывод всех простых чисел. **isprime** – *проверка на простоту (результат – логический массив).* **find** – *вывод индексов ненулевых элементов. Заметьте, что* **find(isprime(a))** *не выводит все простые числа из* **a**, *а только их индексы. Для вывода чисел используйте* **a(isprime(a))**.

```
>> X = [2,4,3,1];
>> X(X>2)

ans =
      4     3
```

Вывод всех элементов, которые больше 2. *Аналогичный результат получается при* **X(find(X>2))**.

```
>> X(X>2) = 2

X =
      2     2     2     1
```

Заменить на 2 *все элементы, которые больше двух* (*это не самый эффективный вариант решения такой задачи, см. ниже*).

```
>> whos L
  Name   Size   Bytes   Class
  L      3x3    9       logical array

Grand total is 9 elements using 9 bytes
```

```
>> L = L + 0;
>> whos L
  Name   Size   Bytes   Class
  L      3x3    72      double array

Grand total is 9 elements using 72 bytes
```

Обратите внимание на изменение типа при прибавлении нуля к логическому массиву (*для изменения типа в системе старше MatLab 5 рекомендуется использовать унарный плюс, см. ниже*).

```
>> [[1,2];[3,4]]==[[1;3],[2;4]]

ans =
      1     1
      1     1
```

В результате сравнения порождается логический массив (обратите внимание на различный способ порождения двух матриц).

Для проверки на равенство

	следует использовать **isequal([[1,2]; [3,4]], [[1;3], [2;4]])**. *Кстати, эта функция может работать со многими аргументами, см.* **isequal(2, 1+1, 3-1)**.
>> any(L) **ans =** **1 1 0**	*Есть ли в столбце ненулевой элемент (ответ для каждого столбца).* **all** *– проверка гипотезы, что в столбце все элементы ненулевые.*
>> x = [1, Inf, 2, NaN]; **>> x(finite(x))** **ans =** **1 2**	*Оставить в массиве только конечные числа.*
>> x = 1:10; **>> x(((x<3)|(x>6))&(x~=8))** **ans =** **1 2 7 9 10**	*Запомните:* **~=** *- не равно (отличие от языка C),* **|** *- логическое ИЛИ,* **&** *- логическое И.*
>> a = 5:6; **>> a([0 1])** **??? Subscript indices must either be real positive integers or logicals.** **>> a(logical([0 1]))** **ans =** **6**	*Тривиальный факт, но часто на этом ошибаются... Для логической индексации надо использовать логический массив (тип* ***logical****)! Запись* **a([1 2])** *допустима (это вывод первого и второго элемента), а запись* **a([0 1])** *формально выводит нулевой(?) и первый.*
>> a = 1:3; **>> b = a==2** **b =** **0 1 0** **>> b(1) = 2** **b =** **1 1 0** **>> b(3) = NaN** **??? NaN's cannot be converted to logicals.** **>> find(b == true)** **ans =**	*Присваивание элементам логического массива значений «не поддаётся логике». Обратите внимание, что в итоге получился логический массив. Кстати,* **(true==2)** *неверно!!!* *В MatLabe есть константы* **true** *и* **false**. *Правда, можно было бы писать* **find(b == 1)**.

```
       1     2
```

```
>> A = true(1,3) % так тоже можно!

A =
       1     1     1

>> B = ones(1,3)

B =
       1     1     1

>> isequal(A,B)

ans =
       1
```

Иллюстрация, что MatLab не различает **1** *и* **true**.

```
>> a = [-2 1 0 2 0 3];

>> +(a~=0) % первый способ

ans =
       1     1     0     1     0     1

>> +(~~a) % второй способ

ans =
       1     1     0     1     0     1
```

Решение задачи «сделать все ненулевые элементы матрицы единичными». Унарный «плюс» нужен для перевода в тип **double**!

Этот способ часто бывает чуть быстрее. Для замены неединичных элементов на единичные (в полной матрице больших размеров) не делайте так: **a(a~=0) = 1**, *лучше так:* **a = +(~~a)**. *Кстати, аналогичный результат даёт команда* **a = double(~~a)** (*выше использовалась операция «унарный плюс»*), *а вот операция* **a = (~~a)+0** *гораздо медленнее.*

§15.13. Оформление *.m-файлов

В m-файле можно сохранять последовательность MatLab-команд, которая будет выполняться, если в командной строке набрать имя соответствующего файла. «Запуск» скрипта (m-файла, содержащего последовательность команд) эквивалентен последовательному набору соответствующих команд. Если m-файл начинается с ключевого слова **function**, то в нём описана функция, для которой прописываются аргументы и значения (см. ниже).

Редактор для оформления m-файлов вызывается командой **edit**. Это обычный текстовый файл, поэтому может быть набран почти в любом текстовом редакторе. Файл следует сохранить в каталоге, который прописан в «путях доступа» (иначе скрипт/функция не будет вызываться по команде).

```
% QUADFORM решение квадратного уравнения
% Это содержимое файла quadform.m
function [x1,x2] = quadform(a,b,c)
```

Вызов такой функции, записанной в m-файле, осуществляется командой:

```
d = sqrt(b^2 - 4*a*c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
```

```
>> [r1,r2] = quadform(1,-2,1)
r1 =
     1
r2 =
     1
```

Обратите внимание на то, что переменная **d** *локальная (её не будет в рабочей области после завершения работы функции). Функция имеет свою область переменных.*

Запомните! Переменные в функциях – локальные, а в скриптах – глобальные!

```
% MPAR сравнение сигналов в 2х метриках
function [p1 p2] = mpar(x, y)
x = mmean(x);
y = mmean(y);
p1 = sum(abs(x-y));
p2 = sqrt(sum((x-y).^2));

% MMEAN вычесть среднее
function x = mmean(x)
x = x - mean(x);

%{
код для иллюстрации
вложенных функций
%}
```

Внутри функции может быть определена другая функция. Такие функции называются вложенными (nested). Вызов **mmean([1 2 3])** *«не работает»! Работают только вызовы* **[p1 p2] = mpar(x, y)** *и* **mpar(x, y)** *(если содержимое сохранено в файле* **mpar.m**).

Так выделяется целый блок комментария.

```
>> f = memoize1(@sin)

f =
     @memoize1/inner

>> f(0)
new

ans =
      0

>> f(0)

ans =
      0

>> f(1)
new

ans =
      0.8415
```

Сохраните в m-файл следующую функцию [Loren]:

```
function f = memoize1(F)
x = [];
y = [];
f = @inner;
    % nested function
    function out = inner(in)
        ind = find(in == x);
        if isempty(ind)
            out = F(in);
            x(end+1) = in;
            y(end+1) = out;
            disp('new');
        else
            out = y(ind);
        end
    end
end
```

Теперь мы сообщаем, какой функцией у нас будет **f** *(в*

```
>> f(0)

ans =
      0

>> s = functions(f)

s =
     function: 'memoize1/inner'
         type: 'nested'
         file: [1x62 char]
    workspace: {[1x1 struct]}

>> s.workspace{1}

ans =
      f: @memoize1/inner
      F: @sin
      x: [0 1]
      y: [0 0.8415]
```

данном случае **sin**). *При обращении к ней вычисляется значение этой функции или, если к ней уже обращались с таким аргументом, значение просто берётся «из памяти».*

Здесь показана «память» функции.

Попробуйте написать аналогичную функцию с памятью, к которой можно обращаться с вектором значений.

```
>> profile on;
% <<вызов функций>>
>> profile report
>> profile off
```

Указывает время выполнения функций и время выполнения каждой строки(!) каждой вызываемой функции.

Если в тексте m-файла встречается команда **keyboard**, то выполнение скрипта прекращается на время. Пользователь может вводить команды в командном окне. При вводе **return** продолжается выполнение m-файла. При использовании команды **return** в m-файле прекращается выполнение функции/скрипта. Зарезервированные имена переменных:

nargin – количество параметров, с которыми была вызвана функция,

nargout – количество выходных параметров, с которыми была вызвана функция.

Команда **global a b** сообщает функции о двух глобальных переменных: **a** и **b**. Заголовок **function myfunc(a, b, varargin)** описывает функцию с двумя аргументами **a, b** и, возможно, ещё какими-то, записанными в массиве ячеек[1] **varargin** (это ключевое слово). Для переменного числа выходных параметров используют имя массива ячеек **varargout**.

Многие функции системы MatLab (**sqrt**, **sin** и т.д.) являются встроенными (built in). Они очень эффективно выполняются, их выполнение нельзя прервать (нажатием Ctr+C), и их код недоступен (это функции ядра).

§15.14. Структуры

```
>> a = f([1, 1])

a =
         Value: 1
      Gradient: [2x1 double]
```

Сохраните в файл **f.m**

```
function fx = f(x)
fx.Value = (x(1)-
1)^2+x(1)*x(2);
fx.Gradient = [2*(x(1)-1) +
x(2); x(1)];
```

[1] Про массивы ячеек см. ниже.

```
>> a.Gradient

ans =

     1
     1
```

Значение поля.

```
>> fieldnames(a)

ans =
    'Value'
    'Gradient'
```

Вывод всех названий полей.

```
>> isfield(a, {'Value', 'Trace',
'Gradient'})

ans =
     1     0     1
```

Проверка, есть ли такие поля в структуре.

```
>> student.name = 'Ivanov';
>> student.mark = 5;
>> student(2).name = 'Petrov';
>> student(2).mark

ans =
     []
```

Массив структур с именами студентов и отметками.

Поле **mark** *у второго элемента уже существует! Можно было бы заполнить второй элемент так:* **student(2) = struct('name','Petrov', 'mark',[])**.

```
>> student(3).name = 'Sidorov';
>> student(3).mark = 4;
>> [student.mark]

ans =
     5     4
```

Массив всех оценок. Поскольку **student.mark** *является массивом ячеек (см. ниже). Данный вызов работает и в случае, если* **student(1).mark = [5 4 5 5]**, *но не в случае* **student(1).mark = [5 4 5 5]'** *(с транспонированием).*

```
>> [student.mark] = deal(5, 4, 4)

student =

1x3 struct array with fields:
    name
    mark
```

Присваивание значений (проставили оценки сразу всем студентам). Попробуйте теперь вызвать **[student.mark]**.

```
% сразу несколько оценок
>> [student(1:3).mark] = ...
              deal([5,4], [4,3,5], [4])

student =

1x3 struct array with fields:
```

Кстати, если сначала сделать присваивания **student.name = 'Ivanov'**, **student(2).name = 'Petrov'**, *то присваивание* **student.mark = 5** *некорректно. Верная запись -* **student(1).mark = 5**.

```
    name
    mark

% второй способ
>> s = {[5,4], [4,3,5], [4]}

s =
   [1x2 double]  [1x2 double]  [4]

>> [student(1:3).mark] = s{1:3}

student =

1x3 struct array with fields:
    name
    mark

>> fields = fieldnames(s)

fields =
         'name'
         'age'

>> field1 = fields(1)

field1 =
         'name'

snames = {student.(field1{:})}

snames =
   'Ivanov'   'Petrov'   'Sidorov'
```

Присваивание не через **deal**. *Сначала порождаем массив ячеек (см. ниже). Кстати, присваивание* **[student(1:3).mark] = {[5,4],[4,3],[4]}** *не получится.*

Дальше идёт иллюстрация одного способа доступа к полям... Сначала получили все имена полей.

Выбрали первое.

Вывели все значения этого поля... Команда **student.(field1{:})** *работает! Заметим, что команда* **snames = [student.(field1{:})]** *создаст строку* **'IvanovPetrovSidorov'**.

```
>> s(1).x = 1;
>> s(2).x = 2;
>> a = struct2cell(s); % перевод в
массив ячеек
>> whos
  Name  Size  Bytes  Class  Attributes

  a     1x1x2 136    cell
  s     1x2   200    struct
```

Обратите внимание, что «первая размерность структуры» - размерность имён полей.

```
>> s1.name = 'Alex';
>> s1.age = 20; % попробуйте поменять
местами эту строчку со следующей
>> s1(2).name = 'Serg';
>> s1(2).age = 19;
>> s2.mark = 5;
>> s2(2).mark = 4;
>> s = cell2struct([struct2cell(s1);
struct2cell(s2)], [fieldnames(s1);
```

Пример как объединять структуры (взято из [Loren]).

Работает, если все имена полей уникальны!

```
fieldnames(s2)],1)

s =

1x2 struct array with fields:
    name
    age
    mark

>> s(1)

ans =
      name: 'Alex'
       age: 20
      mark: 5
```

Вывод первого элемента массива структур (теперь здесь три поля).

```
>> field = 'f';
>> for i = 1:3;
>> a.([field int2str(i)]) = i; end;
>> a

a =
      f1: 1
      f2: 2
      f3: 3
```

Один из способов создания полей и присваивания им значений (динамические имена полей - Dynamic Field References).

§15.15. Встроенные и анонимные функции

Для создания таких функций не нужен отдельный m-файл.

```
>> f = inline('sin(pi*x)+cos(pi*y)+z',
'z', 'x', 'y')

f =
    Inline function:
    f(z,x,y) = sin(pi*x)+cos(pi*y)+z

>> f(1,2,3)

ans =
     -2.2204e-016

>> formula(f)

ans =
      sin(pi*x)+cos(pi*y)+z
```

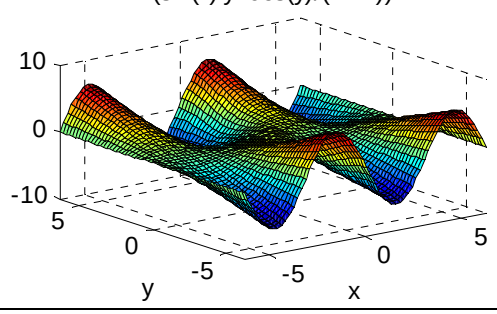
Объявление встроенной функции.

