

Московский авиационный институт  
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу  
«Искусственный интеллект»

Студент: К. В. Лукашкин  
Группа: М8О-308Б

Москва, 2019

## Постановка задачи

Требуется реализовать класс на выбранном языке программирования, который реализует один из алгоритмов машинного обучения. Обязательным является наличие в классе двух методов `fit`, `predict`. Необходимо проверить работу вашего алгоритма на ваших данных (на таблице и на текстовых данных), произведя необходимую подготовку данных. Также необходимо реализовать алгоритм полиномиальной регрессии, для предсказания значений в таблице. Сравнить результаты с стандартной реализацией `sklearn`, определить в чем сходство и различие ваших алгоритмов. Замерить время работы алгоритмов.

Мой вариант:  $7\%6+1 = 2$  KNN (K ближайших соседей)

## Решение

Мной был реализован алгоритм K ближайших соседей для задачи классификации данных и алгоритм полиномиальной регрессии.

К сожалению, к выбранному мной датасету в нулевой лабораторной не представляется возможным применить задачу классификации – много авторов и у каждого мало статей, много издателей и у них публикуется мало статей, теги не проставлены. Для обучения классификатора необходимо иметь набор объектов, для которых заранее определены классы, вручную определять эти классы для изначального датасета не представляется возможным. Поэтому я решил воспользоваться датасетом из вопросов в StackOverflow из примеров tensorflow <https://storage.googleapis.com/tensorflow-workshop-examples/stack-overflow-data.csv>

Для обработки текста я воспользовался препроцессором текста в Azure ML Studio. Непосредственно в модель же требуется массив данных, для этого я составил разреженную матрицу с частотой слов в текстах с помощью инструментов `sklearn` – `CountVectorizer` и `TfidfTransformer`.

Стоит отметить что заданный алгоритм является не лучшим выбором при работе с текстовыми данными, вследствие их большой размерности.

## KNN (K ближайших соседей)

KNN – классификатор. Объект относится к тому классу, которому принадлежит большинство из его  $k$  соседей – ближайших к нему элементов из данных обучения. Также KNN можно использовать для задач регрессии, в таком случае значение для элемента выбирается как среднее ближайших соседей. Кроме алгоритма полного перебора для нахождения расстояния до соседей также существуют `BallTree` и `KDTree`. В задачу есть возможность ввести веса – в таком случае чем дальше сосед от объекта тем меньше он влияет на класс, или же выбрать другую метрику – если какой-нибудь параметр вносит большой вклад в выбор класса (вариант собственной метрики в моей реализации предусмотрен).

Алгоритм считается одним из самых простых и благодаря этому является хорошим примером для исследования.

Алгоритм предсказания класса точки:

```

def predict(self, X, custom_dist=None):
    # сопоставляет данным -- классы
    num_of_classes = len(self.classes)
    train_data = self.train_data
    test_data = X

    # Евклидово расстояние между двумя точками, есть возможность
    задавать своё
    def dist(a, b):
        assert a.shape == b.shape # ошибка если размерности не
        совпадают
        n = a.shape[0]
        return sum(
            (a[i] - b[i]) ** 2 for i in range(n))

    if custom_dist:
        dist = custom_dist

    test_labels = []
    for test_point in test_data:
        # print(len(test_data));start_time = time.time()
        # подсчитываем расстояния между заданной точкой и ВСЕМИ
        точками из TrainData
        test_dist = [[dist(test_point, train_data[i][0]),
            train_data[i][1]]
            for i in range(len(train_data))]

        # подсчитываем количество точек каждого класса для k
        ближайших соседей
        stat = [0 for i in range(num_of_classes)]
        for d in sorted(test_dist)[0:self.k]:
            stat[d[1]] += 1

        # выбираем класс, который встречается чаще всего у соседей
        # np.max
        selected_class = sorted(zip(stat, range(num_of_classes)),
            reverse=True)[0][1]
        # сортировка по количеству точек с
        конкретным классом по убыванию
        test_labels.append(selected_class)
        # print("Затрачено времени на точку", (time.time() -
        start_time))
    return test_labels

```

Моя реализация сравнивается с *sklearn.neighbors.KneighborsClassifier*.

Запуск для табличных данных:

```
konstanze@G5-5587: ~/Desktop/ai/ml2
File Edit View Search Terminal Help
konstanze@G5-5587:~$ cd Desktop/ai/ml2/
konstanze@G5-5587:~/Desktop/ai/ml2$ python3 main.py
Акции
sklearn
precision: 0.5213988750305698
время работы 0.20234060287475586
Собственная реализация
precision: 0.5213988750305698
время работы 23.635408401489258
konstanze@G5-5587:~/Desktop/ai/ml2$
```

Можно заметить что и моя реализация и реализация в sklearn дают одинаково плохой результат. Это связано в первую очередь с тем, что алгоритм не совсем подходит для таких данных. Возможно здесь стоит каким-либо другим образом поставить задачу классификации, но я не смог до него дойти.

В данном случае используется параметр 15, т. е. Выбор класса производится по 15 ближайшим соседям, но точность для 5 и для 21 соседа приблизительно такая же.

Стоит отметить, что алгоритмы дают одинаковую точность, хоть мой и значительно медленнее, это показывает то, что алгоритм был реализован абсолютно точно. При это стоит отметить, что я запускаю алгоритм sklearn с параметром `algorithm='brute'`, т. е. Вычисление расстояний до соседей происходит прямым перебором.

