ВМК

Задание 1. Метрические алгоритмы классификации

Практикум 317 группы, 2020

Начало выполнения задания: 5 октября 2020 года, 06:00.
Мягкий дедлайн: **19 октября 2020 года, 06:00.**
Жёсткий дедлайн: **26 октября 2020 года, 06:00.**

Формулировка задания

Данное задание направлено на ознакомление с метрическими алгоритмами классификации, а также методами работы с изображениями. В задании необходимо:

1. Написать на языке Python собственные реализации метода ближайших соседей и кросс-валидации. Реализации должны соответствовать требованиям, описанным ниже.
2. Провести описанные ниже эксперименты с датасетом изображений цифр MNIST.
3. Написать отчёт о проделанной работе (формат PDF). Отчёт должен быть подготовлен в системе LaTeX.
4. В систему проверки anytask сдаётся .zip архив с модулями с написанным кодом, jupyter-notebook с кодом экспериментов (может быть не структурирован, проверяется при наличии вопросов к результатам экспериментов) и отчёт. Архив необходимо назвать `task1_фамилия_имя.zip`.

Список экспериментов

Эксперименты этого задания необходимо проводить на датасете MNIST. Загрузить датасет можно при помощи функции `sklearn.datasets.fetch_mldata("MNIST original")`. Датасет необходимо разбить на обучающую выборку (первые 60 тыс. объектов) и тестовую выборку (10 тыс. последних объектов).

1. Исследовать, какой алгоритм поиска ближайших соседей будет быстрее работать в различных ситуациях. Для каждого объекта тестовой выборки найти 5 его ближайших соседей в обучающей для евклидовой метрики. Для выборки нужно выбрать подмножество признаков, по которому будет считаться расстояние, размера 10, 20, 100 (подмножество признаков выбирается один раз для всех объектов, случайно). Необходимо проверить все алгоритмы поиска ближайших соседей, указанные в спецификации к заданию.

   Замечание. Для оценки времени долго работающих функций можно пользоваться либо функциями из модуля `time`, либо magic-командой `%time`, которая запускает код лишь один раз.

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

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

   (b) Используется евклидова или косинусная метрика.

3. Сравнить взвешенный метод k ближайших соседей, где голос объекта равен $1/(distance + \varepsilon)$, где $\varepsilon - 10^{-5}$, с методом без весов при тех же фолдах и параметрах.

4. 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. Построить и проанализировать матрицу ошибок (confusion matrix). Визуализировать несколько объектов из тестовой выборки, на которых были допущены ошибки. Проанализировать и указать их общие черты.

   Замечание. Можно воспользоваться функцией `sklearn.metrics.confusion_matrix`. Для визуализации можно воспользоваться `pyplot.subplot`, и `pyplot.imshow` с параметром `cmap="Greys"`. Также можно убрать оси координат при помощи команды `pyplot.axis("off")`.

5. Размножить обучающую выборку с помощью поворотов, смещений и применений гауссовского фильтра. Разрешается использовать библиотеки для работы с изображениями. Подобрать по кросс-валидации с 3 фолдами параметры преобразований. Рассмотреть следующие параметры для преобразований и их комбинации:

(a) Величина поворота: 5, 10, 15 (в каждую из двух сторон)

(b) Величина смещения: 1, 2, 3 пикселя (по каждой из двух размерностей)

(c) Дисперсия фильтра Гаусса: 0.5, 1, 1.5

Проанализировать, как изменилась матрица ошибок, какие ошибки алгоритма помогает исправить каждое преобразование.

Замечание. Не обязательно хранить все обучающие выборки в процессе эксперимента. Достаточно вычислить ближайших соседей для каждой из выборок, а затем выбрать из них ближайших соседей.

6. Реализовать описанный выше алгоритм, основанный на преобразовании объектов тестовой выборки. Проверить то же самое множество параметров, что и в предыдущем пункте. Проанализировать как изменилась матрица ошибок, какие ошибки алгоритма помогает исправить каждое преобразование. Качественно сравнить два подхода (5 и 6 пункты) между собой.

