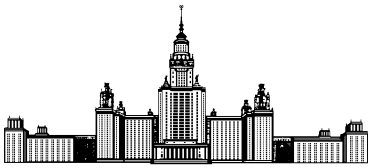
Московский Государственный Университет имени М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Математических Методов Прогнозирования



Курсовая работа

«Обзор алгоритмов бустинга»

«Overview of Boosting Methods»

Работу выполнил:
студент 317 группы
Фонарев Александр Юрьевич

Научный руководитель:
д.ф.-м.н., доцент
Дьяконов Александр Геннадьевич

Москва
Май 2012

Содержание

1 Введение

Исторически считалось, что в машинном обучении обобщающая способность напрямую связана с интуитивной «сложностью» алгоритма. Действительно, всегда можно написать сложный алгоритм, абсолютно точно подстраивающийся под обучающую выборку, но имеющий нулевую обощающую способность. Некоторые даже пытались измерять обощающую способность алгоритма количеством программных команд, которыми этот алгоритм кодировался[1].

Однако эмпирические факты указывали на обратное: адаптивный[2] метод построения композиции простых алгоритмов распознавания позволял существенно увеличить качество, хотя сложность алгоритма при увеличении композиции лишь возрастала. Этот феномен легко проиллюстрировать следующим простым примером. Допустим, у нас есть три алгоритма, каждый из которых верно решает задачу бинарной классификации с вероятностью p, независимо от остальных. Тогда при классификации объекта возможны 8 исходов. Если из данных алгоритмов образовать комитет большинства[3], то вероятность верного ответа полученной композиции равна

$$q = p^3 + 3p^2(1-p) = p^2(3-2p).$$

Построим график зависимости вероятности q правильного ответа композиции от вероятности p правильного ответа исходных алгоритмов (Рис. 1).

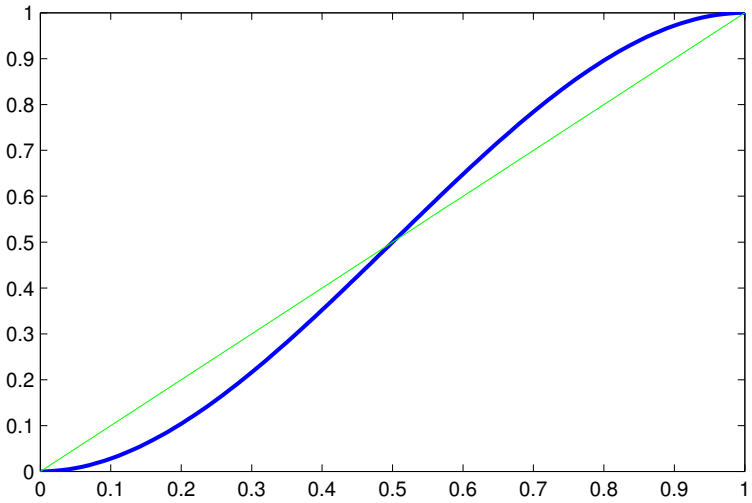


Рис. 1: Жирной линией показан график зависимости вероятности правильного ответа композиции от вероятности правильного ответа базовых алгоритмов

Итак, видно, что, если вероятность правильного ответа базовых алгоритмов хотя бы немного превосходит 0.5, то качество правильно построенной композиции может

[1]Принцип минимальной длины описания [16]

[2]Адаптивный, т.е. последовательный, жадный и минимизирующий ошибки предыдущих алгоритмов

[3]Комитет большинства означает, что объект относят к тому классу, за который «проголосовало» большинство алгоритмов

быть заметно больше качества каждого отдельного алгоритма. Более того, эмпирически было обнаружено, что качество может продолжать увеличиваться при удачном наращивании композиции даже по достижении абсолютно точного предсказания на обучении.

Идея построения композиций алгоритмов была известна уже достаточно давно. Эвристически рассматривались различные возможности голосований алгоритмов, семейства алгоритмов зачастую были фиксированны, и обычно конструкции в целом не имели возможностей для настройки. Первая общая теория композиций алгоритмов была предложена Журавлевым Ю. И. и его учениками в алгебраическом подходе к распознаванию [11]. Однако эта теория оказалась малоприменима к практике.