Обратите внимание: x=2, y=3, z=1!
Попробуйте **f([1,2,3]).**

Вывод формулы. Попробуйте также **argnames(f)** *и* **methods(f).**

```
>> f=@(x,y) (sin(x)*y+cos(y)/(x^2+1));
>> ezsurf(f);
```

Анонимная функция.



```
>> f = @(A,x) A(x)

f =
      @(A,x)A(x)

>> A = [1 2 3; 4 5 6; 7 8 9]

A =
       1     2     3
       4     5     6
       7     8     9

>> f(A,[1 2; 3 4])

ans =
       1     4
       7     2
```

Создание анонимной функции, которая возвращает элементы матрицы **A**.

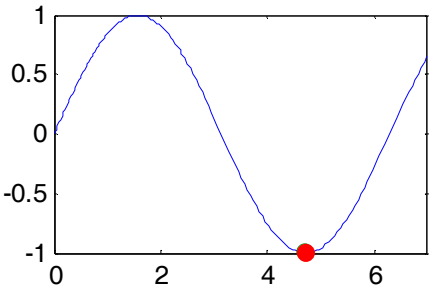
Напомним, что MatLab нумерует элементы по столбцам. Подобная функция полезна для выражений типа **f((triu(ones(3))+diag([1 2 3]))^2, 4)**, *поскольку вызов* **((triu(ones(3))+diag([1 2 3]))^2)(4)** *некорректен.*

```
>> f=@(x) (sin(x)+cos(x)-x);
>> ezplot(f, [0 5])
>> grid on
>> fsolve(f, 3)
Optimization terminated: first-order
optimality is less than options.TolFun.
ans =
      1.2587
```

Решение нелинейного уравнения **f(x)=0**. *Можно просто вызвать* **fsolve('sin(x)+cos(x)-x',3)** (*указывается начальное приближение*).

```
>> h = @sin;
>> x = 0:0.01:7;
>> fplot(h, [0 7]) % plot(x,h(x))
>> fminbnd(h, 0, 7)

ans =
      4.7124

>> hold on;
>> scatter(ans, h(ans), 40,'filled')
```

Пример использования указателя на функцию в задаче минимизации.



Попробуйте минимизировать свою функцию.

§15.16. Массивы ячеек

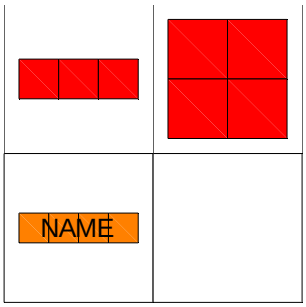
Этот тип данных позволяет хранить в одном массиве элементы разных типов.

```
>> c = cell([2 2])

c =
      []    []
      []    []

>> c{1, 1} = [1, 2, 3];
>> c{1, 2} = eye([2, 2]);
>> c{2, 1} = 'NAME';
>> cellplot(c)
```

Порождения массива ячеек.

Заполнение его содержимым. Оно разнородное (разных типов)!

Результат оператора **cellplot**:




```
>> celldisp({datestr(now), now})

ans{1} =
         26-Feb-2005 14:38:29

ans{2} =
         7.3237e+005
```

Вывод содержимого массива ячеек.

```
>> datevec(datestr(now)) % г м чис ч м с

ans =
      2005    2    26    14    38    45
```

Работа с датой и временем.

```
>> T = cell([1,10]);
>> T{1} = [1];
>> for i = 2:10;
>> T{i} = [T{i-1},0]+[0,T{i-1}];
>> end;
```

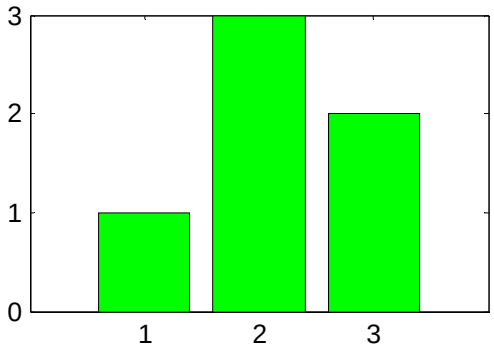
Задание треугольника Паскаля. Попробуйте выполнить **celldisp(T).**

```
>> a = {1,[2 3],[4;5]}

a =
    [1]    [1x2 double]    [2x1 double]

>> a(1:2)

ans =
      [1]    [1x2 double]

>> [a{1:2}]

ans =
       1     2     3

>> arg = {[1 3 2],'g'}

arg =
      [1x3 double]    'g'

>> bar(arg{:})
```

Пример разницы в индексации массивов ячеек.

Срез массива.

Список значений. Посмотрите, что выдаёт команда **a{1:2}.**



Обратите внимание: гистограмма вывелась зелёным цветом.

```
>> A = {{[1],[2 3]}, 4}

A =
    {1x2 cell}    [4]

>> A(end+1,:) = {1, 2}

A =
    {1x2 cell}    [4]
    [       1]    [2]

>> A{2,2} = [1 2]

A =
    {1x2 cell}    [         4]
    [       1]    [1x2 double]

>> A(1,:) = []

A =
    [1]    [1x2 double]
```

Элементами массива ячеек могут быть массивы ячеек.

Присваивание **A{end+1,:} = {1,2}** *невозможно!*

А здесь не пройдёт присваивание **A(2,2) = [1 2]** (*с круглыми скобками*). *Можно сделать так:* **A(2,2) = {[1 2]}.**

Нельзя написать **A{1,:} = []**.

```
>> X = [1 2 3; 4 5 6; 7 8 9];
>> a = {[1 3], ':'}

a =
      [1x2 double]    ':'

>> a{:}

ans =
       1     3

ans =
       :

>> X(a{:})

ans =
       1     2     3
       7     8     9
```

Иллюстрация приёма, который используется для индексации массива.

Вывод подматрицы матрицы **X**, *определяемой массивом ячеек* **a**.

```
>> clear; a = 1; b = 2; X = [0 1];
>> vars = who'

vars =
       'X'    'a'    'b'

>> class(vars)

ans =
      cell
```

Некоторые средства MatLab позволяют получить результат в виде массива ячеек. В этом примере получен массив имён всех переменных.

```
>> c(1:4,1) = {'', 'Alex', 'Den',
'Serg'};
>> c(1:4,2) = {'MATH', 5, 4, 5};
>> c(1:4,3) = {'ECON', 5, 3, 4}
```

Пример показывает, как из массива ячеек выделять вещественные подматрицы.

```
c =
        ''    'MATH'    'ECON'
    'Alex'    [   5]    [   5]
    'Den'     [   4]    [   3]
    'Serg'    [   5]    [   4]

>> reshape([c{2:end,2:end}], size(c,1)-
1, size(c,2)-1)

ans =
      5     5
      4     3
      5     4
```

```
>> [b{1:3}] = deal(1, 2, 3)

b =
      [1]    [2]    [3]

>> [a{:}] = b{:}
??? The left hand side has a{:} inside
brackets, which requires that a be
defined, so that the number of expected
results can be computed.

>> [a{1:3}] = b{:}

a =
      [1]    [2]    [3]
```

Присваивания с помощью **deal**.

Обратите внимание на эту ошибку!

§15.17. Строки

```
>> str = ['12 3', ' ', '4']

str =
      12 3 4

>> double(str)

ans =
      49  50  32  51  32  52

>> char(ans)

ans =
      12 3 4

>> A = str2mat('1',upper('ab'),'456''7')

A =
    1
    AB
    456'7

>> findstr(A(3,:),'''')
```

Порождение строки.

Преобразование в массив чисел.

Обратное преобразование.

Создание матрицы строк. Происходит «пополнение» пробелами (попробуйте **double(A)**). *Обратите внимание на вставку апострофа.*

Определение позиции

ans = 4	апострофа. Вообще, рекомендуется использовать функцию **strfind**, в которой важен порядок аргументов, но не функцию **find(A(3,:)=='''')**.
>> num2str(eval('1+1')) ans = 2	Создание строки **'2'**. Запомните функцию **eval**! Она «исполняет строку». Попробуйте, например, **eval('plot(sin(1:10))')**. Некоторые задачи можно решать «компоновкой подходящей строки», которая затем передаётся функции **eval**.
>> isletter('1a2bcd3') ans = 0 1 0 1 1 1 0	Определение позиций букв.
>> strmatch('ab', ['abc';'bca';'aaa']) ans = 1	Строки, которые начинаются с **'ab'**.
>> num2str(10.11111111111111) ans = 10.1111	Будьте осторожны при использовании **num2str**, поскольку эта функция «обрубает числа»!
>> findstr([0 1 2 0 0 3 4 1 1 2],[1 2]) ans = 2 9	Строковые функции можно применять и для обработки «нестроковых» данных...
>> strrep('Good boy and good girl', 'good', 'bad') ans = Good boy and bad girl	Замена одной подстроки другой (можно и нужно использовать для удаления подстрок).
>> strncmpi({'Good', 'boy', 'good', 'girl'}, 'GOOD', 4) ans = 1 0 1 0	Сравнение строк без учёта регистра.
>> strncmp({'Good', 'boy', 'bad', 'girl'},'b',1) ans = 0 1 1 0	Какие строки начинаются на букву «b»?
>> str = 'Value'; >> str + str	При сложении строк результат

```
ans =
     172   194   216   234   202

>> [str str]

ans =
     ValueValue

>> val = 3;
>> sprintf('str = %s, val = %d or %f, pi
= %2.3f', str, v, v, pi)

ans =
str = Value, val = 99 or 99.000000, pi =
3.142
```

double!

Конкатенацию следует производить так.

sprintf - *удобное средство для вывода информации на экран.*

```
>> X = [1 5 7];
>> n = 3;
>> eval(['[ ' sprintf('A%d ',1:n) '] =
ndgrid( ' repmat('X, ',1,n-1) 'X );']);
>> eval(['A = [' sprintf('A%d(:) ',1:n)
']']);

A =
     1     1     1
     5     1     1
     7     1     1
     1     5     1
     5     5     1
     7     5     1
     1     7     1
     5     7     1
     7     7     1
     1     1     5
     5     1     5
     7     1     5
     1     5     5
     5     5     5
     7     5     5
     1     7     5
     5     7     5
     7     7     5
     1     1     7
     5     1     7
     7     1     7
     1     5     7
     5     5     7
     7     5     7
     1     7     7
     5     7     7
     7     7     7
```

Пример того, как с помощью функций **sprintf** *и* **eval** *можно решать задачи любой сложности методом «формирования строки с нужной командой». Решается следующая задача. Для любого множества* **X** *и произвольного натурального* **n>=2** *надо построить матрицу, в которой по строкам перечислены все элементы множества* **X** *в декартовой степени* **n**.

Команда **['[' sprintf('A%d ',1:n) '] = ndgrid(' repmat('X, ',1,n-1) 'X);']** *формирует строку* **'[A1 A2 A3] = ndgrid(X, X, X);'** , *которую исполняет команда* **eval**. *Аналогично, следующая команда формирует строку* **'A = [A1(:) A2(:) A3(:)]'**.

```
>> G = repmat({X}, n, 1);
>> [G2{1:n}] = ndgrid(G{:});
```

Второй способ решения задачи. Через массивы ячеек

```
>> A = [G2{1:n}];
>> s = size(A);
>> s = [s(1) length(X) n s(3:end)];
>> A = reshape(A, s);
>> per = 1:ndims(A); per(3) = ndims(A);
>> per(end) = 3;
>> A = permute(A, per);
>> A = reshape(A, [length(X).^n n] )
```

и фокусы с размерностями. Попробуйте понять, какой из этих двух способов эффективнее.

```
>> ismember({'a','b','c'},{'a','m','n'})

ans =
      1     0     0
```

Для массива ячеек, наполненного строками, можно использовать **ismember** (*попробуйте заменить одну из строк на число*).

```
>> S = [];

>> for s = {'one','two','three'}
>> S = [S ' ' s{:}]
>> end

S =
    one
S =
    one two
S =
    one two three
```

Цикл может пробегать и массив ячеек.

Обратите внимание, что **s** *– ячейка! Попробуйте заменить эту строку на* **S = [S ' ' s].**

```
>> files = dir('*.mat');
>> for I = 1:length(files)
>> thefile = files(i).name
>> % действия с файлом
>> end;

>> for i = 1:9
>> thefile = sprintf('mat%d.mat', k);
>> % другой способ
>> thefile = ['mat' num2str(k) '.mat'];
>> data = load(thefile);
>> end;
```

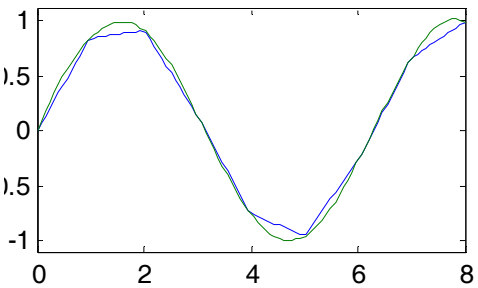
Так перебираются файлы в каталоге.

Так перебираются файлы **mat1.mat, mat2.mat** *и т.д.*

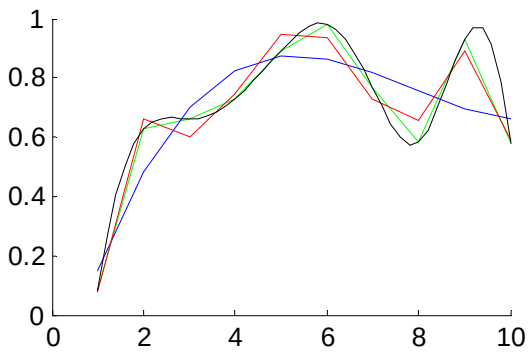
§15.18. Интерполяция и полиномы

```
>> x = 0:8; y = sin(x);
>> xi = linspace(0, 8, 100);
>> yiL = interp1(x, y, xi);
>> yiC = interp1(x, y, xi, 'spline');
>> plot(xi, [yiL; yiC]')
```

Пример линейной интерполяции и интерполяции сплайнами.