Требования к реализации

Прототипы функций должны строго соответствовать прототипам, описанным в спецификации и проходить все выданные тесты. Задание, не проходящее все выданные тесты, приравнивается к невыполненному. При написании необходимо пользоваться стандартными средствами языка Python, библиотеками numpy и matplotlib. Библиотеками scipy и scikit-learn пользоваться запрещается, если это не обговорено отдельно в пункте задания. Для экспериментов по последним двум пунктам разрешается пользоваться любыми открытыми библиотеками, реализующими алгоритмы обработки изображений.

Замечание 1. Далее под выборкой объектов будем понимать `numpy array` размера $N \times D$ или разреженную матрицу `scipy.sparse.csr_matrix` того же размера, под ответами для объектов выборки будем понимать `numpy array` размера N, где N — количество объектов в выборке, D — размер признакового пространства.

Замечание 2. Для всех функций можно задать аргументы по умолчанию, которые будут удобны вам в вашем эксперименте.

Среди предоставленных файлов должны быть следующие модули и функции в них:

1. Модуль `nearest_neighbors`, содержащий собственную реализацию метода ближайших соседей.

   Класс `KNNClassifier`

   Описание методов:

   (a) `__init__(self, k, strategy, metric, weights, test_block_size)` — конструктор класса.

   - `k` — число ближайших соседей в алгоритме ближайших соседей
   - `strategy` — алгоритм поиска ближайших соседей. Может принимать следующие значения:
     - `'my_own'` — собственная реализация (например, на основе кода подсчёта евклидова расстояния между двумя множествами точек из задания №1)
     - `'brute'` — использование `sklearn.neighbors.NearestNeighbors(algorithm='brute')`
     - `'kd_tree'` — использование `sklearn.neighbors.NearestNeighbors(algorithm='kd_tree')`
     - `'ball_tree'` — использование `sklearn.neighbors.NearestNeighbors(algorithm='ball_tree')`
   - `metric` — название метрики, по которой считается расстояние между объектами. Может принимать следующие значения:
     - `'euclidean'` — евклидова метрика
     - `'cosine'` — косинусная метрика
   - `weights` — переменная типа `bool`. Значение `True` означает, что нужно использовать взвешенный метод k ближайших соседей. Во взвешенном методе ближайших соседей голос одного объекта равен $1/(distance + \varepsilon)$, где $\varepsilon = 10^{-5}$.
   - `test_block_size` — размер блока данных для тестовой выборки

   Замечание 1. Для некоторых алгоритмов поиска ближайших соседей вам потребуется хранить обучающую выборку и ответы на ней. Некоторые алгоритмы не требуют хранения выборки, но требуют хранения дополнительной информации о её структуре.

   Замечание 2. При поиске k ближайших соседей некоторые методы строят в памяти матрицу попарных расстояний обучающей выборки и тестовой выборки. Рекомендуется 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.

(b) `fit(self, X, y)`

Описание параметров:

- `X` — обучающая выборка объектов
- `y` — ответы объектов на обучающей выборке

Метод производит обучение алгоритма с учётом стратегии указанной в параметре `strategy`.

(c) `find_kneighbors(self, X, return_distance)`

Описание параметров:

- `X` — выборка объектов
- `return_distance` — переменная типа `bool`

Метод возвращает `tuple` из двух `numpy array` размера (`X.shape[0], k`). `[i, j]` элемент первого массива должен быть равен расстоянию от `i`-го объекта, до его `j`-го ближайшего соседа. `[i, j]` элемент второго массива должен быть равен индексу `j`-ого ближайшего соседа из обучающей выборки для объекта с индексом `i`.

Если `return_distance=False`, возвращается только второй из указанных массивов. Метод должен использовать стратегию поиска указанную в параметре класса `strategy`.