В конце 80ых начали появляться работы с исследованием проблемы связи слабой и сильной обучаемости алгоритмов. Слабая обучаемость алгоритма означает, что за полиномиальное время возможно построить алгоритм распознавания, точность которого будет хотя бы немного больше 50%. Под сильной обучаемостью подразумевается, что возможно за полиномиальное время построить алгоритм, который бы мог давать сколь угодно точные результаты. Исследования показали, что сильная обучаемость эквивалентна слабой, поскольку любую слабую модель можно усилить, построив правильную композицию. В 1996 эти идеи сформировались и приобрели законченнную форму в виде алгоритма AdaBoost [7]. Этот алгоритм быстро набрал популярность благодаря своей простосте и эффективности. Несколькими годами позже появилось важное обобщение этого алгоритма – градиентный бустинг [1].

2 Градиентный бустинг

2.1 Постановка задачи

Рассмотрим сразу наиболее общий метод бустинга [1], частными случаями или модификациями которого, так или иначе, являются все современные методы бустинга.

Рассмотрим задачу распознавания объектов из многомерного пространства X с пространством меток Y. Пусть нам дана обучающая выборка $\{x_i\}_{i=1}^N$, где $x_i \in X$. И пусть на ней известны истинные значения меток каждого объекта $\{y_i\}_{i=1}^N$, где $y_i \in Y$. Необходимо построить распознающий оператор, который как можно более точно сможет предсказывать метки для каждого нового объекта $x \in X$.

Пусть нам задано некоторое семейство базовых алгоритмов H, каждый элемент $h(x;a) \in H : X \to R$ которого определяется некоторым вектором параметров $a \in A$.

2.2 Построение композиции

Будем искать финальный алгоритм классификации в виде композиции

$$F_M(x) = \sum_{m=1}^{M} b_m h(x;a_m), b_m \in R, a_m \in A.$$

Однако подбор оптиматильного набора параметров $\{a_m, b_m\}_{m=1}^M$ – очень трудоемкая задача. Поэтому мы будем пытаться построить такую композицию путем жадного наращивания, каждый раз добавляя в сумму слагаемое, являющееся наиболее оптимальным алгоритмом из возможных. Будем считать, что нами уже построен классификатор F_{m-1} длины $m-1$. Таким образом задача сводится к поиску пары наиболее оптимальных параметров $\{a_m, b_m\}$ для классификатора длины m:

$$F_m(x) = F_{m-1}(x) + b_m h(x;a_m), b_m \in R, a_m \in A.$$

Оптимальность здесь понимается в соответствии с принципом явной максимизации отступов [3]. Это означает, что вводится некоторая функция потерь $L(y_i, F_m(x_i)), i = \overline{1,N}$, показывающая, насколько "сильно"предсказанный ответ $F_m(x_i)$ отличается от правильного ответа y_i. И затем минимизируется функционал ошибки

$$Q = \sum_{i=1}^{N} L(y_i, F_m(x_i)) \to \min.$$

Заметим, что функционал ошибки Q – вещественная функция, зависящая от точек $\{F_m(x_i)\}_{i=1}^N$ в N-мерном пространстве, и нам необходимо решить задачу минимизации этого функционала. Сделаем это, реализуя один шаг метода градиентного спуска [17]. В качестве точки, для которой мы будем искать оптимальное приращение, рассмотрим F_{m-1}. Найдем градиент функционала ошибки:

$$\nabla Q = \left[\frac{\partial Q}{\partial F_{m-1}}(x_i)\right]_{i=1}^N = \left[\frac{\partial(\sum_{i=1}^N L(y_i, F_{m-1}))}{\partial F_{m-1}}(x_i)\right]_{i=1}^N = \left[\frac{\partial L(y_i, F_{m-1})}{\partial F_{m-1}}(x_i)\right]_{i=1}^N.$$

Таким образом, в силу метода градиентного спуска, наиболее выгодно добавить новое слагаемое в классификатор следующим образом:

$$F_m = F_{m-1} - b_m \nabla Q, b_m \in R,$$

где b_m подбирается линейным поиском по вещественным числам R:

$$b_m = \underset{b \in R}{\operatorname{argmin}} \sum_{i=1}^{N} L(F_{m-1}(x_i) - b \triangledown Q_i).$$

