# Метод главных компонент

В данной работе необходимо реализовать метод главных компонент, позволяющий выделять наиболее существенную информацию из данных с помощью линейных преобразований и с помощью него и алгоритма k-ближайших соседей решить задачу распознавания рукописных цифр из базы данных MNIST.

Основная часть работы заключается в реализации собственных классов my\_PCA, my\_kNN, аналогичных имеющимся в библиотеке sklearn:

sklearn.decomposition.**PCA**

sklearn.neighbors.**KNeighborsClassifier**

Можете использовать следующие шаблоны классов :

**class** my\_PCA():  
 **def** \_\_init\_\_(self, n\_components=**None**):  
 **pass  
   
 def** fit(self, X):  
 **pass  
   
 def** transform(self, X):  
 **pass  
   
 def** fit\_transform(self, X):  
 **pass**

**class** my\_kNN():  
 **def** \_\_init\_\_(self, n\_neighbors=5):  
 **pass  
  
 def** fit(self, X, y):  
 **pass  
  
 def** predict(self, X):  
 **pass**

Загрузите датасет MNIST рукописных цифр в виде картинок размера 28\*28 пикселей.

В работе вам понадобятся следующие вспомогательные библиотеки

sklearn.model\_selection.**train\_test\_split** – для разбиения датасета на обучающую и тестовую выборку.

sklearn.metrics.**accuracy\_score** – для оценки точности алгоритмов.

**I. Датасет:**

1. Нужно скачать базу данных MNIST при помощи функции load\_mnistиз пакета mnist.py, используя код ниже:

from mnist import load\_mnist

train, validation, test = load\_mnist()

Получить массивы картинок в виде массива X, а также ответов labels.

2. При помощи функции matplotlib.pyplot.imshow нарисовать несколько примеров картинок из X. Чтобы картинки шли в виде массива, а не друг под другом, используйте функцию subplots.

**II. Алгоритм PCA:**

Метод главных компонент состоит из следующих 4-х шагов:

1) Центрирование данных: , где  – среднее для каждого параметра

2) Вычисление матрицы ковариации: 

3) Вычисление собственных векторов *F* и значений  матрицы ковариации .

4) Преобразование данных в координаты в базисе главных компонент: 

1. Реализуйте описанные выше шаги 1-3 внутри метода fit(), и шаг 4 внутри метода transform() класса my\_PCA. При реализации шага 3 используйте функцию numpy.linalg.eig, либо реализуйте вычисление матрицы *F* и значений  методом сингулярного разложения (singular value decomposition (SVD)) матрицы  без вычисления матрицы ковариации , для чего используйте функцию numpy.linalg.svd.

2. Заметьте, что собственные значения уже упорядочены в порядке убывания. Постройте график собственных значений, а также график отношения кумулятивной суммы к их полной сумме. Посмотрите, какую долю дисперсии данных покрывают первые 15 главных компонент? Как связаны между собой собственные числа и дисперсия данных?

3. Изобразите на графике точки датасета в первых двух координатах главных компонент. Разным цифрам должны соответствовать разные цвета. Сделайте выводы о линейной разделимости классов в этих координатах.

**III. Алгоритм kNN:**

Алгоритм k-ближайших соседей (k-nearest neighbours (kNN)) является одним из простейших метрических алгоритмов для решения задач классификации объектов. Суть его в следующем. Пусть у нас есть некоторая обучающая выборка данных  с известными классами принадлежности объектов . Предположим, есть некоторый объект  с неизвестным классом, который мы хотели бы предсказать. Для этого посчитаем расстояния от  до каждого из объектов  и найдём *k* ближайших (т.е. с наименьшим расстоянием) из этого набора. Поскольку мы знаем к какому классу принадлежат эти *k* соседей, то мы можем предположить, что и наш объект  будет принадлежать к тому классу, из которого наибольшее количество соседей. Например, пусть  и среди этих семи ближайших соседей *четыре соседа оказалось в классе A*, *один сосед оказался в классе B* и ещё *два соседа оказалось в классе C*, значит мы делаем предположение, что и наш *объект  принадлежит классу A*.

1. Реализуйте метод k-ближайших соседей в виде класса my\_kNN. Заметьте, что метод \_\_init\_\_ ничего не делает, кроме сохранения параметра n\_neighbors во внутреннюю переменную self.n\_neighbors. Аналогично метод fit лишь сохраняет переданные обучающие данные во внутренние переменные. Основные вычисления происходят лишь в методе predict, который должен возвращать предполагаемый класс.

2. Разбейте данные X и labels на обучающую и тестовую выборку, используя функцию train\_test\_splitиз модуля sklearn.model\_selection.

3. Создайте классификатор my\_kNN с числом соседей равным 5. Обучите классификатор на обучающих данных. Посчитайте точность на тестовой выборке, используя функцию accuracy\_scoreиз модуляsklearn.metrics. Какова точность полученного алгоритма? Если точность оказалась около 10%, значит ваш алгоритм работает как случайный, а это значит, что ваш код работает неправильно и его нужно исправить.

4. Уменьшите размерность данных с помощью реализованного выше метода главных компонент. Примените алгоритм k-ближайших соседей к преобразованным данным. Попробуйте разное количество соседей (от 1 до 30) и разное количество компонент (начиная с одной и заканчивая всеми 64). Найдите параметры, при которых алгоритм даёт наибольшую точность на тестовой выборке.

Сделайте выводы по всей работе.

Решение в формате Jupyter notebook.

Для единообразия название файла пусть будет:

***Lab6.Фамилия.Имя.Номер\_группы.ipynb***