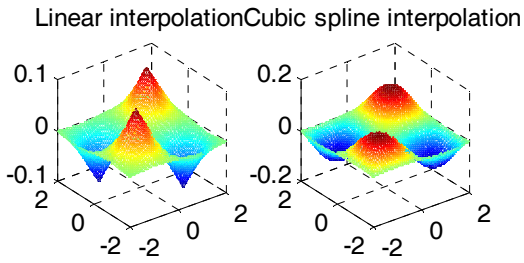


```
>> x = 1:10;
>> y = rand(size(x));
% коэф. аппрокс. полинома 3-ей степени
>> p = polyfit(x, y, 3);
% вычисление его значений
>> f = polyval(p, x);
>> hold on;
>> plot(x, y, 'green');
>> plot(x, f, 'blue');
>> p = polyfit(x, y, 6);
>> f = polyval(p, x);
>> plot(x, f, 'red');

% аппрокс. кубическими сплайнами
>> g = spline(x, y, 1:0.2:10);
>> plot(1:0.2:10, g, 'black');
```



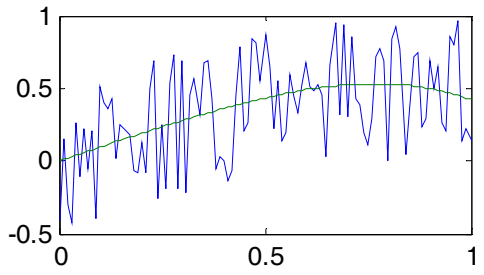

```
>> x = -2:2;
>> y = [-2:2]';
>> z = (sin(y).*exp(-
y.^2))*(sin(x).*exp(-x.^2));
>> xi = linspace(-2,2,100);
>> yi = linspace(-2,2,100)';
>> ziL = interp2(x,y,z,xi,yi);
>> subplot(1,2,1), mesh(xi,yi,ziL)
>> title('Linear interpolation')
>> ziC = interp2(x,y,z,xi,yi,'splnin');
>> subplot(1,2,2), mesh(xi,yi,ziC)
>> title('Cubic spline interpolation')
```

Пример двухмерной интерполяции.




```
>> t = 0:0.01:1;
>> f = cos(t) + t + rand(size(t)) - 1.5;
>> t = t'; f = f';
>> F = [sin(t), sin(2*t), sin(3*t)];
>> a = F\f;
>> g = a(1)*F(:,1) + a(2)*F(:,2) + ...
      a(3)*F(:,3);
>> plot(t, [f,g])
```

Приближение функцией **A*sin(t)+B*sin(2*t)+C*sin(3*t).**




```
>> X = conv([1,-2,3], [1,-1])

X =
     1    -3     5    -3

>> roots(X)

ans =
      1.0000 + 1.4142i
      1.0000 - 1.4142i
      1.0000
```

Перемножение полиномов

Корни полинома.

`>> polyder(X)`
 	
`ans =`
`3 -6 5`
	Производная.

§15.19. Поэлементные операции

`>> arrayfun(@abs, [-1 0;1 -2])`
 	
`ans =`
`1 0`
`1 2`
	Поэлементная операция абсолютного значения над матрицей. Поэлементно можно выполнять любую функцию одного аргумента.

`>> Y = repmat(X, length(p), 1).^repmat(p', 1, length(X))`
 	
`Y =`
`1 1 1 1`
`2 3 4 5`
`4 9 16 25`
`8 27 64 125`
 	
`>> Y2 = cell2mat(arrayfun(@(z)X.^z, p', 'uniformoutput', false))`
 	
`Y2 =`
`1 1 1 1`
`2 3 4 5`
`4 9 16 25`
`8 27 64 125`
	Два способа построения матрицы Вандермонда. Элементы множества **X = 2:5** *возводятся во все степени* **p = 0:3**.

`>> A = fix(8*rand(3))`
 	
`A =`
`1 5 2`
`5 6 1`
`3 7 5`
 	
`>> A = bsxfun(@minus, A, min(A,[],2))`
 	
`A =`
`0 4 1`
`4 5 0`
`0 4 2`
 	
`>> A = bsxfun(@rdivide,A,max(A,[],2))`
 	
`A =`
`0 1.0000 0.2500`
`0.8000 1.0000 0`
`0 1.0000 0.5000`
	Решение задачи нормировки. Построчно-линейным преобразованием перевести максимальный элемент каждой строки в **1**, *а минимальный – в* **0**. *Функция* **bsxfun** *позволяет действовать построчно (не используя* **repmat**).
	
	Запомните:
	`>> bsxfun(@rdivide,[10 5;8 6],[5; 2])`
	
	`ans =`
	`2 1`
	`4 3`
	
	`>> bsxfun(@rdivide,[10 5;8 6],[5, 2])`
	
	`ans =`
	`2.0000 2.5000`
	`1.6000 3.0000`

`>> [i,j] = meshgrid(1:1000, 1:1000);`	*Два решения задачи вычисления всевозможных длин гипотенуз*

```
>> hypos = @(i,j) sqrt(i.^2+j.^2);

>> tic; H = sqrt(i.^2+j.^2); toc

Elapsed time is 0.141593 seconds.

>> tic; H = bsxfun(hypos, i, j); toc

Elapsed time is 0.125817 seconds.
```

прямоугольных треугольников с длинами катетов от **1** *до* **1000**.

Функция **tic** *делает «временную засечку», а* **toc** *сообщает, сколько времени прошло с момента этой засечки.*

Ещё одно полезное применение функции **bsxfun**, *причём быстрое!*

```
>> S = {'djakonov@mail.ru',
'mmp@cs.msu.ru', 'mmp@cs.msu.su'}

S =
  'djakonov@mail.ru'   'mmp@cs.msu.ru'
'mmp@cs.msu.su'

>> f = @(x) x(find(x=='@')+1:end)

f =
    @(x)x(find(x=='@')+1:end)

>> Sr = cellfun(f, S, 'UniformOutput',
false)

Sr =
    'mail.ru'  'cs.msu.ru'  'cs.msu.su'

>> any(strcmp(Sr, 'google.com'))

ans =
     0
```

Решение задачи: оставить в перечне e-mail-адресов только соответствующие домены.

Функция, которая «отбрасывает лишнее».

Полезная функция **cellfun** (*сделать что-то с каждой ячейкой, аналог функции* **arrayfun**). *В итоге получается массив ячеек. Заметим, что после выполнения* **cellfun(@length,S)** *получается обычный массив.*

Для определения, есть ли среди строк, содержащихся в массиве ячеек, заданная, можно использовать такую запись.

```
>> S = sparse(fix(3*rand([2 3])))

S =
    (2,1)        2
    (1,2)        1
    (2,2)        1
    (1,3)        2
    (2,3)        2

>> f = spfun (@exp,S)

f =
    (2,1)        7.3891
    (1,2)        2.7183
    (2,2)        2.7183
    (1,3)        7.3891
    (2,3)        7.3891
```

Операция над всеми ненулевыми элементами разреженной матрицы. Очень полезна при работе с такими матрицами (*хотя использование* **spfun** *иногда неэффективно, см. дальше*).

```
>> S.n1 = 'Name'; S.n2 = 'Mark'

S =
      n1: 'Name'
      n2: 'Mark'

>> structfun(@upper, S, 'UniformOutput',
false)

ans =
      n1: 'NAME'
      n2: 'MARK'
```

Поэлементная функция для структур. В данном примере названия всех полей переводятся в верхний регистр.

```
>> A = cell(2);
>> [A{:}] = deal(0)

A =
      [0]    [0]
      [0]    [0]
```

Способ инициализации массива ячеек (автор Oliver Woodford [Loren]), кстати, более быстрый, чем **num2cell(zeros(2))**.

```
>> funcs = repmat({'sin' 'cos' 'tan'
'log' 'exp'}, 1, 1000);

% 1й способ
>> tic;
>> for f = funcs
>> func = eval(['@' f{1}]);
>> f = func(1.0);
>> end
>> toc

Elapsed time is 0.467254 seconds.

% 2й способ
>> tic;
>> for f = funcs
>> func = str2func(f{1});
>> f = func(1.0);
>> end
>> toc

Elapsed time is 0.142494 seconds.

% изменение индексации во 2ом способе
>> tic;
>> for i = 1:numel(funcs)
>> func = str2func(funcs{i});
>> f = func(1.0);
>> end
>> toc

Elapsed time is 0.139898 seconds.

>> tic;
>> funcs = cellfun (@(x) ['@' x], funcs,
'UniformOutput', false); % переделка
```

Есть перечень функций. В данном случае мы повторили **'sin', 'cos', 'tan', 'log', 'exp'** 1000 *раз. Рассмотрим рекомендации* [Loren] *по вычислению значений функций из перечня в фиксированной точке.*

Самый плохой (долгий) способ - вызывать **eval**.

Если есть только перечень названий функций, то надо использовать **str2func**.

Заметим, что при вызове **cellfun(@(x) ['@' x], {'sin' 'cos'},'UniformOutput',false)** *ответ*

```
ans =
      '@sin'    '@cos'
```

Если есть возможность хранить не массив названий, а массив

```
данных и типов (получаем массив ячеек)
>> toc, tic;
>> for i = 1:numel(funcs)
>>    f = funcs{i}(1.0); % и всё!!!
>> end
>> toc

Elapsed time is 0.111768 seconds.
Elapsed time is 0.028202 seconds.
```

ячеек с указателями, то вычисления получаются сверхэффективными и эффектными.

Переделка в массив ячеек занимает время.
Зато вычисления потом происходят мгновенно!

§15.20. О скорости…

Используйте функции, а не скрипты!

- *Скрипты могут вызывать только функции (но не скрипты).*
- *Функции сразу компилируются в память и хранятся там.*

```
% первый фрагмент
>> clear; tic;
>> A = rand(100);
>> y = ones(100, 1);
>> dt = 0.001;
>> for n = 1:(1/dt)
>> y(:,n+1) = y(:,n) + dt*A*y(:,n);
>> end;
>> toc;

Elapsed time is 1.641000 seconds.

% второй фрагмент
>> clear; tic;
>> A = rand(100);
>> y = ones(100, 1001);
>> dt = 0.001;
>> for n = 1:(1/dt)
>> y(:,n+1) = y(:,n) + dt*A*y(:,n);
>> end;
>> toc

Elapsed time is 0.094000 seconds.
```

Сразу выделяйте память!

Сравните эти два фрагмента кода. Объясните разницу в скорости вычислений.

Часто заранее неизвестно, сколько памяти нужно выделять. Тогда поступают примерно так...

```
X = zeros(1,n);
for i = 1:n
   % какое-то присваивание
   X(i) = f(i);
   % выход по какому-то условию
   if b(i)
      break;
   end
end
% оставить только то, что надо
b = b(1:ind);
```

Хуже писать в теле цикла `X = [X f(i)]` *или* `X(end+1) = f(i)`.

Не изменяйте размеры матриц в циклах!

```
>> A = int8(zeros(100)); % плохо
>> A = zeros(100, 'int8'); % хорошо
```

Сразу создавайте данные нужного типа!

```
>> clear; tic; A = rand(1000);
>> for i = 2:size(A,1)
>> for j = 1:size(A,2)
>> A(i,j) = A(i-1,j) + A(i,j);
>> end
>> end; toc

Elapsed time is 2.797000 seconds.
```

Старайтесь использовать векторные вычисления!

```
>> clear; tic; A = rand(1000);
>> for i = 2:size(A,1)
>> A(i,:) = A(i-1,:) + A(i,:);
>> end; toc

Elapsed time is 0.156000 seconds.
```

Обратите внимание на время вычислений в этих трёх фрагментах кода.

```
>> clear; tic; A = rand(1000);
>> cumsum(A); toc

Elapsed time is 0.094000 seconds.
```

Используйте встроенные функции!

```
>> clear;
>> A = ones(1000,1000);
>> B = zeros(1000,1000);
>> C = zeros(1000,1000);
>> tic,
>> for n = 1:1000
>> for m = 1:1000
>> C(n,m) = A(n,m) + B(n,m);
>> end
>> end
>> toc
>> tic, D = A + B; toc

Elapsed time is 2.484000 seconds.
Elapsed time is 0.031000 seconds.
```

Ещё один пример. «Встроенное суммирование» эффективнее. Попробуйте убрать строчку

C = zeros(1000,1000);

(*приготовьтесь ждать очень долго*). *Кстати, вычисления можно остановить нажатием* **Ctrl+C**.

```
>> tic; s = 0;
>> for i = 1:10000
>> if (isprime(i)) s = s+i;
>> end;
>> end; s, toc;

s =
      5736396
Elapsed time is 0.465790 seconds.
```

Используйте маски (логические массивы)!

Задача: *найти сумму простых чисел от* 2 *до* 10000.

```
>> tic; sum( find( isprime(1:10000) ) ),
toc;

ans =
      5736396
Elapsed time is 0.046846 seconds.
```

Решение с помощью векторизации и логических массивов.

```
>> A = rand(1000,1000);
>> b = rand(1000,1);
```

Не вызывайте лишних функций!

```
>> tic, x1 = inv(A)*b; toc

Elapsed time is 1.516000 seconds.

>> tic, x2 = A\b; toc

Elapsed time is 0.578000 seconds.

>> D = diag(1:1000);
>> tic; D2 = inv(D); toc

Elapsed time is 0.962427 seconds.

>> tic; D3 = diag(1./diag(D)); toc

Elapsed time is 0.010591 seconds.

>> A = rand(1000);
>> tic; p1 = sum(1./eig(A)); toc

Elapsed time is 6.841646 seconds.

>> tic; p2 = trace(inv(A)); toc

Elapsed time is 1.125631 seconds.

>> sum(abs(p1 - p2))

ans =
      8.8818e-014
```

Не надо инвертировать матрицу, если этого можно избежать...

Инвертирование диагональной матрицы также лучше делать другим способом.

Правда, есть случаи, когда «традиционно плохие» функции (типа **inv***) могут быть очень полезны. Вот пример такой задачи (замечено Greg von Winckel в* [Loren]*).*

Результаты «почти одинаковые» (из-за специфики операций с плавающей точкой, см. ниже).

```
>> A = rand(2000);
>> [X1 X2 X3 X4 X5] = deal(zeros(2000));
% сразу выделили память, чтобы на это не
тратилось время
>> s = sum(A,2); % суммы всех строк

>> tic; X1 = diag(1./s)*A; toc

Elapsed time is 6.822780 seconds.

>> tic; X2 = A./repmat(s,1,size(A,2));
toc

Elapsed time is 1.875061 seconds.

>> tic; X3 = A.*repmat(1./s, 1,
size(A,2)); toc

Elapsed time is 0.130031 seconds.

>> tic; X4 = bsxfun(@rdivide,A,s); toc

Elapsed time is 0.278485 seconds.
```

Экспериментируйте!

