ВМК

Задание 2. Метрические алгоритмы классификации

Практикум 317 группы, 2015

Начало выполнения задания: 13 октября 2015 года.
Срок сдачи: **27 октября 2015 года, 23:59.**

Задание

Данное задание направлено на ознакомление с метрическими алгоритмами классификации, а также методами работы с текстовой информацией.

Задание должно быть выполнено в IPython Notebook, формат сдачи — .ipynb и при необходимости .py файлы. Каждая ячейка в .ipynb должна быть прокомментирована с помощью `markdown` ячеек. За отсутствие анализа экспериментов и выводов балл будет снижаться!

1. Загрузить датасет MNIST при помощи функции `sklearn.datasets.fetch_mldata("MNIST original")`.

2. Разбить датасет на обучающую выборку (первые 60 тыс. объектов) и тестовую выборку (10 тыс. последних объектов). Ответы на тестовой выборке не следует использовать ни в каких экспериментах, кроме финального.

3. Визуализировать по 5 случайных объектов из каждого из 10 классов. Воспользуйтесь методами `np.reshape`, `pyplot.subplot`, и `pyplot.imshow` с параметром `cmap="Greys"`. Также можно убрать оси координат при помощи команды `pyplot.axis("off")`.

4. Исследовать, каким точным алгоритмом поиска ближайших соседей следует пользоваться в различных ситуациях. Будем искать 5 ближайших соседей в обучающей выборке для тестовой выборки (при этом ответы на тестовой выборке не используются!). Метрика евклидова. Число признаков: 10, 20, 100. Подмножество признаков выбирается один раз, случайно. Алгоритмы поиска ближайших соседей:

   (a) Собственная реализация на основе кода подсчёта евклидова расстояния между двумя множествами точек из задания №1

   (b) `sklearn.neighbors.NearestNeighbors(algorithm='brute')`

   (c) `sklearn.neighbors.NearestNeighbors(algorithm='kd_tree')`

   (d) `sklearn.neighbors.NearestNeighbors(algorithm='ball_tree')`

   Замечание 1. При поиске k ближайших соседей некоторые методы строят в памяти матрицу попарных расстояний обучающей выборки и тестовой выборки. Рекомендуем написать функцию, которая ищет ближайших соседей блоками, то есть делает запросы ближайших соседей для первых N тестовых объектов, затем для следующих N, и так далее, и в конце объединяет полученные результаты.

   Замечание 2. Для оценки времени долго работающих функций можно пользоваться либо командой `time.clock()`, либо magic-командой `%time`, которая запускает код лишь один раз.

5. Реализовать генерацию индексов обучающей и валидационной выборки для кросс-валидации с n фолдами. Не разрешается использовать модуль `sklearn.cross_validation`.

6. Пусть дана обучающая и валидационная выборка. Реализовать оценку точности метода k ближайших соседей по валидационной выборке для нескольких параметров k: $[k_1, \ldots, k_n]$, $k_1 < k_2 < \cdots < k_n$. Сложность алгоритма для одного объекта из валидационной выборки должна иметь порядок $O(k_n)$. Нельзя использовать класс `sklearn.neighbors.KNeighborsClassifier`, можно пользоваться готовыми реализациями поиска ближайших соседей.

7. Оценить по кросс-валидации с 3 фолдами точность (долю правильно предсказанных ответов) и скорость метода k ближайших соседей в зависимости от следующих факторов:

   (a) k от 1 до 10 (только влияние на точность).

   (b) Используется евклидова или косинусная метрика.

8. Реализовать взвешенный метод k ближайших соседей, где голос объекта равен $1/(distance + \varepsilon)$, где ε — малое число. Сравнить с методом без весов при тех же фолдах и тех же параметрах.

9. Применить лучший алгоритм к исходной обучающей и тестовой выборке. Подсчитать точность. Сравнить с точностью по кросс-валидации. Сравнить с указанной в Интернете точностью лучших алгоритмов на данной выборке.

10. Визуализировать несколько объектов из тестовой выборки, на которых были допущены ошибки. Проанализировать и указать их общие черты. Построить и проанализировать матрицу ошибок (confusion matrix). Можно воспользоваться функцией `sklearn.metrics.confusion_matrix`.

11. Загрузить обучающую выборку датасета 20 newsgroups при помощи метода `sklearn.datasets.fetch_20newsgroups`. Убрать все заголовки, подписи и цитаты, используя аргумент `remove`.

12. Перевести во всех документах все буквы в нижний регистр. Заменить во всех документах символы, не являющиеся буквами и цифрами, на пробелы. Полезные функции: `str.lower`, `str.isalnum`.

13. Разбить каждый документ на термы по пробельным символам (пробелы, переносы строки). Полезная функция: `str.split`.

14. Преобразовать датасет в разреженную матрицу `scipy.sparse.csr_matrix`, где значение x в позиции (i, j) означает, что в документе i слово j встретилось x раз. Необходимо воспользоваться наиболее эффективным конструктором `csr_matrix((data, indices, indptr), shape=(M, N))`.

    Замечание. Не забудьте указать параметр `shape`, так как число используемых термов в тестовой выборке может отличаться от числа термов в обучении.

15. Произвести tf-idf преобразование датасета при помощи `sklearn.feature_extraction.text.TfidfTransformer`. Используйте параметры по умолчанию.

16. Оценить точность (долю правильно предсказанных ответов) и скорость метода k ближайших соседей при помощи кросс-валидации с 3 фолдами. Требования к реализации – аналогично пункту 6. Исследуйте на одних и тех же фолдах влияние следующих факторов:

    (a) k от 1 до 10 (только влияние на точность).

    (b) Используется евклидова или косинусная метрика.

    (c) Используется ли преобразование tf-idf.

    (d) Используются взвешенный или невзвешенный метод k ближайших соседей.

17. Загрузите тестовую выборку (параметр `subset` метода `fetch_20newsgroups`). Примените лучший алгоритм к тестовой выборке. Сравнить точность с полученной по кросс-валидации.

18. Вывести несколько документов из тестовой выборки, на которых были допущены ошибки. Проанализировать их. Построить и проанализировать матрицу ошибок (confusion matrix). Можно воспользоваться функцией `sklearn.metrics.confusion_matrix`.

19. Сделать выводы: в каких случаях следует пользоваться каким алгоритмом, какие параметры важно оптимизировать, какие ошибки допускают алгоритмы.