Однако $\triangledown Q$ представляет из себя лишь вектор оптимальных значений для каждого объекта x_i, а не базовый алгоритм из семейства H, определенный $\forall x \in X$. Поэтому нам необходимо найти $h(x, a_m) \in H$ наиболее похожий на $\triangledown Q$. Сделаем это, опять минимизируя функционал ошибки, основанный на принципе явной максимизации отступов:

$$a_m = \underset{a \in A}{\operatorname{argmin}} \sum_{i=1}^{N} L(\triangledown Q_i, h(x_i, a)) \equiv \text{обучить} \left(\{x_i\}_{i=1}^{N}, \{\triangledown Q_i\}_{i=1}^{N} \right),$$

что просто соотвествует базовому алгоритму обучения. Далее найдем коэффициент b_m, используя линейный поиск:

$$b_m = \underset{b \in R}{\operatorname{argmin}} \sum_{i=1}^{N} L(F_{m-1}(x_i) - b h(x_i, a_m)).$$

2.3 Псевдокод алгоритма

Это и есть алгоритм градиентного бустинга для построения композиции алгоритмов распознавания. В завершение приведем псевдокод вышеописанного алгоритма:

Algorithm 1 Градиентный бустинг

Вход: $\{x_i\}_{i=1}^{N}, \{y_i\}_{i=1}^{N}, M$;
Выход: $F_M(x)$;
  $F_0(x) = learn\left(\{x_i\}_{i=1}^{N}, \{y_i\}_{i=1}^{N}\right)$;
  для $m = \overline{1, M}$
    $\triangledown Q = \left[\frac{\partial L(y_i, F_{m-1})}{\partial F_{m-1}}(x_i)\right]_{i=1}^{N}$;
    $a_m = \text{обучить}\left(\{x_i\}_{i=1}^{N}, \{\triangledown Q_i\}_{i=1}^{N}\right)$;
    $b_m = \operatorname{argmin}_{b \in R} \sum_{i=1}^{N} L(F_{m-1}(x_i) + b h(x_i, a_m))$;
    $F_m(x) = F_{m-1}(x) + b_m h(x; a_m)$;

3 Частные случаи градиентного бустинга

3.1 Решение задачи регрессии

Рассмотрим частные случаи для решения задач регрессии.

3.1.1 Метод наименьших квадратов

Рассмотрим следующую функцию потерь:

$$L(y, F) = \frac{(y - F)^2}{2}.$$

Тогда ∇Q алгоритме градиентного бустинга примет следующий вид:

$$\nabla Q_i = y_i - F_{m-1}(x_i).$$

Этот случай бустинга называется LS-boosting[4] или GentleBoost.

3.1.2 Минимизация среднего модуля отклонения

Далее рассмотрим еще более простой вариант функции потерь – модуль отклонения:

$$L(y, F) = |y - F|.$$

Тогда ∇Q примет следующий вид:

$$\nabla Q_i = sign(y_i - F_{m-1}(x_i)).$$

Тогда линейный поиск приобретает следующий вид:

$$b_m = \underset{b \in R}{\operatorname{argmin}} \sum_{i=1}^{N} |y_i - F_{m-1}(x_i) - bh(x_i, a_m)| = \underset{b \in R}{\operatorname{argmin}} \sum_{i=1}^{N} |h(x_i, a_m)| \left| \frac{y_i - F_{m-1}(x_i)}{h(x_i, a_m)} - b \right| =$$

$$= median_W \left\{ \frac{y_i - F_{m-1}(x_i)}{h(x, a_m)} \right\}_{i=1}^{N}, W = \{h(x_i, a_m)\}_{i=1}^{N},$$

где $median_W$ – взвешенное медианное значение набора. Т.е. задача свелась к задаче поиска порядковой статистики в массиве, а для нее существуют решения за $O(N)$ [18]. Что заметно быстрее линейного поиска по вещественным числам. Такая разновидность бустинга называется LAD-boosting[5].

