AIMasters

Практическое задание 1
Классификация изображений цифр метрическими методами

курс «Машинное обучение 1», 2022

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 спецификации, описанной в прилагающихся модулях.

   Частично проверить правильность выполнения своих реализаций можно с помощью системы ejudge в соревновании http://45.77.142.102/cgi-bin/new-client?contest_id=5.

   Внимание. Прохождение всех тестов в соревновании не гарантирует правильность решения.

2. Провести описанные ниже эксперименты с датасетом изображений цифр MNIST.

3. Написать отчёт о проделанной работе. Отчёт должен быть подготовлен с помощью jupyter notebook и сдан в форматах .ipynb и .html одновременно.

   Внимание. Отчет подразумевает под собой реализацию экспериментов, замера требуемых характеристик (например, скорости работы и/или качества работы алгорима), представленных в формате таблиц и/или графиков, и комментариев к экспериментам.

   Замечание. Чтобы экспорировать jupyter notebook в .html нужно выбрать:
   `File -> Download as -> HTML (.html)`.

   Для экспорта notebook в .html в Google Colab, воспользуйтесь следующим кодом.

   Большая просьба: подписывайте свой отчет (в названии файла и внутри ноутбука).

4. В систему проверки необходимо сдать отчёт в обоих форматах и .zip архив с заполненными модулями.

   Большая просьба: jupyter notebook и html файл не запаковывать в архив, а сдавать отдельно.

Реализация алгоритмов (10 баллов)

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 и библиотекой numpy. Библиотеками scipy и scikit-learn пользоваться **запрещено**, если это не обговорено отдельно в пункте задания. Для экспериментов в бонусной части разрешается пользоваться любыми открытыми библиотеками, реализующими алгоритмы обработки изображений.

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

1. Модуль `knn.distances` с реализацией функции для вычисления расстояния:
   (a) `euclidean_distance(X, Y)` — реализация евклидова расстояния с заданными свойствами;
   (b) `cosine_distance(X, Y)` — реализация косинусного расстояния с заданными свойствами;

2. Модуль `knn.nearest_neighbors`, содержащий собственную реализацию поиска ближайших соседей.

   Класс `NearestNeighborsFinder` с методами:

   (a) `__init__(self, n_neighbors, metric="euclidean")` — конструктор (инициализатор) класса;
   (b) `fit(self, X, y=None)` — обучение алгоритма;
   (c) `kneighbors(self, X, return_distance=False)` — поиск ближайших соседей.

3. Модуль `knn.classification`, содержащий собственную реализацию классификатора на основе метода ближайших соседей.

   Класс `KNNClassifier` с методами:

   (a) `__init__(self, n_neighbors, algorithm='my_own', metric='euclidean', weights='uniform')` — конструктор (инициализатор) класса;

   (b) `fit(self, X, y=None)` — обучение алгоритма;

   (c) `kneighbors(self, X, return_distance=False)` — поиск ближайших соседей;

   (d) `predict(self, X)` — редсказание на новых данных;

   (e) `_predict_precomputed(self, indices, distances)` — вспомогательный метод.

   Вспомогательные классы `BatchedMixin` и `BatchedKNNClassifier`. Данные классы могут быть полезны, если при работе алгоритма он полностью не помещается в оперативную память.

4. Модуль `knn.model_selection` с реализациями функций для применения кросс-валидации:

   (a) `knn_cross_val_score(X, y, k_list, scoring, cv=None, **kwargs)` — функция для измерения качества на кросс-валидации.

Ожидается, что реализациия всех классов и функций будет максимально эффективной. Дополнительно вам предоставлены открытые unit-тесты, которые находятся рядом с модулем `knn` в директории `tests`. Чтобы запустить тесты в консоли требуется выполнить одну из команд:

```
$ python -m unittest                            # запуск всех тестов
$ python -m unittest tests/test_distances.py    # запуск конкретных тестов
```

Эксперименты (15 баллов)

Эксперименты этого задания необходимо проводить на датасете MNIST. Загрузить датасет можно при помощи функции `sklearn.datasets.fetch_openml("mnist_784")` или скачать вручную с сайта http://yann.lecun.com/exdb/mnist/. 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.

   Для подсчёта точности можно воспользоваться функцией `accuracy_score` из библиотеки scikit-learn.

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 метрических 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 1. 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")`.

Замечание 2. Не нужно копировать свой код из модулей в jupyter notebook, пользуйтесь им, как если бы это была библиотека. Для этого поместите директорию knn рядом с notebook-ом. Пример, как может выглядеть содержимое вашей рабочей директории:

```
$ tree .
.
├── knn
│   ├── __init__.py
│   ├── classification.py
│   ├── distances.py
│   ├── model_selection.py
│   └── nearest_neighbors.py
├── tests
│   ├── __init__.py
│   ├── test_classification.py
│   ├── test_distances.py
│   ├── test_model_selection.py
│   └── test_nearest_neigbours.py
└── experiments.ipynb
```

Тогда в jupyter notebook `experiments.ipynb` нужные классы и функции можно импортировать следующим образом:

```
from knn import KNNClassifier
from knn.distances import euclidean_distance, cosine_distance
from knn.model_selection import knn_cross_val_score
```

Для того, чтобы не перезагружать jupyter notebook каждый раз после того, как вы внесли изменения в модуль `knn`, можно добавить ячейку с таким содержимым:

```
%load_ext autoreload
%autoreload 2
```

Бонусная часть (до 5 баллов)

5. (до 3 баллов) Размножить обучающую выборку с помощью поворотов, смещений, применений гауссовского фильтра и морфологических операций. 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

   (d) Морфологические операции: эрозия, дилатация, открытие, закрытие с ядром 2

   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 по каждой из выборок, а затем выбрать из них ближайших соседей.

6. (до 2 баллов) Реализовать предложенный на семинаре 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. Сравнить два подхода (5 и 6 пункты) между собой.