Общая информация

Срок сдачи: 5 апреля 2022, 08:30

Штраф за опоздание: по 1 баллу за 24 часа задержки. Через 5 дней домашнее задание сгорает.

При отправлении ДЗ указывайте фамилию в названии файла Присылать ДЗ необходимо в виде ссылки на свой github репозиторий на почту ml1.sphere@mail.ru с указанием темы в следующем формате:

[ML0220, Задание 1] Фамилия Имя.

Используйте данный Ipython Notebook при оформлении домашнего задания.

Штрафные баллы:

1. Отсутствие фамилии в имени скрипта (скрипт должен называться по аналогии со stroykova_hw1.ipynb) -0.5 баллов
2. Все строчки должны быть выполнены. Нужно, чтобы output команды можно было увидеть уже в git'е. В противном случае -0.5 баллов

In [1]:
```
from scipy import stats

import numpy as np
import matplotlib.pyplot as plt

from sklearn import datasets
from sklearn.base import BaseEstimator
from sklearn.datasets import fetch_20newsgroups

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.neighbors import KDTree
```

Задание 1 (1 балл)

Реализовать KNN в классе MyKNeighborsClassifier (обязательное условие: точность не ниже sklearn реализации) Разберитесь самостоятельно, какая мера расстояния используется в KNeighborsClassifier дефолтно и реализуйте свой алгоритм именно с этой мерой. Для подсчета расстояний можно использовать функции отсюда

In [2]:
```
class MyKNeighborsClassifier(BaseEstimator):

    def euclidean(self, X_train, X_test):
        X_train = X_train[:,:, np.newaxis]
        dist = (X_train - X_test.T) ** 2
        dist = np.sqrt(np.sum(dist, axis=1))
        return dist

    def brute(self, X):
        distances = self.euclidean(self.X_train, X)
        distances = distances.argpartition(self.K, axis=0)[:self.K, :]
        values = np.take(self.y_train, distances.T)
        return stats.mode(values, axis=1)[0].reshape(-1)

    def kd_tree(self, X):
        nearest_x_ind = self.tree.query(X, k=self.K, return_distance=False)
        values = np.take(self.y_train, nearest_x_ind)
        return stats.mode(values, axis=1)[0].reshape(-1)

    def __init__(
        self,
        n_neighbors,
        algorithm='brute',
        leaf_size = 40,
    ):
        self.K = n_neighbors
        self.algorithm_name = algorithm

        if algorithm == 'brute':
            self.algorithm = self.brute

        elif algorithm == 'kd_tree':
            self.algorithm = self.kd_tree
            self.leaf_size = leaf_size

        else:
            assert 0, 'Wrong algorithm\'s name'

    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)
        if self.algorithm_name == 'kd_tree':
            self.tree = KDTree(X, leaf_size=self.leaf_size)

    def predict(self, X):
        return self.algorithm(X)
```

IRIS

В библиотеке scikit-learn есть несколько датасетов из коробки. Один из них Ирисы Фишера

In [3]:
```
iris = datasets.load_iris()
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.1, stratify=iris.target)
```

In [5]:
```
clf = KNeighborsClassifier(n_neighbors=2, algorithm='brute')
my_clf = MyKNeighborsClassifier(n_neighbors=2, algorithm='brute')
```

In [6]:
```
clf.fit(X_train, y_train)
my_clf.fit(X_train, y_train)
```

In [7]:
```
sklearn_pred = clf.predict(X_test)
my_clf_pred = my_clf.predict(X_test)
assert abs( accuracy_score(y_test, my_clf_pred) -  accuracy_score(y_test, sklearn_pred ) )<0.005, "Score must be simillar"
```

Задание 2 (0.5 балла)

Давайте попробуем добиться скорости работы на fit, predict сравнимой со sklearn для iris. Допускается замедление не более чем в 2 раза. Для этого используем numpy.

In [8]:
```
%time clf.fit(X_train, y_train)
```
```
CPU times: user 2.01 ms, sys: 187 µs, total: 2.2 ms
Wall time: 7.62 ms
```
Out[8]:
```
KNeighborsClassifier(algorithm='brute', n_neighbors=2)
```

In [9]:
```
%time my_clf.fit(X_train, y_train)
```
```
CPU times: user 435 µs, sys: 40 µs, total: 475 µs
Wall time: 2.26 ms
```

In [10]:
```
%time clf.predict(X_test)
```
```
CPU times: user 3.57 ms, sys: 0 ns, total: 3.57 ms
Wall time: 13.6 ms
```
Out[10]:
```
array([2, 2, 1, 0, 0, 0, 1, 2, 2, 0, 2, 2, 2, 0, 1])
```

In [11]:
```
%time my_clf.predict(X_test)
```
```
CPU times: user 1.57 ms, sys: 0 ns, total: 1.57 ms
Wall time: 1.58 ms
```
Out[11]:
```
array([2, 2, 1, 0, 0, 0, 1, 2, 2, 0, 2, 2, 2, 0, 1])
```

Задание 3 (1 балл)

Добавьте algorithm='kd_tree' в реализацию KNN (использовать KDTree из sklearn.neighbors). Необходимо добиться скорости работы на fit, predict сравнимой со sklearn для iris. Допускается замедление не более чем в 2 раза. Для этого используем numpy. Точность не должна уступать значению KNN из sklearn.