3.1.3 Бустинг над решающими деревьями

До этого момента мы не уточняли, какие именно базовые алгоритмы мы рассматривали. Это могли быть, например, линейные методы восстановления регрессии. Сейчас же рассмотрим в качестве базового семейства алгоритмов регрессионные решаюшие деревья из J вершин [9]. Здесь нам не особенно важны методы построения решающих деревьев. Важен тот факт, что каждое решающее дерево имеет J листовых

[4]LS-boosting от англ. Least-Squares Boosting

[5]LAD-boosting от англ. Least-Absolute-Deviation Boosting

вершин, соответствующие J непересекающимся областям $\{R_j\}_{j=1}^J$, на которые разбивается пространство объектов X. Каждой листовой вершине соответствует некоторое значение регрессии b_j, которое будет ответом классификатора в случае попадания анализируемого объекта в соотвествующую область. Можно записать этот факт следующей формулой:

$$h(x, \{a_j, R_j\}_{j=1}^J) = \sum_{j=1}^J a_j I[x \in R_j],$$

где $I[A]$ – индикатор события A. Видно, что в этой сумме ровно одно слагаемое будет ненулевым. Тогда добавление слагаемого в градиентном бустинге будет происходить следующим образом:

$$F_m(x) = F_{m-1}(x) + b_m \sum_{j=1}^J a_{jm} I[x \in R_j] =$$

$$= F_{m-1}(x) + \sum_{j=1}^J c_{jm} I[x \in R_j], c_{jm} = a_{jm} b_m.$$

Таким образом мы просто прибавляем к имеющему алгоритму некоторое другое решающее дерево. Попробуем найти оптимальные значения c_{jm} для уже построенных R_j:

$$\{c_{jm}\}_{j=1}^J = \underset{\{c_j\}_{j=1}^J}{\operatorname{argmin}} \sum_{i=1}^N L\left(y_i, F_{m-1}(x) + \sum_{j=1}^J c_j I[x \in R_j]\right).$$

И, поскольку области R_j не пересекаются, можем переписать эту формулу в следующем виде:

$$c_{jm} = \underset{c}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + c).$$

Таким образом вместо того чтобы выполнять линейный поиск коэффициента перед новым слагаемым, как в классическом градиентном бустинге, мы полностью перенастраиваем параметры дерева с фиксированными $\{R_j\}$. Это позволяет строить более качественную композицию. Такая разновидность бустинга называется Tree-boost.

Если в бустинге над регрессионными деревьями взять LAD функцию потерь, то оптимальные параметры деревьев примут следующий вид:

$$c_{jm} = median_{x_i \in R_{jm}}\{y_i - F_{m-1}(x_i)\}.$$

Данный алгоритм является достаточно устойчивым к шуму: во-первых, ∇Q_i принимает только значения $\{+1, -1\}$, а во-вторых, обновление параметров происходит засчет взятия медианных значений, а медианы устойчивы к выбросам. Также этот алгоритм может похвастаться высокой скоростью работы засчет быстрого подбора параметров.

3.1.4 M-бустинг

В некоторых задачах выгоднее использовать не квадратичную функцию потерь, а ее модификацию, функцию потерь Хубера [10]. Она имеет вид квадратичной функции потерь вблизи нуля и линейной функции потерь вдали от нуля:

$$L(y,F) = \begin{cases} \frac{1}{2}(y-F)^2, & |y-F| \leqslant \sigma \\ \sigma(|y-F| - \frac{\sigma}{2}), & |y-F| > \sigma. \end{cases}$$

Это позволяет использовать метод наименьших квадратов для небольших ошибок и линейную функцию потерь для больших ошибок. Это позволяет существенно изменить алгоритм для распределений ошибки с тяжелыми хвостами, что очень полезно для многих задач (например, для задач с сильно зашумленными данными).