Попробуйте несколько вариантов решения одной задачи и выберите оптимальный. Здесь представлено решение типичной задачи нормировки матрицы: каждый элемент надо разделить на сумму элементов в строке. Для чистоты эксперимента следует повторить присваивания матрицам **X1, X2** *и т.д. в разном порядке. Заметим, что специально сразу выделили память под эти матрицы, чтобы на выделение не тратилось время.*

Ещё один способ сделать нормировку!

```
>> tic; X5 = sparse(diag(1./s))*A; toc

Elapsed time is 0.238213 seconds.


>> tic; X6 = diag(sparse(1./s))*A; toc

Elapsed time is 0.187723 seconds.



>> [sum(X1(:)~=X2(:)) sum(X1(:)~=X3(:))
sum(X2(:)~=X3(:))]

ans =
      1439405           0     1439405

>> [sum(abs(X1(:)-X2(:))) sum(abs(X1(:)-
X3(:))) sum(abs(X2(:)-X3(:)))]

ans =
  1.0e-012 *
           0.1073   0   0.1073
```

Первый способ можно СУЩЕСТВЕННО улучшить, если умножать на разреженную диагональную матрицу.

Ещё бОльшее улучшение получается, если диагональную матрицу сразу порождать разреженной.

Вторая матрица отличается от первых двух!

На самом деле, это специфика арифметики с плавающей точкой! Это надо учитывать.

```
>> clear
>> A = sparse(ceil(2000*rand([1
100000])), ceil(2000*rand([1 100000])),
rand([1 100000]), 2000, 2000);
>> s = sum(A,2);
>> tic; X1 = diag(1./s)*A; toc

Elapsed time is 0.894294 seconds.

>> tic; X2 = A./repmat(s,1,size(A,2));
toc

Elapsed time is 1.241347 seconds.

>> tic; X2 = A./repmat(full(s), 1,
size(A,2)); toc

Elapsed time is 0.336742 seconds.

>> tic; X3 =
A.*repmat(1./s,1,size(A,2)); toc

Elapsed time is 0.181846 seconds.

>> tic; X3 =
A.*repmat(full(1./s),1,size(A,2)); toc

Elapsed time is 0.185420 seconds.

>> tic; X4 = bsxfun(@rdivide,A,s); toc
```

Для разреженных матриц действуют другие законы!

Предыдущий пример для разреженной матрицы. «Самый долгий» первый способ теперь становится быстрее (для сильно разреженной матрицы он может стать самым быстрым).

Обратите внимание на сокращение времени при приведении матрицы к полной!

Для разных операций действую разные законы! *Попробуйте повторить все эксперименты, не нормируя данные, а центрируя*: **bsxfun(@minus,A,mean(A,2))**.

```
Elapsed time is 0.298258 seconds.

>> tic; X5 = diag(sparse(1./s))*A; toc

Elapsed time is 0.014129 seconds.


>> clear; S = sparse(fix(3*rand(1000)-
1));

>> tic; abs(S); toc

Elapsed time is 0.026724 seconds.

>> tic; spfun(@abs,S); toc

Elapsed time is 0.069756 seconds.

>> tic; S+1; toc

Elapsed time is 0.036316 seconds.

>> tic; 2*S; toc

Elapsed time is 0.009565 seconds.
```

А вот самый быстрый метод! Правильно всё-таки делать так:

```
X1 = sparse(1:size(A,1),
1:size(A,1), (1./s))*A;
```

Новая серия экспериментов с разреженной матрицей...

Встроенные «универсальные» функции часто эффективнее поэлементной реализации.

Сложение выполняется медленнее умножения! Всё дело в том, что **если меняется множество ненулевых элементов в разреженной матрице, то операция выполняется дольше!**

```
>> A = nan(2000);
>> B = ones(2000);
>> C = zeros(2000);
>> tic; A+B; toc

Elapsed time is 1.160856 seconds.

>> tic; A+C; toc

Elapsed time is 1.160238 seconds.

>> tic; B+C; toc

Elapsed time is 0.078591 seconds.

>> tic; B*C; toc

Elapsed time is 6.451293 seconds.

>> tic; A*C; toc

Elapsed time is 6.449248 seconds.
```

Некоторые данные «тормозят» работу MatLaba.

В примере показано, что операции с матрицей, заполненной NaNами происходят дольше.

Интересно, что для умножения ощутимой разницы нет!

```
>> x = round(rand(1, 1e7));
>> tic; y = 1./x; toc % половина делении на ноль

Elapsed time is 1.540780 seconds.

>> x = x + 1; % теперь делений на ноль не будет
>> tic; y = 1./x; toc

Elapsed time is 0.329759 seconds.
```

«Формирование» NaNов замедляет вычисления. Часто «выручает» добавления к вектору **eps**, *чтобы избежать делений на ноль.*

```
>> str = char(fix(26*rand([1 10000000]))+'a'); % создали случайную строку

>> tic; f1 = findstr(str,'z'); toc

Elapsed time is 0.117200 seconds.

% рекомендуется такой способ
>> tic; f2 = strfind(str,'z'); toc

Elapsed time is 0.117126 seconds.


>> tic; f3 = find(str=='z'); toc

Elapsed time is 0.312151 seconds.


>> str = fix(26*rand([1 10000000]));

>> tic; f1 = findstr(str, [0]); toc

Elapsed time is 0.085916 seconds.

>> tic; f2 = strfind(str, [0]); toc

Elapsed time is 0.084098 seconds.

>> tic; f3 = find(str==0); toc

Elapsed time is 0.173870 seconds.
```

Есть очень быстрые способы вычислений! Надо просто их знать... *Сначала рассмотрим пример поиска в очень большой строке буквы «z».*

Это самый лучший способ!

А вот «интуитивно лучший» оказался медленным! Парадокс, но **не всегда логические массивы хороши.**

Рассмотрим такую же задачу, но с другими типами данных: среди массива чисел (от 0 до 25) найти все нулевые элементы.

Для «числовой задачи» лучшим оказался способ, который использует строковые функции! *Отметим, что отсюда не следует, что все элементы надо искать как строки (необходимо помнить о специфике арифметики и представления данных в MatLabe, см. ниже):*

```
>> strfind(0.01:0.01:0.07, [0.06])

ans =
     []
```

```
>> clear
>> str = char(fix(26*rand([1
```

Теперь рассмотрим удаление буквы из строки.

```
10000000]))+'a');

>> str1 = str;
>> tic; str1 = str1(str1~='z'); toc

Elapsed time is 0.406539 seconds.

>> str2 = str;
>> tic; str2(strfind(str2, 'z')) = [];
toc

Elapsed time is 0.455166 seconds.

>> str3 = str;
>> tic; str3 = strrep(str3,'z',''); toc

Elapsed time is 0.234637 seconds.
```

Здесь уже применение логического массива лучше **strfind**.

Но самый лучший метод опять строковый!!!

```
>> Z = rand([1 10000000]);

% Замещение нужными элементами
>> A = Z;
>> tic; A = A(A>0.5); toc

Elapsed time is 0.598856 seconds.

% Удаление лишних
>> A = Z;
>> tic; A(A<=0.5) = []; toc

Elapsed time is 1.010796 seconds.

% Плохой способ замещения
>> A = Z;
tic; A = A(find(A>0.5)); toc

Elapsed time is 0.873953 seconds.

% Создание нового массива (ещё быстрее!)
>> A = Z;
>> tic; B = A(A>0.5); toc

Elapsed time is 0.556692 seconds.

% Предварительная операция с основным
массивом
>> clear B
>> A = Z;
>> A(1) = A(1)+sin(0);
>> tic; A = A(A>0.5); toc

Elapsed time is 0.567282 seconds.

% Пример того, как не надо делать
>> A = Z;
```

Даже простейшие операции допускают несколько способов реализации. Выбирайте лучший!

Здесь показано несколько способов удаления элементов массива.

Ещё раз... **Не вызывайте лишних функций!** *В данном случае* **find**. *Вообще, вместо* **a(find(a>0.5)) = 1** *пишите* **a(a>0.5) = 1** (*типичная ошибка*). *Попробуйте сравнить* **>> A = Z; tic; L = find(A>0.5); A(L) = 1; toc** **>> A = Z; tic; L = A>0.5; A(L) = 1; toc**

Попробуйте объяснить, почему происходит уменьшение времени выполнения операции (не обязательно способа замещения, остальных тоже) при добавлении строки **A(1) = A(1)+sin(0).**

```
>> tic; L = find(A<=0.5); l = length(L);
A(L(1:l))=[]; toc

Elapsed time is 1.486722 seconds.

% аналогичная задача обнуления...
>> a = rand([1000 10000]); b = a;
>> tic; a(a<0.5) = 0; toc

Elapsed time is 0.660367 seconds.

>> a = b;
>> tic; a = a.*(a>=0.5); toc

Elapsed time is 0.536720 seconds.

>> a = b;
>> tic; a(find(a<0.5)) = 0; toc

Elapsed time is 1.055111 seconds.
```

Решим следующую задачу: занулить все элементы массива, которые меньше 0.5.

Здесь выгоднее использовать логический массив для обычного умножения, а не индексации массива!

С функцией **find** *опять «ужасно»!*

```
function f = Fib1(n)
F = zeros(1, n+1);
F(2) = 1;
for i = 3:n+1
    F(i) = F(i-1) + F(i-2);
end
f = F(n);
%------------------------------
function f = Fib2(n)
if n==1
    f = 0;
    elseif n==2
        f = 1;
else
    f1 = 0; f2 = 1;
    for i = 2:n-1
        f = f1 + f2;
        f1 = f2; f2 = f;
    end

end
%------------------------------
function f = Fib3(n)
if n==1
    f = 0;
    elseif n==2
        f = 1;
else
    f = Fib3(n-1) + Fib3(n-2);
end
%------------------------------
function f = Fib4(n)
A = [0 1; 1 1];
y = A^n*[1; 0];
```

Это пять реализаций вычисления **n***-го числа Фибоначчи из* [Griffiths], [Loren]. *Рекурсивная версия* (**Fib3**) *самая медленная. Время работы функции* **Fib3** *при небольших* **n=20** *на несколько порядков превышает время работы остальных функций.*

Сравните время работы всех функций при больших значениях **n**.

Удивительно, но лучшие из этих функций работают быстрее, чем вычисления

```
alpha1 = (1 + sqrt(5)) / 2;
alpha2 = (1 - sqrt(5)) / 2;
c1 = (1 - alpha2) / (alpha1
- alpha2);
c2 = (alpha1 - 1) / (alpha1
- alpha2);
fn = round(c1 * alpha1^(n-2)
+ c2 * alpha2^(n-2)).
```

Заметим, что мы считали первое число Фибоначчи равное нулю, т.е. ряд этих чисел:

0 1 1 2 3 5 8 ...

`f = y(1);`
`%------------------------------`
`function f = Fib5(n)`
`F = [0 1 zeros(1,n-2)];`
`a = [1 -1 -1];`
`b = 1;`
`F = filter(b, a, F);`
`f = F(n);`	

`>> A = rand([10000 1000]);`
`>> b = rand([10000 1]);`
 	
`>> tic; x = pinv(A)*b; toc`
 	
`Elapsed time is 37.103009 seconds.`
 	
`>> tic; x = inv(A'*A)*A'*b; toc`
 	
`Elapsed time is 6.254909 seconds.`
 	
`>> tic; x = inv(A'*A)*(A'*b); toc`
 	
`Elapsed time is 2.608321 seconds.`
 	
`>> tic; x = (A'*A)\A'*b; toc`
 	
`Elapsed time is 6.415567 seconds.`
 	
`>> tic; x = (A'*A)\(A'*b); toc`
 	
`Elapsed time is 2.265513 seconds.`
	Пять способов решения уравнения **A*x=b** *с неквадратной матрицей* **A**. *Очень часто используется в задачах линейной регрессии!*
	
	Самый быстрый способ... Всё решают скобки!

`>> n = 70; e = ones(n, 1);`
`>> T = spdiags([e,-2*e,e], [-1,0,1], n, n);`
`>> A = full(T); b = ones(n, 1);`
`>> s = sparse(b);`
`>> tic, T\s; sparsetime=toc, tic, A\b; fulltime = toc`
 	
`sparsetime =`
`6.3137e-005`
`fulltime =`
`0.0769`
	Сравнение скоростей вычислений с разреженной и обычной (полной) матрицей из [Sigmon].
	
	Всегда используйте специфику данных!

`>> a = 10;`
`% какие-то вычисления`
`>> a = 'A';`
	Не меняйте тип переменной! *Лучше создать новую переменную другого типа.*

	Функции **load** *и* **save** *быстрее функций* **fread** *и* **fwrite**.

Упомянем ещё об одном интересном случае, когда лучше не использовать встроенные «эффективные» средства системы. В задачах анализа данные часто необходимо из матрицы **X** выделять подматрицы, соответствующие одинаковым значениям элементов первого столбца (например, в первом столбце записан номер

класса, и надо быстро перечислять объекты из заданного класса). В этом случае конструкции типа

```
>> X(X(:,1)==className,:)
```

могут оказаться очень неэффективными. Гораздо лучше упорядочить матрицу (один раз!) по значениям элементов первого столбца (часто матрица уже упорядочена) и создать векторы начал и концов классов:

```
>> X = sortrows(X, 1); % сортировка строк
>> [classes indxF] = unique(X(:,1), 'first'); % классы и их начала
>> [classes indxL] = unique(X(:,1), 'last'); % последние эл-ты классов
```

Теперь доступ к классу **className** может быть реализован командами

```
>> ic = find(classes==className, 1, 'first'); % номер нашего класса
>> X(indxF(ic):indxL(ic),:) % вывод объектов
```

§15.21. О точности и памяти…

```
>> ((1e-15 + 1e-15) + 1e30) - 1e30

ans =
       0

>> (1e-15 + 1e-15) + (1e30 - 1e30)

ans =
       2.0000e-015

>>  realmax + 1000000 - realmax

ans =
       0

>> realmax*1.000000000001

ans =
       Inf
```

Специфика арифметики с плавающей запятой!