In [12]:
```
clf = KNeighborsClassifier(n_neighbors=2, algorithm='kd_tree')
my_clf = MyKNeighborsClassifier(n_neighbors=2, algorithm='kd_tree')
```

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.1, stratify=iris.target)
```

In [14]:
```
%time clf.fit(X_train, y_train)
```
```
CPU times: user 0 ns, sys: 1.88 ms, total: 1.88 ms
Wall time: 9.67 ms
```
Out[14]:
```
KNeighborsClassifier(algorithm='kd_tree', n_neighbors=2)
```

In [15]:
```
%time my_clf.fit(X_train, y_train)
```
```
CPU times: user 879 µs, sys: 85 µs, total: 964 µs
Wall time: 2.67 ms
```

In [16]:
```
%time clf.predict(X_test)
```
```
CPU times: user 3.63 ms, sys: 353 µs, total: 3.98 ms
Wall time: 27.4 ms
```
Out[16]:
```
array([2, 1, 1, 0, 1, 0, 1, 2, 2, 2, 0, 2, 0, 0, 1])
```

In [17]:
```
%time my_clf.predict(X_test)
```
```
CPU times: user 3.16 ms, sys: 0 ns, total: 3.16 ms
Wall time: 5.32 ms
```
Out[17]:
```
array([2, 1, 1, 0, 1, 0, 1, 2, 2, 2, 0, 2, 0, 0, 1])
```

In [18]:
```
sklearn_pred = clf.predict(X_test)
my_clf_pred = my_clf.predict(X_test)
assert abs( accuracy_score(y_test, my_clf_pred) -  accuracy_score(y_test, sklearn_pred ) )<0.005, "Score must be simillar"
```

Задание 4 (2.5 балла)

Рассмотрим новый датасет 20 newsgroups

In [19]:
```
newsgroups = fetch_20newsgroups(subset='train',remove=['headers','footers', 'quotes'])
```

In [20]:
```
train_size = len(newsgroups['data'])
data = newsgroups['data']
target = newsgroups['target']
```

Преобразуйте текстовые данные из data с помощью CountVectorizer. Словарь можно ограничить по частотности.

In [21]:
```
max_features = 100
vectorizer = CountVectorizer(max_features=max_features, stop_words='english', max_df=0.06)
X_train = vectorizer.fit_transform(data).toarray()
y_train = np.asarray(target)
```

Так мы получили векторное представление наших текстов. Значит можно приступать к задаче обучения модели

Реализуйте разбиение выборки для кросс-валидации на 3 фолдах. Разрешено использовать sklearn.cross_validation

In [22]:
```
folds = 3

#тестовые фолды
split_X = np.array_split(X_train, folds)
split_y = np.array_split(y_train, folds)

#трэйновые фолды
Xtrain = [np.concatenate([split_X[j] for j in range(folds) if j != i]) for i in range(folds)]
ytrain = [np.concatenate([split_y[j] for j in range(folds) if j != i]) for i in range(folds)]
```

Напишите метод, позволяющий найти оптимальное количество ближайших соседей(дающее максимальную точность в среднем на валидации на 3 фолдах). Постройте график зависимости средней точности от количества соседей. Можно рассмотреть число соседей от 1 до 10. (было рассмотрено от 1 до 100, т. к. при числе соседей до 10 точность сильно меньше, чем при больших значениях.)

In [23]:
```
K_min = 1 #Минимальное число соседей, которое будет проверяться
K = 100 #Сколько вариантов будет проверяться
K_step = 1 #Шаг, с которым будет проверяться число соседей
k_range = range(K_min, K_min+K*K_step, K_step) #Числа соседей, которые будут проверяться
def optimal(split_X, split_y, folds):
    scores = np.zeros(K)
    for k in k_range:
        print(k, end=' ') #печатает, на каком этапе выполнение функции
        my_clf = MyKNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
        score = 0
        for i in range(folds):
            my_clf.fit(Xtrain[i], ytrain[i])
            my_clf_pred = my_clf.predict(split_X[i])
            score += accuracy_score(split_y[i], my_clf_pred)
        scores[(k-K_min) // K_step] = score / folds
    print("")
    return scores
```

Метрика евклидова, векторизация с помощью CountVectorizer:

In [24]:
```
accuracy = optimal(split_X, split_y, folds)
```
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

In [25]:
```
plt.plot(k_range, accuracy)
print("Лучшая точность: ", accuracy.max(), "\nЧисло соседей: ", accuracy.argmax()*K_step+K_min)
```
```
Лучшая точность:  0.24845376320319187
Число соседей:  46
```

Как изменится качество на валидации, если:

1. Используется косинусная метрика вместо евклидовой.
2. К текстам применяется TfIdf векторизацию(sklearn.feature_extraction.text.TfidfVectorizer)

Сравните модели, выберите лучшую.

In [26]:
```
#Далее все рассчёты быдут проводиться только для 46 соседей
K_min = 46
K = 1
K_step = 1
k_range = range(K_min, K_min+K*K_step, K_step)
```

Метрика косинусная, векторизация с помощью CountVectorizer:

In [27]:
```
X_train = X_train / (np.sqrt((X_train ** 2).sum(axis=1))[:, np.newaxis] + 0.00001) #нормировка векторов