Градиент примет вид:

$$\nabla Q_i = \begin{cases} y_i - F_{m-1}(x_i), & |y_i - F_{m-1}(x_i)| \leqslant \sigma \\ \sigma sign(y_i - F_{m-1}(x_i)), & |y_i - F_{m-1}(x_i)| > \sigma. \end{cases}$$

Можно использовать следующее приближение для случая решающих деревьев:

$$c_{jm} = \tilde{r}_{jm} + \frac{1}{|R_{jm}|} \sum_{x_i \in R_{jm}} sign(r_{m-1}(x_i) - \tilde{r}_{jm}) \min(\sigma_m, abs(r_{m-1}(x_i) - \tilde{r}_{jm}))),$$

где введены обозначения $r_{m-1}(x_i) \equiv y_i - F_{m-1}(x_i)$ и $\tilde{r}_{jm} \equiv median_{x_i \in R_{jm}}\{r_{m-1}(x_i)\}$.

Параметр σ для функции Хубера подбирается так, чтобы учитывать возможные распределения ошибки с тяжелыми хвостами. Поскольку распределение ошибок на каждой итерации бустинга разное, то σ_m подбирается заново на каждой итерации бустинга. Для этого используется следующая формула

$$\sigma_m = quantile_\alpha\{|y_i - F_{m-1}(x_i)|\}_{i=1}^N,$$

где α может подбираться из разных соображений. Например, можно искать такую точку α, чтобы не было заметного ухудшения качества по сравнению с обычным методом наименьших квадратов.

Бустинг с использованием функции потерь Хубера называется M-бустингом.

3.2 Решение задачи классификации

Идея бустинга также применима для задачи классификации. В случае бинарной классификации это означает, что $Y = \{-1, +1\}$. Тогда часто подразумевается, что каждый алгоритм $h \in H$ возващает вещественную «степень» принадлежности объекта к некоторому классу, а результирующий ответ $\tilde{F}$ получается применением порогового правила к композициии.

3.2.1 Веса объектов вместо градиента

В случае классификации обычно используется функция потерь от одного аргумента:

$$L(y,F) = L(yF),$$

т.е. отступ заменяется произведением настоящего класса и предсказанного значения.

В таком случае существует немного другой взгляд на подход градиентного бустинга, нежели который был описан выше. Под градиентом функционала ошибки

можно подразумевать вектор весов обучающих объектов поэлементно умноженный на верные значения классов:

$$\nabla Q = \left[\frac{\partial L(y_i F_{m-1})}{\partial F_{m-1}}(x_i)\right]_{i=1}^{N} = \left[y_i \frac{\partial L(y_i F_{m-1})}{\partial (y_i F_{m-1})}(x_i)\right]_{i=1}^{N} = [y_i w_i]_{i=1}^{N},$$

где $w_i \equiv \frac{\partial L(y_i F_{m-1})}{\partial (y_i F_{m-1})}(x_i)$. Тогда алгоритм обучения в соответствии с принципом максимизации отступов приобретает следующий вид:

$$h(x, a_m) = \text{обучить}\left(\{x_i\}_{i=1}^{N}, \{\nabla Q_i\}_{i=1}^{N}\right) =$$

$$= \underset{a_m \in A}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(\nabla Q_i h(x_i, a_m)\right) = \underset{a_m \in A}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i w_i h(x_i, a_m)\right).$$

Таким образом w_i можно рассматривать с точки зрения весов (степени «важности»), которые придаются объектам и учитываются при обучении каждого базового алгоритма. Этот взгляд сложился исторически раньше, чем градиентный подход. К тому же, он более интуитивно понятен.

3.2.2 AdaBooost

Разновидность бустинга AdaBoost подразумевает, что используется экспоненциальная функция потерь.

$$L(y, F) = exp(-yF).$$

AdaBoost является первым теоретически исследованным вариантом бустинга. Для него впервые была доказана основная теорема бустинга с аналитически вычисленными оптимальными b_m на каждом шаге. Ранняя версия AdaBoost рассматривала композицию из алгоритмов $h \in H$, которые возвращают лишь значения из $Y = \{-1, +1\}$[6]. Затем он был обобщен на случай, когда $h \in H$ возвращают вероятность принадлжености классу $\{+1\}$[7].

Не вдаваясь в теоретические обоснования, сразу выпишем псевдокод дискретного AdaBoost (Algorithm 2). Здесь используется вышеописанная идея с весами объектов, однако здесь они нормируются на каждом шаге. Также здесь под p_m подразумевается частота ошибок, которые выдает текущий базовый алгоритм.