Операции не ассоциативны.

```
>> X = [1.1:0.01:10];
>> find(S==1.13)

ans =
     Empty matrix: 1-by-0

>> X(4)

ans =
     1.1300

>> X(1:5)-[1.1 1.11 1.12 1.13 1.14]

ans =
     1.0e-015 *
         0   0   0   0.2220   0.2220
```

Пример «неверной» работы функции. Элемент **1.13** *есть в массиве, но функция* **find** *его не видит. Его видит функция* **find(abs(S-1.13)<eps)**, *но её использование также не всегда корректно, поскольку*

```
>> 0==(0+eps/1000000000000)

ans =
       0
```

Попробуйте разобраться, в чём тут проблема. См. **help eps.**

```
>> ismember(0:10,10.*(0:0.1:5))

ans =
```

Ещё одна иллюстрация этой проблемы. Сравните с действием **ismember(0:10,**

```
     1   1   1   0   1   1   0

>> 3 == ((0.1*3)*10)

ans =
     0

>> (0.3 - 0.1*3)

ans =
   -5.5511e-017
```

round(10.*(0:0.1:5))).

Заметьте, что **0.3** *не равно* **0.1*3***! *Дело в том, что число* **0.3** *не представимо точно «в бинарной форме»...*

```
>> a = single(0.1) - 0.1

a =
    1.4901e-009
```

Лишние цифры могут возникать при изменении точности.

```
x = -1:0.01:1 + eps; plot(sin(x)./x);
```

Чтобы график получался без разрывов можно сместить значения на эпсилон (совет Antenna Geek из [Loren]).

```
% Первый способ хранения в структуре
>> for i = 1:10
>> X1(i).a = 5;
>> X1(i).b = 5;
>> X1(i).c = 5;
>> end;

% Второй способ хранения в структуре
for i = 1:10
X2.a(i) = 5;
X2.b(i) = 5;
X2.c(i) = 5;
end;

% хранение в массиве ячеек
>> Y = struct2cell(X1);
>> Y = squeeze(Y) % т.к. размер 3*1*10

Y =
     [5]  [5]  [5]  [5]  [5]  [5]  [5]
[5]  [5]  [5]  [5]  [5]  [5]  [5]  [5]
[5]  [5]  [5]  [5]  [5]  [5]  [5]  [5]
[5]  [5]  [5]  [5]  [5]  [5]  [5]

% хранение в массиве
>> Z = cell2mat(Y)

Z =
   5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5  5  5  5  5  5  5  5  5  5  5  5
5  5  5

>> Z2 = int8(Z);
>> Z3 = sparse(Z);
>> whos
```

Второй способ хранения данных (структура массивов) более предпочтителен с точки зрения хранения памяти.

Самый экономный способ хранения данных – в массиве (правда, для разнотипных данных это не всегда возможно).

Часто память удаётся экономить, если учесть специфику данных. Например, хранить целочисленные матрицы в формате **int8**, *а матрицы с подавляющим числом нулей переводить в формат* **sparse** (*но если нулевых элементов мало, то возникает обратный эффект*). *Попробуйте набрать команды*

```
>> A = sparse(round(rand([1
100])));
>> A2 =
sparse(round(2/3*rand([1
100])));
>> A3 =
sparse(round(3/5*rand([1
100])));
>> whos
```

```
  Name  Size   Bytes  Class  Attributes
  X1    1x10    2232  struct
  X2    1x1      612  struct
  Y     3x10    2040  cell
  Z     3x10     240  double
  Z2    3x10      30  int8
  Z3    3x10     404  double  sparse
  i     1x1        8  double
```

```
function f = myfunc(a,b,c)
a = a + c; % new
b(1) = b(1) + 1; % new!!!
f = a(c>0) + b(c>0);
```

В этой функции переменные **a, b** *меняются, поэтому под них будут созданы новые области памяти, а под переменную* **c** *– нет.*

```
>> feature('memstats')

    Physical Memory (RAM):
  In Use:      1381 MB (56551000)
  Free:        1689 MB (69933000)
  Total:       3070 MB (bfe84000)
    Page File (Swap space):
  In Use:      1376 MB (56047000)
  Free:        4987 MB (137b0b000)
  Total:       6363 MB (18db52000)
    Virtual Memory (Address Space):
  In Use:       590 MB (24e38000)
  Free:        1457 MB (5b1a8000)
  Total:       2047 MB (7ffe0000)
    Largest Contiguous Free Blocks:
   1. [at 1f960000]  1198 MB (4aee0000)
   2. [at 7c41b000]    50 MB ( 32d5000)
   3. [at 6f49d000]    39 MB ( 2763000)
   4. [at 1c990000]    32 MB ( 2000000)
   5. [at 71fb7000]    18 MB ( 1219000)
   6. [at 775d3000]    13 MB (  d0d000)
   7. [at 1b0d0000]     8 MB (  800000)
   8. [at 6ec8b000]     7 MB (  7c5000)
   9. [at 7f7f0000]     7 MB (  79f000)
  10. [at 1a0d0000]     7 MB (  700000)
                     ======= ==========
                     1382 MB (566a2000)
```

Выводится информация о памяти. В том числе информация о свободных непрерывных блоках памяти.

Функция **pack** *переписывает содержимое памяти, используемое MatLabом, на диск, а затем загружает в один непрерывный блок.*

Помните, что на 32-битных платформах можно получить доступ только к 4Гб ОЗУ, а некоторые операционные системы (например Windows) снижают память, доступную приложениям, до 2Гб. В Vista с этим можно бороться командой

BCDEdit /set increaseuserva 3072

в XP SP2 – загрузкой ОС с параметром **/3gb** (в файле Boot.ini).

Отметим также, что хотя использование циклов является дурным тоном в системе MatLab (если без них можно обойтись), часто циклы применяют из-за ограничений по памяти. Если не удаётся совершить операцию над матрицей больших размеров, поскольку необходимо создание дополнительных матриц, то матрицу делят на «куски» и операцию производят «по кускам» (при этом возникает цикл для перебора фрагментов матрицы).

§15.22. Примеры анимации, GUI

```
>> M = moviein(50);
>> x = rand([1 10]);
```

Здесь будет храниться 50 кадров.

```
>> y = rand([1 10]);
>> axis([-1 1 -1 1])
>> axis square
>> grid off
>> h = plot(x, y, '.' );
>> set(h, 'MarkerSize' , 20);
>> for i = 1:50
>> x = x + (rand([1 10])-0.5)/100;
>> y = y + (rand([1 10])-0.5)/100;
>> set(h , 'XData' , x , 'YData' , y)
>> M(:, i) = getframe;
>> end;
>> movie(M)
```

Копируем в ***i****-й кадр.*

Напомним, что комбинация **Ctrl+C** *не прерывает выполнение встроенной функции! Поэтому выполнение* **movie(M)** *не может быть приостановлено.*

Ещё пример анимации.

```
>> [x y z] = deal(linspace(-1,1,50));
>> N = 20;
>> M = moviein(N);
>> n = 1;
>> [X,Y,Z] = meshgrid(x,y,z);
>> t = 0;
>> k = 10;
>> l = 10;
>> N2 = 1;
>> w = 0.5;
>> m = sqrt((k^2 + l^2)*(N2-w^2));
>> while n < N
>>  rho = 1 + 0.1*Z;
>>  rhopr = -0.02*sin(k*X+l*Y+m*Z-w*t);
>> Hs = slice(X,Y,Z,rho+rhopr, 0, 0, 0);
>>  set(Hs, 'facecolor', 'interp')
>>  title('Internal waves', ...
 'FontName', 'Times', 'FontSize', [18]);
>>  box on
>>  axis off
>>  drawnow
>>  t = t + 0.5;
>> A = getframe(1); % сохранение
>> M(n) = A;
>> n = n + 1;
>> end
```

Создание кнопки.

Перемещение кнопки в случайные места.

Обработчик события «нажатие кнопки».

Попробуйте вызов **get(b)** *для*

```
>> b = uicontrol('Style' , 'pushbutton',
'Units' ,'normalized', 'Position', [.5
.5 .2 .1], 'String', 'Нажми меня');
>> s = 'set(b, ''Position'', [.8*rand
.9*rand .2 .1])';
>> eval(s);
>> set(b, 'Callback', s);
```

`>> s = 'set(b, ''FontSize'' , get(b, ''FontSize'') + 1)';` `>> set(b, 'Callback' , s)`	*перечня всех свойств кнопки. Поменяйте значения некоторых свойств.*

`>> h = waitbar(0, 'Converting...');`
`>> waitbar(55/100, h, 'Converting...')`
`>> close(h)`
	Вывод «счётчика состояния процесса» (выполняйте эти команды последовательно).

`>> pause`	*Пауза в вычислениях, прерывается нажатием клавиши (полезна в m-файлах).*
`>> xpsound`	*Демонстрация работы со звуком.* *Основные функции*: **sound** *и* **wavread**.
`>> mcc –m simple`	*Получение exe-файла (для его работы нужен Matlab Compiler).*

§15.23. Перестановки

`>> randperm(5)`
 	
`ans =`
`3 2 5 1 4`
	Случайная перестановка. Выбор **k** *из* **n** *элементов без повторений*: **X = randperm(n); x = X(1:k).**

`>> S = 'abcd'; k = 2;`
`>> nchoosek(S, k)`
 	
`ans =`
`ab`
`ac`
`ad`
`bc`
`bd`
`cd`
 	
`>> nchoosek(length(S), k)`
 	
`ans =`
`6`
	Всевозможные сочетания по k элементов.
	
	Число таких сочетаний. Если **n** *и* **k** *векторы, то можно использовать запись* **round(exp(gammaln(n+1) - gammaln(k+1) - gammaln(n-k+1))).**

`>> perms('ABC')`
 	
`ans =`
`CBA`
`CAB`
`BCA`
`BAC`
`ABC`
`ACB`
 	
`>> factorial(3)`
 	
`ans =`
`6`
	Всевозможные перестановки.
	
	Их число.

§15.24. Символьные вычисления

```
>> syms a b;
>> simplify((a+1)*(a-1)+b*b+2*b+2)

ans =
     a^2+1+b^2+2*b

>> simplify(sin(2*a)/cos(a))

ans =
     2*sin(a)

>> expand((a+1)*(a-1)*(b+2))

ans =
     a^2*b+2*a^2-b-2

>> factor(a^3-b^3)

ans =
     (a-b)*(a^2+a*b+b^2)

>> subs(sin(a)+sin(b), b, pi)

ans =
     sin(a)

>> diff(sin(a^3))

ans =
     3*cos(a^3)*a^2

>> int(sin(a)^2)

ans =
     -1/2*sin(a)*cos(a)+1/2*a


>> q = solve('a+b=1', 'a^2-b=2');
>> q.a

ans =
     -1/2-1/2*13^(1/2)
     -1/2+1/2*13^(1/2)

>> q.b

ans =
     3/2+1/2*13^(1/2)
     3/2-1/2*13^(1/2)

>> dsolve('Da+a=exp(t)')
```

Пример символьных вычислений.

Упрощение выражений.

Раскрытие выражений (ср. с упрощением).

Разложение на множители.

Подстановка.

Производная.

Интеграл.

Системы уравнений.

Дифференциальные уравнения.

```
ans =
     1/2*exp(t)+exp(-t)*C1
```

```
>> syms x1 x2 x3
>> W = [1 x1 x1^2; 1 x2 x2^2; 1 x3 x3^2]

W =
   [  1, x1, x1^2]
   [  1, x2, x2^2]
   [  1, x3, x3^2]

>> pretty(W)
   [           2]
   [1   x1   x1 ]
   [            ]
   [           2]
   [1   x2   x2 ]
   [            ]
   [           2]
   [1   x3   x3 ]

>> a = det(W)

a =
    x2*x3^2-x2^2*x3-
x1*x3^2+x1^2*x3+x1*x2^2-x1^2*x2

>> factor(a)

ans =
     -(-x2+x1)*(x3-x2)*(x3-x1)
```

Пример символьного вычисления определителя Вандермонда.

«Удобный» вид арифметических выражений.

```
>> syms x;
>> limit([sin(x)/x, (1+x)^(1./x)])

ans =
     [   1, exp(1)]

>> Taylor(sin(x), x, 0, 8)

ans =
     x-1/6*x^3+1/120*x^5-1/5040*x^7

>> syms x y z;
>> laplace(x+y*z)

ans =
     1/s^2+y*z/s
```

Вычисление пределов.

Разложение в ряд Тейлора в окрестности нуля (выписать 8 *первых членов разложения).*

Преобразование Лапласа.

```
>> x = linspace(0,pi,100);


>> trapz(x,sin(x))

ans =
      1.9998
```

«Несимвольно» интегралы берутся так...

Интегрирование методом трапеций.

`>> quad('sin(x)',0,pi)`
 	
`ans =`
`2.0000`
	Интегрирование по формуле Симпсона (с автоматическим выбором шага).

§15.25. Задания для самостоятельной работы

Для начала разберём подробно решение следующей задачи [CVD] (попробуйте её решить, не заглядывая в ответ): *заполнить в векторе* **x** *все нулевые значения предыдущими ненулевыми значениями. Для вектора* **x = [7 0 0 1 0 0 0 3 0]** *должен получиться ответ* **x = [7 7 7 1 1 1 1 3 3]**. Решение состоит из трёх команд:

`>> x = [7 0 0 1 0 0 0 3 0];`
`>> valind = find(x)`
 	
`valind =`
`1 4 8`
	Находим ненулевые позиции.

`>> x(valind(2:end)) = diff(x(valind))`
 	
`x =`
`7 0 0 -6 0 0 0 2 0`
	Подготавливаем вектор для действия кумулятивной суммы... заменяем элемент разностью между ним и предыдущим ненулевым.

`>> x = cumsum(x)`
 	
`x =`
`7 7 7 1 1 1 1 3 3`
	Теперь вычисляем кумулятивную сумму и получаем ответ.