Запуск для текстовых данных

```
konstanze@G5-5587: ~/Desktop/ai/ml2
File Edit View Search Terminal Help
konstanze@G5-5587:~$ cd Desktop/ai/ml2/
konstanze@G5-5587:~/Desktop/ai/ml2$ python3 main.py
Текстовые данные
sklearn
precision: 0.8793969849246231
время работы 0.09357523918151855
Собственная реализация
precision: 0.8793969849246231
время работы 1956.4692914485931
konstanze@G5-5587:~/Desktop/ai/ml2$
```

Запуск моей реализации для данных такой большой размерности занял очень долгое время. Для того чтобы оно было приемлемым я определял классы только для данных принадлежащих классам «с++» и «python» из всего датасета.

Хотя точность результатов и получилась высокой, однако, я ожидал ещё более точных результатов – поскольку в тексте должны упоминаться специфичные для тега слова, причём часто.

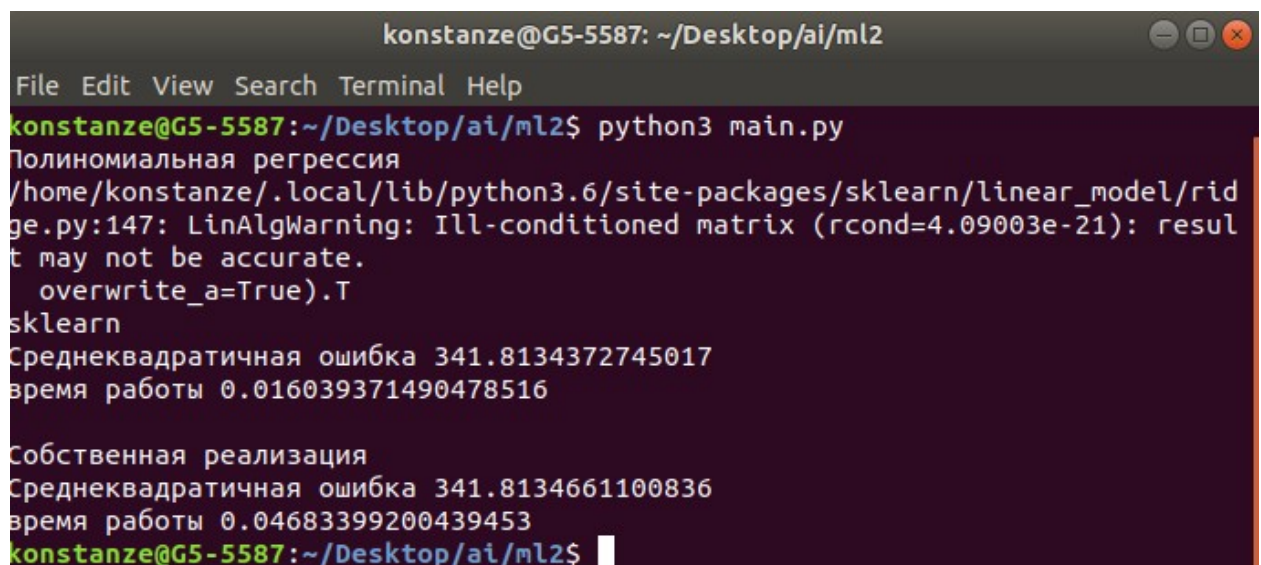
Думаю, что результат можно улучшить для KNN, если ввести веса например, упоминание слова *Java* у соседей, скорее всего является признаком того, что в данном тексте речь ведётся именно о ней.

## Полиномиальная регрессия

В алгоритме полиномиальной регрессии как и в линейной строится многочлен по входным параметрам, однако порядок многочлена может быть выше единицы. При обучении ищутся такие коэффициенты, при которых результат вычисления многочлена был как можно ближе к заданному значению.

Для этого используется решение СЛАУ методом наименьших квадратов. При этом для решения я пользуюсь `sklearn.preprocessing.PolynomialFeatures` (матрица со значениями и их степенями) и `np.linalg.lstsq` (решение системы)

Запуск программы:



```
konstanze@G5-5587: ~/Desktop/ai/ml2
File Edit View Search Terminal Help
konstanze@G5-5587:~/Desktop/ai/ml2$ python3 main.py
Полиномиальная регрессия
/home/konstanze/.local/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=4.09003e-21): result may not be accurate.
  overwrite_a=True).T
sklearn
Среднеквадратичная ошибка 341.8134372745017
время работы 0.016039371490478516

Собственная реализация
Среднеквадратичная ошибка 341.8134661100836
время работы 0.04683399200439453
konstanze@G5-5587:~/Desktop/ai/ml2$
```

При повышении степени полинома среднеквадратичная ошибка сильно возрастает у обеих реализаций, это можно связать с тем, что цены акций за длительный период сложно представить полиномом таких степеней. Чем выше степень тем выше и значение ошибки, полином по полученным коэффициентам отклоняется от тестовых точек.

Весь код на github: [https://github.com/memosiki/mai\\_ai/tree/master/ml2](https://github.com/memosiki/mai_ai/tree/master/ml2)

## Выводы.

Выполнив лабораторную работу, я ознакомился с библиотекой `sklearn`, изучил и разработал два алгоритма машинного обучения – KNN и полиномиальной регрессии, сравнил их работу с библиотечными функциями `sklearn`