3.2.3 LogitBoost

Попробуем строить композицию бинарных классификаторов, используя идеи логистической регрессии [19]. Функция потерь имеет вид:

$$L(y, F) = \log(1 + \exp(-2yF)),$$

где

$$F(x) = \frac{1}{2} \log \frac{P(y = 1|x)}{P(y = -1|x)}.$$

[6]Дискретный AdaBoost или Discrete AdaBoost

[7]Вещественный AdaBoost или Real AdaBoost

Algorithm 2 Дискретный AdaBoost

Вход: $\{x_i\}_{i=1}^N, \{y_i\}_{i=1}^N, M$;
Выход: $F_M(x)$
$w_i = \frac{1}{N}, i = \overline{1, N}$
для $m = \overline{1, M}$
$a_m = $ обучить$(\{x_i, y_i, w_i\}_{i=1}^N)$;
$p_m = \frac{1}{N} \sum_{i=1}^N I[y_i \neq h(x_i, a_m)]$;
$b_m = \frac{1}{2} \log \frac{p_m}{1-p_m}$;
$w_i = w_i \exp\left(-y_i b_m h(x_i, a_m)\right), \overline{1, N}$;
$w_i = \frac{w_i}{\sum_{j=1}^N w_i}, \overline{1, N}$;
$F_M(x) = \sum_{m=1}^M b_m h(x, a_m)$;

Такая разновидность бустинга называется LogitBoost. Тогда, например, в случае решающих деревьев параметры каждого дерева будут пересчитываться следующим образом:

$$c_{jm} = \underset{c \in C}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} \log(1 + \exp(-2y_i(F_{m-1}(x_i) + c)).$$

Эта задача решается за $O(N)$, что в некоторых задачах неприемлимо долго. Поэтому иногда значения удобно аппроксимировать одним шагом алгоритма Ньютона-Рафсона [20]:

$$c_{jm} = \frac{\sum_{x_i \in R_{jm}} \nabla Q_i}{\sum_{x_i \in R_{jm}} |\nabla Q_i|(2 - |\nabla Q_i|)}.$$

3.2.4 Мультиклассовая классификация

Идея бустинга для бинарной классификации легко обобщается на случай K классов. Вводится следующая функция потерь:

$$L(y, F) = -\sum_{i=1}^{k} y_i \log p_i(x).$$

Здесь $y_i \in \{0, 1\}$ показывает принадлежность объекта классу i, а p_i показывает вероятность принадлежности объекта классу i, получаемая в ходе работы логистической регрессии. Запишем формулы для классификатора класса K мультиклассовой логистической регрессии:

$$f_k(x) = \log p_k(x) - \frac{1}{K} \sum_{l=1}^{K} \log p_l(x).$$

После преобразований можно получить

$$\nabla Q_i = y_{ik} - p_{k,m-1}(x_i).$$

Если представляемая задача оказывается слишком вычислительно сложной, то в случае решающих деревьев возможно использование первого шага алгоритма Ньютона-Раффсона в качестве аппроксимизации:

$$c_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} \nabla Q_{ik}}{\sum_{x_i \in R_{jkm}} |\nabla Q_{ik}|(1 - |\nabla Q_{ik}|)}$$

Итовый классификатор ищет такой класс для объекта, чтобы вероятность принадлежности другим классам была минимальна:

$$k(x) = \underset{k \in [1,K]}{\operatorname{argmin}} \sum_{\tilde{k}=1}^{K} c(k, \tilde{k}) p_{\tilde{k}M}(x).$$

Здесь за $p_{km}(x)$ обозначена вероятность принадлжености классу k в результате m-й итерации бустинга. За $c(k, \tilde{k})$ обозначена функция стоимости ошибки, если предсказано, что объект принадлежит классу k, хотя на самом деле он принадлежит классу $\tilde{k}$.