Здесь «ключевой» является вторая команда. Чтобы решить задачу, необходимо догадаться, что следует применять функцию **cumsum**. Это не так сложно сделать, поскольку, когда мы её применяем к вектору, соседние нулевые позиции она заполняет одинаковыми значениями. Осталось добиться, чтобы эти значения были «нужными».

При решении приведённых ниже задач нельзя пользоваться циклами и условными операторами. Прочитайте условие в правой колонке, попробуйте решить самостоятельно. Попробуйте не просто решить, а дать компактное (минимальное число строк кода) и эффективное (быстро выполняется даже при больших размерах входных данных) решение. Ответ приводится в левой колонке.

Ответ	Задача

`>> X = setdiff(2:2:N, 6:6:N)`
 	
`% гораздо хуже:`
`>> setdiff(2*(1:fix(N/2)),`
`3*(1:fix(N/3)))`
	Выписать все чётные числа от **1** *до* **N**, *которые не делятся на* **3**.

`>> ismember(A, B1) | ismember(A, B2)`	*Пометить в матрице* **A** *все элементы, перечисленные в векторах* **B1** *и* **B2**. *Для* **B1 = [1 2], B2 = [2 5], A =** **2 3 4** **1 2 2**

	3 3 5 *ответ* 1 0 0 1 1 1 0 0 1
% в одну строчку: >> reshape(repmat(X,N,1), 1, N*length(X)); % через вспомогательную переменную и индексный вызов: >> A = X(ones(1,N),:); >> A = A(:).'	*В вектор строке* **X** *повторить все значения* **N** *раз. При* **N = 3, X = [2 0 1]** *ответ* **ans =** **2 2 2 0 0 0 1 1 1**
>> dec2bin(0:(2^n-1))-'0' % чуть сложнее... >> +(dec2bin(0:(2^n-1))=='1') ans = 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	*Построить матрицу (типа* ***double****), в которой по строкам записаны все* **n**-*мерные бинарные векторы.*
>> X'*Y % типичная ошибка – использование repmat!	*Построить таблицу умножения всевозможных пар элементов таких, что первый берётся из множества* **X**, *а второй – из* **Y**. *Например, при* **X = [-1 0 1], Y = [2 3 5]** *ответ* **ans =** -2 -3 -5 0 0 0 2 3 5
>> cumsum(ones(n)) % менее эффективный вариант: >> tril(ones(n))*ones(n)	*Для натурального числа* **n** *построить матрицу размера* **n×n**, *все элементы* **i**-*й строки которой равны* **i**. *Например, при* **n = 4** **ans =** 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 *Команда порождения должна «умещаться» в одной строке. Можно использовать любые стандартные функции, кроме* **for**, **repmat** *и* **meshgrid**.
>> b = A==max(A);	*Занумеровать в порядке*

>> cumsum(b).*b	*следования все максимальные элементы в векторе* **A**. *Например, для вектора* **A = [1 2 3 3 2 1 3 1]** *получить* **[0 0 1 2 0 0 3 0]**.
>> X = triu(repmat(1:n, n, 1)); **>> X(X(:)'~=0)**	*Для натурального числа* **n** *породить вектор, в котором сначала записана единица, потом две двойки, потом три тройки и т.д. Последние элементы вектора –* **n n**-ок. Например, при **n = 4** **X =** **1 2 2 3 3 3 4 4 4 4**
>> [X Y] = meshgrid(A, B); **>> max(max(sin(X + Y)))**	*Дано множество* **A** *и множество* **B**. *Найти максимум функции* **sin(a+b)** *при* **a** *из* **A** *и* **b** *из* **B**.
>> max(max(A)-min(B), max(B)-min(A)) **% Функция abs не используется!**	*Дано множество* **A** *и множество* **B**. *Найти (самым эффективным способом) максимум функции* **abs(a-b)** *при* **a** *из* **A** *и* **b** *из* **B**.
>> max(A)<=min(B)	*Самым простым способом выяснить, верно ли, что все элементы множества* **B** *не меньше всех элементов множества* **A**.
>> [X Y] = meshgrid(1:n, 1:m); **>> abs(X - Y)**	*Придумать универсальный способ реализации матриц, в которых* **ij**-*й элемент равен расстоянию до главной диагонали в метрике L1 (без использования функции* **toeplitz**). *Пример матрицы:* **ans =** **0 1 2 3 4** **1 0 1 2 3** **2 1 0 1 2**
>> [x y] = meshgrid(A, B); **>> X = [x(:), y(:)]**	*Для двух множеств* **A**, **B** *сформировать матрицу, в которой по строкам записаны все пары – элементы декартова произведения этих множеств. Например, для множеств* **A = [2 3], B = [1 2]** *ответ* **X =** **2 1** **2 2** **3 1** **3 2**

`>> max(A(isprime(A)))`	*Найти максимальный простой элемент множества чисел* **A**.

`>> [a a a]  = unique(a)`
 	
`% обходной путь...`
`>> [b i] = sort(a);`
`>> b = cumsum([1 diff(b)]>0);`
`>> a(i) = b`
	Заменить элементы вектор-строки **a** *на индексы соответствующих элементов вектора* **unique(a)**. *Для строки* **a = [9 2 5 6 2 5 5]** *ответ –*
	a =
	4 1 2 3 1 2 2

`>> [B I] = unique(A, 'first');`
`>> [B J] = unique(A, 'last');`
`>> B(I==J)`
 	
`% другой вариант`
`>> b=sort(A);`
`>> setdiff(b, b(([1 diff(b)]==0)))`
	Во множестве **A** *оставить только уникальные элементы (входят только один раз). Ответ для множества* **A = [3 2 1 3 4 4 7 5 5]**:
	ans =
	1 2 7

`>> sum(A=='a')`	*Для строки* **A** (*например для* **A = 'abbcaaccab'**) *посчитать число вхождений буквы* **'a'** (*самым простым и быстрым способом*).

`>> n = 2;`
`>> b = (2.^(0:n))';`
`>> A = repmat((0:n)', 1, n+1);`
`>> A = A.^(A');`
`>> A\b`
`ans =`
`1.0000`
`0.5000`
`0.5000`
	Дано натуральное число **n**. *Найти коэффициенты полинома степени* **n**, *который в точках* **0,1,...,n** *принимает значения* **1,2,4,8,...,2^n** (*все степени двойки*). *Написать ответ при* **n = 2**.

`>> all(A == A(end:-1:1))`	*Проверить, является ли вектор* **A** *симметричным. Например, векторы* **A = [3 4 5 4 3], A = [6 6], A= [7]** *являются, а векторы* **A = [1 2], A = [1 2 3 4 1]** *– нет.*
`>> sort([A, B])` `% без sort не будет отсортированным`	*Как объединить два множества с учётом кратности элементов, т.е. объединение* **A = [2,1,3,4,3]** *и* **B = [2,3,5]** *даст* **[1, 2, 2, 3, 3, 3, 4, 5]**?
`>> sum(B(:) == 2) == 2`	*Как проще всего проверить условие «в матрице* **B** *только два элемента равны двум»?*
`>> X = repmat(1:n, n, 1);` `>> min(X, X')`	*Придумайте способ порождения матриц вида* **ans =** **1 1 1** **1 2 2** **1 2 3**

	ans = **1 1 1 1** **1 2 2 2** **1 2 3 3** **1 2 3 4**
S(1, :) = s1; **S(2, 1:length(s2)) = s2;** **S = S(:)';** **S(S==0) = []**	*Написать код, который соединяет две строки* **s1, s2** *следующим образом:* **[s1(1) s2(1) s1(2) s2(2) ...].** *Для* **s1 = '12345', s2 = 'abc'** *должна получаться строка* **'1a2b3c45'**, *а для* **s1 = '123', s2 = 'abcde'** – *строка* **'1a2b3cde'**.
>> [sortS i] = sort(double(S)); **>> S(i(([1, diff(sortS)] == 0))) = []** **% без приведения double не получится сделать diff**	*Оставить в строке* **S** *только первые вхождения всех элементов. Для* **S = 'try to find unique elements'** *ответ* **S =** **try ofinduqelms** (*ответ не отсортирован в общем случае*).
% Решение из [Loren] **>> i = any([a;b] == 0)** **>> a(i) = []** **>> b(i) = []** **% Другой вариант решения, который можно** **% улучшить, имея априорные сведения** **% о количестве нулей в векторах** **>> i = find(a~=0);** **>> i = i(b(i)~=0);** **>> a = a(i);** **>> b = b(i);**	*Оставить в двух векторах* **a, b**, *содержащих одинаковое число элементов, только те элементы, которые соответствуют позициям ненулевых элементов в обоих векторах. Для* **a = [NaN 1 2 0 0 Inf 0];** **b = [1 0 3 4 0 0 NaN];** *ответ* **a =** **NaN 2** **b =** **1 3**
>> uS = unique(S)'; **>> [m i] = max(sum((repmat(uS, 1, size(S,2)) == repmat(S, length(uS), 1))'));** **>> [uS(i) m]** **% подумайте, нельзя ли улучшить этот метод**	*Для мультимножества* **S** *определить, какой элемент встречается в нём наибольшее число раз. Вывести этот элемент и его кратность. Для* **S = [1 1 3 2 3 3 1 3 9]** *ответ –* **ans =** **3 4** (*элемент 3 встретился 4 раза*). *Для* **S = [2 1 2 1]** *ответ –* **[1 2]** *или* **[2 2].**
>> isequal(sort(A), sort(B)) **% обходной путь** **>> (length(A) == length(B)) &&**	*Сравнить, совпадают ли два мультимножества? Например, мультимножества* **A = [1 2 3 1 1 2]** *и* **B = [3 1 1 1 2 2]**

`all(sort(A) == sort(B))`	*равны, а* **A = [1 2 3 1 2]** *и* **B = [1 1 3 2 3]** *– нет.*
`>> max(A(find(A(1:end-1)==0)+1))`	*Найти максимальный элемент в вектор-строке среди элементов, перед которым стоит нулевой. Для* **A = [6,2,0,3,0,0,5,7,0]** *ответ* **ans = 5**

`% этот способ не рекомендуют из-за лишних умножений...`
`>> X = val.*ones(siz);`
`% аналог со сложениями`
`>> X = val + zeros(siz);`
`% часто рекомендуемый способ`
`>> X(prod(siz)) = val;`
`>> X = reshape(X, siz);`
`>> X(:) = X(end);`
`% улучшение предыдущего варианта`
`>> X(1:prod(siz)) = val;`
`>> X = reshape(X,siz)`
`% один из самых быстрых способов`
`>> X = zeros(siz); X(:) = val;`
`% самый «естественный» способ`
`>> X = repmat(val,siz);`
`% самый плохой (долгий) способ`
`>> X = val(ones(siz));`
	Реализуйте различные способы генерации массива размера **siz** *с элементами, равными* **val**. *Чем больше, тем лучше. При* **siz = [2 3]** *и* **val = 5** *ответ*
	X =
	5 5 5
	5 5 5

`>> X(mod((1:end)-k-1, end)+1)`	*Реализуйте в одну строчку сдвиг вектор-строки* **X** *на* **k**. *Например, при* **X = 'MatLab'** *и* **k = 2** *результат -* **'abMatL'**, *а при* **k = -1** *-* **'atLabM'**. *Нельзя использовать функции MatLab'а, содержащие в названии слово* **shift**.
`>> X = ceil(s*rand([m n]));`	*Сгенерируйте случайную матрицу размера* **m×n** *с элементами из* **1:s**.
`>> A(1: (size(A,1)+1): (size(A,1)*min(size(A))))`	*Реализуйте функцию* **diag(A)** *через другие функции MatLab'а.*
`>> Y = reshape(X, [size(X,1) size(X,2)* size(X,3)])`	*Заданы двумерные матрицы* **A, B, C, D** *одинаковых размеров. По матрице* **X = cat(3, A, B, C, D)** *получите матрицу* **[A, B, C, D]**.
`>> Y = permute(X,[1 3 2]);` `>> Y = reshape(Y, [size(X,1)* size(X,3) size(X,2)])`	*Заданы двумерные матрицы* **A, B, C, D** *одинаковых размеров. По матрице* **X = cat(3, A, B, C, D)** *получите матрицу* **[A; B; C; D]**
`>> B = repmat(A, [m n])`	Реализуйте присваивание **B = kron(ones(m,n), A)** более