(d) `predict(self, X)`

Описание параметров:

- `X` — тестовая выборка объектов

Метод должен вернуть одномерный `numpy array` размера `X.shape[0]`, состоящий из предсказаний алгоритма (меток классов) для объектов тестовой выборки.

2. Модуль `cross_validation` с реализациями функций для применения кросс-валидации:

(a) `kfold(n, n_folds)`

Описание параметров:

- `n` — количество объектов в выборке
- `n_folds` — количество фолдов на которые делится выборка

Функция реализует генерацию индексов обучающей и валидационной выборки для кросс-валидации с `n_folds` фолдами. Функция возвращает список длины `n_folds`, каждый элемент списка — кортеж из двух одномерных `numpy array`. Первый массив содержит индексы обучающей подвыборки, а второй валидационной.

(b) `knn_cross_val_score(X, y, k_list, score, cv, **kwargs)`

Описание параметров:

- `X` — обучающая выборка
- `y` — ответы объектов на обучающей выборке
- `k_list` — список из проверяемых значений для числа ближайших соседей, числа в списке упорядочены по возрастанию
- `score` — название метрики, по которой оценивается качество алгоритма. Обязательно должна быть реализована метрика 'accuracy' (доля правильно предсказанных ответов)
- `cv` — список из кортежей, содержащих индексы обучающей и валидационной выборки — выход функций `kfold` или `stratified_kfold`. Если параметр не задан, необходимо внутри функции реализовать генерацию индексов с помощью функции `kfold`
- `**kwargs` — параметры конструктора класса KNNClassifier

Функция для измерения метрики качества `score` алгоритма ближайших соседей, реализованного через класс `KNN_classifier` на кросс-валидации, заданной списком индексов `cv` для обучающей выборки `X`, ответов на ней `y`. Оценку качества метода ближайших соседей нужно рассчитать для нескольких значений k: $[k_1, \ldots, k_n]$, $k_1 < k_2 < \cdots < k_n$, заданных в `k_list`. Сложность алгоритма для одного объекта из валидационной выборки должна иметь порядок $O(k_n)$.

Функция должна возвращать словарь, где ключами являются значения k, а элементами — `numpy array` размера `len(cv)` с качеством на каждом фолде.

Замечание. Для тестирования алгоритма удобно использовать функцию `cross_val_score` из библиотеки `scikit-learn`.

3. Модуль `distances` с реализацией функции для вычисления расстояния:

(a) `euclidean_distance(X, Y)`

Описание параметров:

- `X` — `np.array` размера $N \times D$
- `Y` — `np.array` размера $M \times D$

Функция возвращает `np.array` размера $N \times M$, каждый элемент которого — евклидово расстояние между соответствующей парой векторов из массивов X и Y.

Замечание. Для тестирования алгоритма удобно использовать функцию `pdist` из библиотеки `scipy`.

(b) `cosine_distance(X, Y)`

Описание параметров:

- `X` — `np.array` размера $N \times D$
- `Y` — `np.array` размера $M \times D$

Функция возвращает `np.array` размера $N \times M$, каждый элемент которого — косинусное расстояние между соответствующей парой векторов из массивов X и Y.

Замечание. Для тестирования алгоритма удобно использовать функцию `pdist` из библиотеки `scipy`.

Бонусная часть

1. (до 3 баллов) Написать параллельную реализацию поиска ближайших соседей (например, с помощью библиотеки joblib)

2. (до 5 баллов) Улучшить качество работы метрических алгоритмов на датасете MNIST с помощью средств, не использующихся в задании. Например, можно реализовать ансамбль метрических алгоритмов, реализовать новые метрики, новые признаковые описания объектов. Размер бонуса зависит от величины улучшения и от изобретательности подхода.

3. (до 5 баллов) Качественное проведение дополнительного (не пересекающегося с основным заданием) исследования по теме метрических алгоритмов: формулируется изучаемый вопрос, ставятся эксперименты, позволяющие на него ответить, делаются выводы. Перед исследованием необходимо обсудить тему с преподавателем.