4 Модификации бустинга

4.1 Использование вклада объектов в LogitBoost

Запишем функционал ошибки в LogitBoost:

$$Q = \sum_{i=1}^{N} \log(1 + \exp(-2y_i F_{m-1}(x_i)) \exp(-2y_i b h(x_i, a))).$$

Видно, что при больших значениях $y_i F_{m-1}(x_i)$ вклад множителя с искомыми параметрами будет практически несущественен. Это элементы, на которых обучение уже происходит достаточно хорошо, и увеличение отступа для них не имеет особого смысла. Поэтому иногда из рассмотрения на новом шаге специально выбрасывают такие элементы.

4.2 Стохастический градиентный бустинг

Основная причина эффективности бустинга в том, что алгоритм на каждой итерации строит базовый алгоритм, который действительно эффективен лишь на части подвыборки. Этот принцип можно усилить, сделав небольшую модификацию бустинга. А именно, на каждом шаге алгоритма новое слагаемое считается опираясь не на всю обучающую выборку, а лишь на случайную подвыборку фиксированного размера. Эта идея очень популярна и эффективна. Она является объединением техник градиентного бустинга и беггинга[8].

Также можно брать не случайную подвыборку объектов, а еще и случайную подвыборку признаков объектов. Это называется техникой случайных подпространств[9].

Результаты работы таких модификаций зачастую заметно превосходят по качеству различные нестохастические варианты.

4.3 Устойчивость и регуляризация

У градиентного бустинга есть один важный внешний параметр M. При увеличении M повышается устойчивость и точность у результирующей композиции F_M. Можно рассматривать это как первый параметр регуляризации.

Теперь немного модифицируем шаг алгоритма градиентного бустинга, на котором добавляется новый базовый алгоритм в композицию. А именно, добавим дополнительный константный коэффициент ν для каждого базового алгоритма:

$$F_m = F_{m-1} + \nu b_m h(x; a_m), \nu \in [0, 1].$$

При уменьшении ν каждый новый базовый алгоритм, будь он достаточно точен или плох, будет учитываться не полностью. Это повышает устойчивость получаемой композиции. Параметр ν можно рассматривать как второй параметр регуляризации[10].

Заметим, что при уменьшении ν требуется увеличение общего количества алгоритмов M. Таким образом вычислительная сложность построения композиции увеличивается, однако увеличивается и финальное качество.

[8] Беггинг от англ. bagging [13]

[9] Метод случайных подпространств, RSM – от англ. Random Subspace Method [15]

[10] В англоязычной литературе и спецификациях различных алгоритмов он называется shrinkage

4.4 Борьба с шумом

На каждой итерации градиентного бустинга алгоритм стремится максимально исправить все ошибки на обучении. Однако это бессмысленно при наличии шума в исходных данных и ведет к переобучению. Проблему можно решить, учитывая веса объектов на каждой итерации, ведь по ним можно судить о сложности обучения на них. Действительно, большой вес у объекта показывает, что предыдущие алгоритмы плохо работали на нем и, возможно, этот объект шумовой.

Такая гипотеза порождает широкое семейство алгоритмов бустинга с модификацией для борьбы с шумом. Наиболее популярным вариантом является алгоритм BrownBoost, являющийся модификацией AdaBoost с весовым подавлением шума.

4.5 Подбор коэффициентов с помощью SVM

Пусть на некотором шаге построено m базовых алгоримов $\{h_i(x)\}_{i=1}^m$. Этот вектор можно принять за новое признаковое описание объекта x и подобрать коэффициенты b_i как коэффициенты оптимальной разделяющей гиперплоскости в этом новом пространстве. Для этого можно использовать различные алгоритмы, но наиболее удачно использование метода опорных векторов SVM, поскольку он максимизирует суммарный отступ между классами [14].

Также стоит учитывать, чтобы коэффициенты b_i были положительны, поскольку $b_i \leqslant 0$ показывает, то алгоритм h_i показывает такое плохое качество, что его надо брать с отрицательным знаком. Имеет смысл вообще занулять коэффициент при таких алгоритмах.

Использовать эту стратегию подсчета коэффициентов можно на полученной в результате композиции из M алгоритмов. Также можно использовать ее на каждом шаге бустинга. Таким образом коэффициент каждый раз будет подбираться не жадно, а оптимально по уже построенной композиции. Также возможно добавление параметров регуляризации для SVM.

5 Заключение

В работе были рассмотрены самые важные идеи и принципы, лежащие в основе бустинга.