	простым способом для любых натуральных **m, n** и матрицы **A**. Например, при **m = 2; n = 3; A = [4 5];** **B =** **4 5 4 5 4 5** **4 5 4 5 4 5**
>> B = A(repmat(1:size(A,1),m,1), repmat(1:size(A,2),n,1)) **% Менее эффективный способ из [Acklam]** **>> i = 1:size(A,1);** **>> j = 1:size(A,2);** **>> B = A(i(ones(1,m),:), j(ones(1,n),:))**	*Реализуйте присваивание* **B = kron(A, ones(m,n))** *более простым способом для любых натуральных* **m, n** *и матрицы* **A**. *Например, при* **m = 2; n = 3; A = [4 5; 6 7];** **B =** **4 4 4 5 5 5** **4 4 4 5 5 5** **6 6 6 7 7 7** **6 6 6 7 7 7**
>> B = reshape([repmat([A; repmat(zeros(size(A)), n, 1)], n-1, 1); A], n*size(A,1), n*size(A,2)); **>> I = reshape((1:n*size(A,2))', n, size(A,2))';** **>> B = B(:,I);** **% Более эффективный и эффективный способ из [Acklam]** **>> B = zeros([size(A) n n]);** **>> B(:,:,1:n+1:n^2) = repmat(A, [1 1 n]);** **>> B = permute(B, [1 3 2 4]);** **>> B = reshape(B, n*size(A));** **% Самый эффективный и эффектный способ** **% (формирование «нужной» команды)** **>> eval(['B = blkdiag(' repmat('A,',1,n-1) 'A);'])**	*Реализуйте присваивание* **B = kron(eye(n),A)** *более простым способом для любых натуральных* **m, n** *и матрицы* **A**. *Например, при* **n = 3; A = [4 5; 6 7];** **B =** **4 5 0 0 0 0** **6 7 0 0 0 0** **0 0 4 5 0 0** **0 0 6 7 0 0** **0 0 0 0 4 5** **0 0 0 0 6 7**
% Способ из [Acklam] **>> B = zeros([size(A) n n]);** **>> B(:,:,1:n+1:n^2) = repmat(A, [1 1 n]);** **>> B = permute(B, [3 1 4 2]);** **>> B = reshape(B, n*size(A));**	*Реализуйте присваивание* **B = kron(A,eye(n))** *более простым способом для любых натуральных* **m, n** *и матрицы* **A**. *Например, при* **n = 3; A = [4 5; 6 7];** **B =** **4 0 0 5 0 0** **0 4 0 0 5 0** **0 0 4 0 0 5** **6 0 0 7 0 0** **0 6 0 0 7 0** **0 0 6 0 0 7**
>> reshape(permute(repmat(A, [1 1 size(B, 1) size(B, 2)]), [3 1 4 2]),size(A, 1)*size(B, 1), size(A, 2)* size(B, 2)).* repmat(B, [size(A, 1)	*Реализуйте функцию* **kron(A, B)** *для любых двумерных матриц через функции* **reshape, permute, repmat,**

size(A, 2)])	**size**. *Для* **A = [2 3; 4 5], B = [1 7; 6 0]** *должна получаться матрица* **ans =** **2 14 3 21** **12 0 18 0** **4 28 5 35** **24 0 30 0**
>> D = C(:,size(C,2):-1:1)'	*Реализуйте присваивание* **D = rot90(C)**, *не используя функции* **rot90**.
>> D = C(size(C,1):-1:1,size(C,2):-1:1)	*Реализуйте поворот матрицы на* 180 *градусов без использования функции* **rot90**. *При* **C = [1 2 3; 4 5 6]** *результат поворота* **D =** **6 5 4** **3 2 1**
>> all(x(:) > 0) **% или так (хуже)** **>> sum(x(:)<=0)==0**	*Напишите код, который проверяет, что в матрице* **x** *все элементы положительные.*
>> prod(size(x))<=1 **% обратите внимание, что size([])= [0 0]**	*Напишите код, который проверяет, что матрица* **x** *является скаляром или пустой матрицей.*
>> ndims(x) <= 2 & sum(size(x) > 1) <= 1	*Напишите код, который проверяет, что* **x** *является вектор-столбцом или вектор-строкой* (*быть может, пустой*).
>> m = 10; n = 2; **>> X = rand([m n]);** **>> y = rand([1 n]);** **>> [m i] = min(sum((X - repmat(y, m, 1)).^2, 2));** **>> scatter(X(:,1),X(:,2));** **>> hold on;** **>> scatter(y(:,1),y(:,2),'filled');** **>> line ([X(i,1) y(1)],[X(i,2) y(2)])**	*Сгенерируйте на плоскости* **m** *случайных точек. Для новой случайной точки найдите среди этих точек ближайшую* (*реализуйте метод ближайшего соседа*) *по евклидовой метрике. Сделайте визуализацию.*
>> D = reshape(sqrt(sum((repmat(X, size(Y,1), 1) - reshape(repmat(Y', size(X,1), 1), size(X, 1)*size(Y, 1), size(X, 2))).^2, 2)), size(X, 1),size(Y, 1)) **% Эффектный способ из [Acklam]** **>> D = sqrt(sum(abs(repmat(permute(X, [1 3 2]), [1 size(Y,1) 1]) - repmat(permute(Y, [3 1 2]), [size(X,1) 1 1])).^2, 3))**	*Для матрицы* **X = rand([m n])** *и матрицы* **Y = rand([h n])**, *в которых по строкам перечислены координаты* **n**-*мерных точек постройте* **m**×**h**-*матрицу попарных евклидовых расстояний*.

`>> [x(:,2) x(:,1)] = find(tril(ones(n), -1))`	*Сгенерировать все сочетания из* **n** *элементов по два, не используя специальных функций для работы с перестановками (т.е. реализовать функцию* **nchoosek(1:n, 2)**).

`% Первый способ`
`>> m = length(a);`
`>> len = b - a + 1;`
`>> n = sum(len);`
`>> c = ones(1, n);`
`>> c(1) = a(1);`
`>> len(1) = len(1) + 1;`
`>> c(cumsum(len(1:end-1))) = a(2:m) - b(1:m-1);`
`>> c = cumsum(c);`
 	
`% Эффектное решение методом формирования «нужной» команды`
`>> c = eval(['[', sprintf('%d:%d ',[a;b]), ']'])`
 	
`% Ещё одно эффектное решение из [Loren], предложенное Matt Fig и Jason Merrill.`
`>> f = @(a,b) cell2mat(arrayfun(@(x,y) {x:y}, a, b));`
`>> c = f(a,b);`
	Для двух векторов одинаковой длины **a** *и* **b** *сгенерировать вектор* **c = [a(1):b(1) a(2):b(2) ...]**. *Для* **a = [1 3 5 6], b = [3 3 5 7]** *ответ*
	
	`c =`
	`1 2 3 3 5 6 7`

`>> A = repmat((1:max(a))', 1, length(a));`
`>> A(A>repmat(a, max(a),1)) = 0`
	Для вектор-строки **a** *с натуральными элементами сгенерировать матрицу, у которой в первом столбце записаны числа от* 1 *до* **a(1)**, *во втором – от* 1 *до* **a(2)** *и т.д. При этом матрица дополнена нулевыми элементами. Так, для* **a = [3 1 2 3]** *должна порождаться матрица*
	`A =`
	`1 1 1 1`
	`2 0 2 2`
	`3 0 0 3`

`function m = mymean(X)`
`L = isnan(X);`
`X(L) = 0;`
`m = sum(X)./sum(~L);`
	Напишите функцию **mymean**, *которая находит средние значения (по одному направлению) с учётом* **NaN** *элементов матрицы. Для*
	`X =`
	`NaN 1 2`
	`NaN 0 6`
	`1 5 NaN`
	
	`>> mymean(X)`

	ans = **1 2 4**
>> H = min(max(H, minH), maxH)	*В матрице* **H** *заменить все значения, которые больше* **maxH** *на* **maxH**, *а все значения, которые меньше* **minH** *на* **minH**. *Например, при* **H = [1 5 3; 5 2 8; 7 9 10], maxH = 8, minH = 4** *должна получиться матрица* **H =** **4 5 4** **5 4 8** **7 8 8**
>> B = cumsum(A); **>> diff([0 B(cumsum(a))])**	*Для вектор-сток* **A** *и* **a** *вычислить сумму первых* **a(1)** *элементов вектора* **A**, *сумму следующих* **a(2)** *элементов вектора* **A** *и т.д. Например, при* **A = 1:10, a = [2 1 3]** *ответ* **ans =** **3 3 15**
>> i = abs(x1-y)>abs(x2-y); **>> a = x1;** **>> a(i) = x2(i);**	*Даны вектор-строки* **x1**, **x2** *и* **y**. *Сформировать вектор-строку* **a**, *в которой* **a(i) = x1(i)**, *если* **y(i)** *ближе к* **x1(i)**, *а иначе* **a(i) = x2(i)**. *Для* **x1 = [1 2 3 4 5], x2 = [3 1 2 1 2], y = [0 1 2 3 4]** *ответ* **a =** **1 1 2 4 5**
>> A(:,~v) = 0 **% чуть хуже вариант из [Cambridge]** **>> A = A*diag(v)** **% и совсем плохо:** **>> A = A.*repmat(v,size(A,1),1)** **% возможное решение** **>> A = bsxfun(@(x,y) x.*y, A, v);**	*Дана матрица* **A** *и бинарная вектор-строка* **v**. *В матрице оставить без изменения все столбцы, которые соответствуют единицам вектора* **v**, *а элементы остальных столбцов занулить. Для* **v = [0 1 0 1]** *и* **A = [1:4;3:6]** *ответ* **A =** **0 2 0 4** **0 4 0 6**
>> i = (a == 0); **>> a(i(1:end-1)&i(2:end)) = []**	*Все нули, идущие подряд в векторе* **a**, *заменить одним нулём. Для* **a = [0 0 1 2 0 0 0 3 0 4 0 0]** *ответ* **a =** **0 1 2 0 3 0 4 0**
>> Z = sum(X*W.*conj(Y), 2);	*Реализовать эффективно код* [Acklam]

`% менее эффективное решение`
`>> Z = diag(X*W*Y');`
	`Z = zeros(m, 1);`
	`for i = 1:m`
	`Z(i) = X(i,:)*W*Y(i,:)';`
	`end`

`% вариант из [Cambridge]`
`>> K = X*X';`
`>> d = diag(K);`
`>> one = ones(length(d), 1);`
`>> D = sqrt(d*one'+one*d'-2*K);`
	Построить матрицу попарных евклидовых расстояний системы точек, которые покоординатно записаны в матрице **X**.

`% решение [Acklam]`
`>> [i, j, v] = find(X);`
`>> t = logical(diff([0; j]));`
`>> i = i(t)';`
`>> v = v(t)';`
	В каждом столбце матрицы **X** *есть ненулевой элемент. Найти порядковые номера (в столбце) и значения всех первых ненулевых элементов каждого столбца. Для* **X =**
	`0 2 0`
	`0 0 3`
	`4 4 5`
	ответ
	`3 1 2`
	`4 2 3`
	(т.е. 3й элемент первого столбца равен 4, 1й элемент второго – 2 и т.д.)

`>> B = bsxfun(@(x,y) x.*(x==y), A, max(A,[],2))`	*В каждой строке матрицы* **A** *оставить неизменным максимальный элемент, а остальные обнулить. Ответ записать в новую матрицу* **B**. *Для* **A =**
	`8 9 4 0`
	`7 8 3 2`
	`8 3 1 6`
	ответ **B =**
	`0 9 0 0`
	`0 8 0 0`
	`8 0 0 0`

§15.26. Как не надо программировать в системе MatLab

Весь предыдущий текст направлен на обучение программированию на примерах. Описание MatLab-команд есть в хорошо организованных файлах помощи (большинство современных книг по этой системе просто «повторяют» их содержание). Язык системы очень прост в освоении. Достаточно серьёзные программы удаётся начинать писать прямо сразу после знакомства с системой MatLab[1]. Однако, несмотря на простоту освоения языка, есть куча тонкостей, которыми надо овладеть… Им и было уделено основное внимание. Практика показывает, что большинство студентов не может сразу проникнуться всеми тонкостями. Ниже приведены примеры типичных ошибок студентов, только что «освоивших» язык.

[1] Как правило, пользователи уже знают язык C.

Написано	Верный вариант
`>> A = reshape(A, [1 size(A,1)*size(A,2)])` `% для двумерной матрицы A`	`>> A = A(:)'`

`>> b = size(x);`
`>> b(end)`
`% для вектор-строки x`
	`>> b = length(x);`

`>> A - tril(A)`	`>> triu(A,1)`

`>> sum(a==b) == size(a, 2)`
`% для вектор-строк a и b`
 	
`>> sum((a - b) ~= 0) == 0`
	`>> isequal(a,b)`
	`% универсальный способ!`
	
	`>> all(a == b)`

`>> X = X(:)'; X(X==0) = []`	`>> X(X(:)'~=0)`
`>> logical(A + B)` `% для логических массивов A и B`	`>> A | B`
`>> nz = 2:length(A);` `>> nz(A(1:end-1) == 0)`	`>> find(A(1:end-1) == 0) + 1` `% часто так лучше!`

`>> find(ismember(A,0))`	`>> find(A==0)`
	`% есть ещё решение через`
	`% строковые функции`

`>> min(x(:)) > 0`
 	
`>> all(all(x>0))`
 	
`>> (length(x(:))-length(x(x>0)))`
`== 0`
`% сложно поверить, что это написал человек хоть что-то понимающий в программировании`
	`>> all(x(:) > 0)`

`>> max(a(find(a==0) + 1))` `% для вектор-строки a`	*Возможен выход за пределы массива (при* `a = [0]`*). Верно:* `>> max(a(find(a(1:end-1)==0)+1))`

`>> [x y] = meshgrid(1:n, 1:n);`
`>> x = x - y;`
`% y потом не используется`
	`>> x = repmat(1:n,n,1);`
	`>> x = x - x';`
	или
	`>> x = -ones(n);`
	`>> x(1,:) = 0:n-1;`
	`>> x = cumsum(x);`

`>> cumsum(A==b).*(A==b)`	`>> z = A==b; cumsum(z).*(z)`
`>> setdiff(A, intersect(A,B))`	`>> setdiff(A, B)`
`>> [B I] = unique(A)` `>> A(I(I>=5))`	`>> [B I] = unique(A)` `>> B(I>=5)`

§15.27. Советы по оформлению m-файлов

1. Имена переменных и функций должны быть «говорящими» (примеры имён переменных: **`fileName`**, **`maxValue`**, примеры имён функций: **`classify`**, **`selectFeatures`** и т.д.).

2. При именовании переменных и функций придерживайтесь определённого стиля (который принят в Вашей группе разработчиков). Опишем некоторые наиболее часто встречающиеся рекомендации… Переменные, которые в цикле меняют свои значения, лучше начинать с букв **`i`**, **`j`**, **`k`**; переменные, обозначающие число, – с

букв **n**, **m**. Если функция возвращает только указатель или ничего не возвращает, то её название отражает, что она делает (**plot**). Если функция возвращает одно значение, то её название отражает смысл этого значения (**max**, **min**, **mean**). В названиях не используют отрицаний (**isFound**, но не **isNotFound**). Часто используют префиксы:

префикс (примеры)	используют
get, set (getobj(), setappdata())	при доступе к объекту или свойству;
compute (computeweights(), computespread())	для отражения, что вычисление функции может занять достаточно много времени;
find (findnearestneighbor(), findheaviestelement())	когда что-то «ищется»;
is, has, can, should (iscomplete(), hasLicense(), canEvaluate())	когда возвращается логическая переменная.