split_X = np.array_split(X_train, folds)
Xtrain = [np.concatenate([split_X[j] for j in range(folds) if j != i]) for i in range(folds)]

accuracy_cos = optimal(split_X, split_y, folds)
```
```
46
```

In [28]:
```
#plt.plot(k_range, accuracy_cos)
print("Точность: ", accuracy_cos.max(), "Число соседей: ", accuracy_cos.argmax()*K_step+K_min)
```
```
Точность:  0.2605618973245524 Число соседей:  46
```

Метрика евклидова, векторизация с помощью TfidfVectorizer:

In [29]:
```
tfidf_vectorizer = TfidfVectorizer(max_features=max_features, stop_words='english', max_df=0.06)
```

In [30]:
```
X_train = tfidf_vectorizer.fit_transform(data).toarray()
split_X = np.array_split(X_train, folds)

accuracy_tfidf = optimal(split_X, split_y, folds)
```
```
46
```

In [31]:
```
#plt.plot(k_range, accuracy_tfidf)
print("Точность: ", accuracy_tfidf.max(), "Число соседей: ", accuracy_tfidf.argmax()*K_step+K_min)
```
```
Точность:  0.2644511344459714 Число соседей:  46
```

Метрика косинусная, векторизация с помощью TfidfVectorizer:

In [32]:
```
X_train = X_train / (np.sqrt((X_train ** 2).sum(axis=1))[:, np.newaxis] + 0.00001) #нормировка векторов

split_X = np.array_split(X_train, folds)
Xtrain = [np.concatenate([split_X[j] for j in range(folds) if j != i]) for i in range(folds)]

accuracy_tfidf_cos = optimal(split_X, split_y, folds)
```
```
46
```

In [33]:
```
#plt.plot(k_range, accuracy_tfidf_cos)
print("Точность: ", accuracy_tfidf_cos.max(), "Число соседей: ", accuracy_tfidf_cos.argmax()*K_step+K_min)
```
```
Точность:  0.26445101727486436 Число соседей:  46
```

Итог: Лучше всего работает TfIdf векторизация, метрика особо ни на что не влияет, формально точнее была евклидова метрика. Число соседей 46.

Загрузим теперь test часть нашей выборки и преобразуем её аналогично с train частью. Не забудьте, что наборы слов в train и test части могут отличаться.

In [34]:
```
newsgroups = fetch_20newsgroups(subset='test',remove=['headers','footers', 'quotes'])
```

Оценим точность вашей лучшей модели на test части датасета. Отличается ли оно от кросс-валидации? Попробуйте сделать выводы, почему отличается качество.

In [35]:
```
test_size = len(newsgroups['data'])
data_test = newsgroups['data']
target_test = newsgroups['target']

#объединение словарей
data_all = data + data_test
target_all = np.concatenate((target, target_test))

X = tfidf_vectorizer.fit_transform(data_all).toarray()
X_test = X[train_size:train_size+test_size, :]
#X_test = X_test / (np.sqrt((X_test ** 2).sum(axis=1))[:, np.newaxis] + 0.00001)
y_test = np.asarray(target_test)
X_train = X[:train_size, :]
y_train = target[:train_size]
#X_train = X_train / (np.sqrt((X_train ** 2).sum(axis=1))[:, np.newaxis] + 0.00001)

my_clf = MyKNeighborsClassifier(n_neighbors=46, algorithm='kd_tree')
my_clf.fit(X_train, y_train)
my_clf_pred = my_clf.predict(X_test)
print("Точность на тестовой выборке: ", accuracy_score(y_test, my_clf_pred))
```
```
Точность на тестовой выборке:  0.24362719065321295
```

Точность чуть меньше (была 0.265). Но, во-первых, на разных выборка всегда будут разные результаты, во-вторых, при добавлении тестовой выборки был изменён словарь, а при достаточно скудной возможности выбора максимального числа слов, это тоже влияет на погрешность измерений.

In [36]:
```
scores = np.zeros(folds)

split_X = np.array_split(X_train, folds)
split_y = np.array_split(y_train, folds)

#трэйновые фолды
Xtrain = [np.concatenate([split_X[j] for j in range(folds) if j != i]) for i in range(folds)]
ytrain = [np.concatenate([split_y[j] for j in range(folds) if j != i]) for i in range(folds)]
for i in range(folds):
    my_clf.fit(Xtrain[i], ytrain[i])
    my_clf_pred = my_clf.predict(split_X[i])
    scores[i] = accuracy_score(split_y[i], my_clf_pred)
```

In [37]:
```
print(f"Точности на 1-ом, 2-ом и 3-м фолдах: {scores}")
```
```
Точности на 1-ом, 2-ом и 3-м фолдах: [0.25795334 0.24635375 0.2543092 ]
```