5.1 Преимущества бустинга

На сегодняшний день является одним из самых мощных алгоритмов распознавания. Это достигается благодаря вышеупомянутой адаптивной технике построения композиции. К тому же, бустинг предоставляет множество возможностей для вариаций.

Во-первых, можно рассматривать различные функции потерь. Это позволяет решать как задачи классификации, так и задачи регрессии. К тому же, возможность выбора произвольной функции потерь позволяет акцентировать внимание на особенностях данных в задаче.

Во-вторых, возможно рассмотрение любого семейства базовых алгоритмов. А это, опять же, дает широкиие возможности учета особенностей даннной задачи. Бустинг над решающими деревьями считается одним из наиболее эффективных вариантов бустинга. А учитывая, что решающие деревья в свою очередь тоже используют базовые алгоритмы (например, пороговые, линейные и т.п.), в результате получается огромное количество вариантов для настройки.

В-третьих, благодаря достаточной простоте метода и четкому математическому обоснованию, в каждой конкретной вариации бустинга не сложно провести некоторые математические и алгоритмические оптимизации, которые заметно ускорят работу алгоритма.

5.2 Недостатки бустинга

Разумеется, бустинг не лишен недостатков.

Во-первых, бустинг – трудоемкий метод, и работает он достаточно медленно. Зачастую требуется построение сотен или даже тысяч базовых алгоритмов для композиции.

Во-вторых, без дополнительных модификаций он имеет свойство полностью подстраиваться под данные, в том числе под ошибки и выбросы в них.

В-третьих, идея бустинга обычно плохо применима к построению композиции из достаточно сложных и мощных алгоритмов. Построение такой композиции занимает очень много времени, а качество существенно не увеличивается.

В-четвертых, результаты работы бустинга сложно интерпретируемы, особенно если в композицию входят десятки алгоритмов.

Список литературы

[1] Friedman J. Greedy Function Approximation: A Gradient Boosting Machine. — IMS 1999 Reitz Lecture.

[2] Mark Culp, Kjell Johnson, George Michailidis. Ada: an R Package for Stochastic Boosting. — Journal of Statistical Software, Volume VV, Issue II.

[3] Воронцов К. В. Машинное обучение (курс лекций).

[4] Friedman J. Stochastic Gradient Boosting. — 1999.

[5] Friedman J., Hastie T., Tibshirani R. Additive Logistic Regression: a Statistical View of Boosting. — 1998.

[6] Терехов С. А. Гениальные комитеты умных машин. Нейроинформатика-2007.

[7] Freund Y., Shapire R. Experiments with a New Boosting Algorithm. —1996.

[8] Sochman J., Matas J. AdaBoost. — Center for Machine Perception Czech Technical University, Prague.

[9] Breiman L., Friedman J., Olshen R., Stone C. Classification and Regression Trees. — Wadsworth, 1983.

[10] Huber P. Robust Estimation of a Location Parameter. — Annals of Statistic, 1964.

[11] Журавлёв, Ю. И. Об алгебраическом подходе к решению задач распознавания или классификации. — Проблемы кибернетики: Вып.33. — 1978. — С. 5–68.

[12] Дьяконов А. Г. Алгебра над алгоритмами вычисления оценок (учебное пособие). — Москва, 2005.

[13] Breinman L. Bagging Predictors. — Machine Learning, 24, 123–140, 1996.

[14] Ratsch G., Scholk B., Mika S., Muller K.-R. SVM and Boosting: One Class. — 2000.

[15] Ho, Tin. The Random Subspace Method for Constructing Decision Forests. — IEEE Transactions on Pattern Analysis and Machine Intelligence, 1998.

[16] `http://www.mdl-research.org/`

[17] `http://en.wikipedia.org/wiki/Gradient_descent`

[18] `http://e-maxx.ru/algo/kth_order_statistics`

[19] `http://en.wikipedia.org/wiki/Logistic_regression`

[20] `http://en.wikipedia.org/wiki/Newton's_method`

[21] Воронцов К. В. LaTeX2e в примерах. — 2005.

[22] Львовский С. М. Набор и вёрстка в пакете LaTeX. — М., Космосинформ, 1994.