Фрагмент кода

```
for iFile = 1:nFiles
    for jPosition = 1:nPositions

    fileName = getname(); % Имя функции - маленькими буквами
    fileSize = filesize(); % Не надо fsz
    if (checkTiffFormat()) % Не checkTIFFFormat!
       tableNo = % определение номера таблицы

       % ...

       for i = 1:MAX_ITERATIONS % Константа заглавными буквами
           Segment.length = % Название структуры заглавными
                            % Не надо Segment.segmentlength
       end;
    end; % if (checkTiffFormat())

    end; % for jPosition = 1:nPositions
end; % iFile = 1:nFiles
```

3. Программа должна быть разбита на «небольшие» части: функции. Если какой-то кусок кода встречается несколько раз, то неплохо бы сделать его отдельной функцией. Если функция используется только в теле другой функции, то надо сделать её встроенной (описать в том же файле).

4. При передаче данных функции лучше использовать аргументы, а не глобальные переменные. Использование структур помогает избавиться от длинных списков аргументов.

5. Используйте уже существующие функции, если их можно эффективно применить для решения Вашей задачи. Много известных алгоритмов уже реализовано (если не

в самой системе MatLab, то в одной из её библиотек). Для поиска нужных функций научитесь пользоваться помощью системы.

6. Не ленитесь документировать функцию! Не оставляйте это «на потом». Назначения всех переменных должны быть описаны. Заголовок функции должен поддерживать использование **help** (выводит первый непрерывный блок комментариев) и **lookfor** (сканирует первую строку файла). Кстати, в поле **Description** окна **Current Directory** выводится первый комментарий, найденный в тексте файла (это очень удобно при просмотре содержимого каталога).

Схема для заголовка функции

```
%FINDCLASS Классификация выборки методом MVS.
% function [classes, StructInfo] =
% findClass(trainData, trainMarks, testData)
%
% DESCRIPTION:
% Описание функции
%
% INPUTS:
% Описание входных данных. Вот пример...
% trainData - ОБУЧЕНИЕ - двумерная матрица "объект-признак".
% trainMarks – МЕТКИ ОБУЧЕНИЯ – вектор-столбец.
% testData - КОНТРОЛЬ - двумерная матрица "объект-признак".
% Обе матрицы имеют одинаковое число столбцов.
% Возможны пропуски значений (NaN).
% Вектор trainMarks имеет длину равную числу строк trainData.
%
% OUTPUTS:
% Описание выходных данных.
%
% USAGE:
% Примеры использования функции.
%
% NOTES:
% Особенности функции.
%
% Needs:
% Перечень файлов, необходимых для работы функции.
%
% See also:
% Перечень «похожих» функций.
%
% GLOBALS:
% Перечень и описание глобальных переменных.
%
% HISTORY:
% История изменений функции.
%
% AUTHOR:
% Данные автора (+ почта для связи).
%
% REFERENCES:
```

```
% Необходимые ссылки
function [classes, StructInfo] = findClass(trainData, trainMarks,
testData)
```

7. Используйте выравнивание, отступы и пробелы. Это повышает «читабельность». Отделяйте пробелом комментарий от знака **%**, отделяйте левую и правую части присваивания от знака **=** и т.д. Используйте скобки (особенно в логических выражениях). Это понятно и позволяет избежать ошибок. Избегайте слишком длинных строк.

Пример грамотного выравнивания

```
weightedPopulation = (doctorWeight * nDoctors) + ...
                     (lawyerWeight * nLawyers) + ...
                      (chiefWeight * nChiefs);
```

8. В случае ошибки желательно, чтобы функция её диагностировала и выдавала пользователю рекомендации по исправлению. Не забывайте про **try**, **catch**, **end**. Проверяйте число аргументов функции. Всегда «ожидайте ошибки» (например, условный оператор **switch** должен включать условие **otherwise**).

9. Помните, что некоторые функции ухудшают «читабельность» (**eval**, **feval**, **exist**, **break**, **continue**). Переменная, которая «создаётся» в цикле, должна быть инициализирована непосредственно перед этим циклом:

```
result = zeros(nEntries,1);
for index = 1:nEntries
    result(index) = computeFactors(index);
end
```

10. Не дублируйте данные.

```
% плохо
trainObjects = [.1 .23 .2 .15 .7];
etalons = [.1 .2 .7];

% хорошо
trainObjects = [0.1 0.23 0.2 0.15 0.7]; % и 0 надо указывать!
indexOfEtalons = [1 3 5];
```

11. Пишите «минимальный» код (по числу команд) и простой код (не используйте слишком сложные синтаксические конструкции, которые не свойственны языку).

12. Не решайте сразу слишком общую задачу – решайте свою (в интерпретаторе код, который решает общую задачу, очень сложно отладить). Сразу проверяйте код, написанный для решения какой-то подзадачи, на серии тестов.

13. Твёрдо знайте базовые команды и специфику их использования (см. весь текст выше). Например, старайтесь использовать функции, а не скрипты, структуры массивов, а не массивы структур, и т.д.

Ещё больше хороших советов можно найти в [Robbins], [Johnson].

§15.28. Аналоги системы MatLab

GNU **Octave** (http://www.gnu.org/software/octave/),
Scilab (http://www.scilab.org/),
Java-MathLib (http://www.jmathlib.de/),
Euler (http://eumat.sourceforge.net/)
FreeMat (http://freemat.sourceforge.net/) – бесплатные ПО с языком «максимально совместимым» с системой MatLab.
IDL (Interactive Data Language, http://www.ittvis.com/) – коммерческое ПО (область применения аналогична системе MatLab, больше подходит для обработки изображений и визуализации).
O-Matrix (http://www.omatrix.com) – коммерческое ПО, в котором есть режим совместимости с системой MatLab.
LyME (http://www.calerga.com) – ПО, которое позволяет использовать некоторые MatLab-команды на Palm-устройствах.

При составлении заданий и примеров автору очень помогло учебное пособие Питера Аклама [Acklam], а также обсуждения на блоге Лорен Шью [Loren]. Автор позволил себе не указывать у каждого примера источник, поскольку некоторые примеры повторяются на многих сайтах Интернета (и установить их авторство достаточно проблематично). Из русскоязычных источников автору больше всего нравится книга [Потемкин, 1999], которая, возможно, уже немного устарела. В последнее время появляются учебные ресурсы, посвящённые использованию системы MatLab в анализе данных, например [Золотых, MatLab].

ЛИТЕРАТУРА, ССЫЛКИ

Ниже указаны лишь те источники, которые представляют читателю «образовательную» ценность. В них можно найти ссылки на книги и статьи разработчиков рассмотренных методов (формат данного учебного пособия не позволил указать их здесь).

[Местецкий, 2002] *Местецкий Л.М.* Математические методы распознавания образов. – Курс лекций, ВМиК МГУ, кафедра ММП. – 2002. // http://www.ccas.ru/frc/ papers/mestetskii04course.pdf

[Воронцов] *Воронцов К.В.* Курс лекций «Математические методы обучения по прецедентам» // http://www.ccas.ru/voron/, http://www.machinelearning.ru/

[Золотых] *Золотых Н.Ю.* Учебные материалы по машинному обучению http://www.uic.unn.ru/~zny/ml/

[Clop] http://clopinet.com/CLOP/

[Журавлёв, 1978] *Журавлёв Ю.И.* Об алгебраическом подходе к решению задач распознавания или классификации // Пробл. кибернетики. – М.: Наука, 1978. – Вып. 33. – С. 5–68 (статья также напечатана в книге *Журавлёв Ю.И.* Избранные научные труды. – М.: «Магистр», 1998. – 420 с.).

[Корнилов, 2005] *Корнилов Е.Н.* Программирование шахмат и других логических игр. – СПб.: БХВ-Петербург, 2005. – 272 с.

[Люгер, 2005] *Люггер Д.Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем. – М.: Издательский дом «Вильямс», 2005. – 864 с.

[Ветров, Кропотов] Лекции Д.П. Ветрова и Д.А. Кропотова «Байесовские методы машинного обучения» // выложено на [ML]:
http://www.machinelearning.ru/wiki/images/e/e1/BayesML-2007-textbook-1.pdf
http://www.machinelearning.ru/wiki/images/4/43/BayesML-2007-textbook-2.pdf

[Шурыгин, 2009] *Шурыгин А.М.* Математические методы прогнозирования. Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2009. – 180 с.

[Дуда, Харт, 1976] *Дуда Р., Харт П.* Распознавание образов и анализ сцен. – М.: Мир, 1976.

[Ту, Гонсалес, 1978] *Ту Дж., Гонсалес Р.* Принципы распознавания образов. – М.: Мир, 1978.

[Дьяконов, 2006] *Дьяконов А.Г.* Алгебра над алгоритмами вычисления оценок: Учебное пособие. – М.: Издательский отдел ф-та ВМиК МГУ им. М.В. Ломоносова, 2006. – 72с.

[Стрижов, Пташко, 2007] *Стрижов В.В., Пташко Г.О.* Алгоритмы поиска суперпозиций при выборе оптимальных регрессионных моделей // Сообщения по прикладной математике, М.: ВЦ РАН, 2007. – 54с. //
http://strijov.com/papers/strijov06ccas.pdf

[Голяндина, 2004] *Голяндина Н.Э.* Метод «Гусеница»-SSA: анализ временных рядов: Учеб. пособие. – СПб., 2004. – 76 с.

[Васильев, 2002] *Васильев Ф.П.* Методы оптимизации – М.: Факториал, 2002, 823 с.

[Burges, 1998] *Burges C. J. C.* A tutorial on support vector machines for pattern recognition // Data Mining and Knowledge Discovery. – 1998. – Vol. 2, №2. – Pp. 121 – 167. http://citeseer.ist.psu.edu/burges98tutorial.html.

[Gunn, 1998] *Steve R. Gunn* Support Vector Machines for Classification and Regression // Technical Report (University of Southampton, Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, 10 May 1998). http://www.svms.org/tutorials/Gunn1997.pdf

[Хайкин, 2006] *Хайкин С.* Нейронные сети: полный курс, 2-е издание. – М. Издательский дом «Вильямс», 2006. – 1104 с. (*Haykin S.S.* Neural Networks: A Comprehensive Foundation. – Prentice-Hall, second edition, 1998.)

[Минский, Пейперт] *Минский М., Пейперт С.* Персептроны = Perceptrons. – М.: Мир, 1971. – 261 с. (*Minsky M., Papert S.* Perceptrons: an Introduction to Computational Geometry. – MIT Press, 1968.)

[Luke, 2009] *Sean Luke* Essentials of Metaheuristics (A Set of Undergraduate Lecture Notes) // Online Version 0.6 http://cs.gmu.edu/~sean/book/metaheuristics/

[Дюкова, 2003] *Дюкова Е.В.* Дискретные (логические) процедуры распознавания: принципы конструирования, сложность реализации и основные модели. Учебное пособие для студентов математических факультетов педвузов. – М.: Прометей, 2003. – 29 с. // http://www.ccas.ru/frc/papers/djukova03mp.pdf

[Boosting] www.boosting.org

[Clustering] *A.K. Jain, M.N. Murty, P.J. Flynn* Data Clustering: A Review // ACM Computing Surveys, Vol. 31, No. 3, September 1999. // http://www.cs.rutgers.edu/~mlittman/courses/lightai03/jain99data.pdf

[Tübingen] *Lal T.N, Hinterberger T., Widman G., Schroeder M., Hill J., Rosenstiel W., Elger C., Schölkopf B., Birbaumer N.* Methods Towards Invasive Human Brain Computer Interfaces // Advances in Neural Information Processing System – 2005. – V.17. – Pp. 737 – 744. http://eprints.pascal-network.org/archive/00001507/

[BCI 3] http://ida.first.fraunhofer.de/projects/bci/competition_iii/

[Ford] http://home.comcast.net/~nn_classification/

[NN3] www.neural-forecasting-competition.com/NN3/

[NN5] www.neural-forecasting-competition.com/NN5/

[Weka] http://www.cs.waikato.ac.nz/~ml/weka/

[Weka Wiki] http://weka.wikispaces.com/

[Weka, 2009] *Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten* (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1. // http://www.kdd.org/explorations/issues/11-1-2009-07/p2V11n1.pdf

[WekaManual-3-7-1.pdf] *Remco R. Bouckaert, Eibe Frank, Mark Hall, Richard Kirkby, Peter Reutemann, Alex Seewald, David Scuse* WEKA Manual for Version 3-7-1 (файл поставляется вместе с программой Weka версии 3.7.1).

[Sun] http://java.sun.com

[RM] http://www.rapidminer.com/, http://rapid-i.com

[RM KDD, 2006] *Mierswa, I. and Wurst, M. and Klinkenberg, R. and Scholz, M. and Euler, T.*, Yale (now: RapidMiner): Rapid Prototyping for Complex Data Mining Tasks. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), 2006.

[MathWorks] http://www.mathworks.com

[Acklam] *Peter J. Acklam* MATLAB array manipulation tips and tricks // http://home.online.no/~pjacklam

[Loren] *Loren Shure* (блог) Loren on the Art of MATLAB // http://blogs.mathworks.com/loren/

[Griffiths] *David F. Griffiths* An Introduction to Matlab Version 2.3 // The University Dundee DD1 4HN, 2005. – http://maxwell.me.gu.edu.au/spl/matlab-page/matlab_griffiths.pdf

[Sigmon] *Kermit Sigmon* MATLAB Primer // http://terpconnect.umd.edu/~nsw/ench250/primer.htm

[Cambridge] Cambridge University Engineering Department – Matlab // http://www-h.eng.cam.ac.uk/help/tpl/programs/matlab.html

[CVD] The technical note «How Do I Vectorize My Code?» // http://www.mathworks.com/support/tech-notes/1100/1109.html

[Robbins] *Michael Robbins* Good Matlab Programming Practices for the Non-Programmer // http://www.mathworks.com/matlabcentral/fileexchange/2371-good-matlab-programming-practices

[Johnson] *Richard Johnson* MATLAB Programming Style Guidelines // http://www.datatool.com/prod02.htm

[ML] http://www.machinelearning.ru

[Guide] MATLAB® 7 Getting Started Guide http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf

[Потемкин, 1999] *Потемкин В.* Система инженерных и научных расчетов MATLAB 5.x (в 2-х томах). Диалог-МИФИ. 1999.

[Золотых, MatLab] *Золотых Н.Ю.* MATLAB в научной и исследовательской работе http://www.uic.unn.ru/~zny/matlab/

Mihi ipsi scripsi!