В этом задании будет использоваться датасет digits из sklearn.datasets. Оставьте последние 25% объектов для контроля качества, разделив X и y на X\_train, y\_train и X\_test, y\_test.

Целью задания будет реализовать самый простой метрический классификатор — метод ближайшего соседа, а также сравнить качество работы реализованного вами 1NN с RandomForestClassifier из sklearn на 1000 деревьях.

**Задание 1**

Реализуйте самостоятельно метод одного ближайшего соседа с евклидовой метрикой для задачи классификации. Можно не извлекать корень из суммы квадратов отклонений, т.к. корень — монотонное преобразование и не влияет на результат работы алгоритма.

Никакой дополнительной работы с признаками в этом задании делать не нужно — мы еще успеем этим заняться в других курсах. Ваша реализация может быть устроена следующим образом: можно для каждого классифицируемого объекта составлять список пар (расстояние до точки из обучающей выборки, метка класса в этой точке), затем сортировать этот список (по умолчанию сортировка будет сначала по первому элементу пары, затем по второму), а затем брать первый элемент (с наименьшим расстоянием).

Сортировка массива длиной N требует порядка N log N сравнений (строже говоря, она работает за O(N log N)). Подумайте, как можно легко улучшить получившееся время работы. Кроме простого способа найти ближайший объект всего за N сравнений, можно попробовать придумать, как разбить пространство признаков на части и сделать структуру данных, которая позволит быстро искать соседей каждой точки. За выбор метода поиска ближайших соседей в KNeighborsClassifier из sklearn отвечает параметр algorithm — если у вас уже есть некоторый бэкграунд в алгоритмах и структурах данных, вам может быть интересно познакомиться со структурами данных ball tree и kd tree.

Доля ошибок, допускаемых 1NN на тестовой выборке, — ответ в задании 1.

**Задание 2**

Теперь обучите на обучающей выборке RandomForestClassifier(n\_estimators=1000) из sklearn. Сделайте прогнозы на тестовой выборке и оцените долю ошибок классификации на ней. Эта доля — ответ в задании 2. Обратите внимание на то, как соотносится качество работы случайного леса с качеством работы, пожалуй, одного из самых простых методов — 1NN. Такое различие — особенность данного датасета, но нужно всегда помнить, что такая ситуация тоже может иметь место, и не забывать про простые методы.

Пример:

from sklearn.datasets import load\_digits

data = load\_digits()

# может быть по-другому надо округлять?

test\_size = int(len(data.target) \* 0.25)

X\_train, X\_test, y\_train, y\_test = (

data.data[:-test\_size], data.data[-test\_size:],

data.target[:-test\_size], data.target[-test\_size:])

from sklearn.ensemble import RandomForestClassifier

# может быть другой random\_state

rf = RandomForestClassifier(n\_estimators=1000, random\_state=0)

rf.fit(X\_train, y\_train)

# доля ошибок

from sklearn.metrics import accuracy\_score

1-accuracy\_score(y\_test, rf.predict(X\_test))

# loading libraries

import numpy as np

from sklearn.cross\_validation import train\_test\_split

# create design matrix X and target vector y

X = np.array(df.ix[:, 0:4]) # end index is exclusive

y = np.array(df['class']) # another way of indexing a pandas df

# split into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

# loading library

from sklearn.neighbors import KNeighborsClassifier

# instantiate learning model (k = 3)

knn = KNeighborsClassifier(n\_neighbors=3)

# **algorithm** : {‘auto’, ‘ball\_tree’, ‘kd\_tree’, ‘brute’}

# fitting the model

knn.fit(X\_train, y\_train)

# predict the response

pred = knn.predict(X\_test)

# evaluate accuracy

print accuracy\_score(y\_test, pred)

def predict(X\_train, y\_train, x\_test, k):

# create list for distances and targets

distances = []

targets = []

for i in range(len(X\_train)):

# first we compute the euclidean distance

distance = np.sqrt(np.sum(np.square(x\_test - X\_train[i, :])))

# add it to list of distances

distances.append([distance, i])

# sort the list

distances = sorted(distances)

# make a list of the k neighbors' targets

for i in range(k):

index = distances[i][1]

targets.append(y\_train[index])

# return most common target

return Counter(targets).most\_common(1)[0][0]

def kNearestNeighbor(X\_train, y\_train, X\_test, predictions, k):

# train on the input data

train(X\_train, y\_train)

# loop over all observations

for i in range(len(X\_test)):

predictions.append(predict(X\_train, y\_train, X\_test[i, :], k))

# making our predictions

predictions = []

kNearestNeighbor(X\_train, y\_train, X\_test, predictions, 7)

# transform the list into an array

predictions = np.asarray(predictions)

# evaluating accuracy

accuracy = accuracy\_score(y\_test, predictions)

print('\nThe accuracy of our classifier is %d%%' % accuracy\*100)