1. Рубежный контроль №2

Пряхин Владимир Геннадьевич, группа ИУ5-24М.
Вариант №4.

1.1. Задание

Необходимо решить задачу классификации текстов на основе выбранного датасета. Классификация может быть бинарной или многоклассовой. Целевой признак из выбранного датасета может иметь любой физический смысл.

Необходимо сформировать признаки на основе `CountVectorizer` или `TfidfVectorizer`.

В качестве классификаторов необходимо использовать два классификатора

1. KNeighborsClassifier
2. Complement Naive Bayes

Для каждого метода необходимо оценить качество классификации. Сделайте вывод о том, какой вариант векторизации признаков в паре с каким классификатором показал лучшее качество.

1.2. Решение

1.2.1. Загрузка и предобработка данных

[1]:
```
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.feature_extraction.text import CountVectorizer,␣
 ↪TfidfVectorizer
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from typing import Dict, Tuple
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.naive_bayes import ComplementNB
```

[2]:
```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
```

```
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))
```

Набор данных доступен по следующему адресу: https://www.kaggle.com/kritanjalijain/amazon-reviews?select=amazon_review_polarity_csv.tgz

Данные представляют собой отзывы на amazon. Содержит столбцы value, header, text. Эти 3 столбца соответствуют индексу класса (1 или 2), заголовку обзора и тексту обзора.

value — 1 для отрицательной и 2 для положительной.

header — заголовок отзыва.

text — тело обзора.

[3]:
```
# Загрузка данных
imdb_df = pd.read_csv("/home/hino/Загрузки/amazon_review_polarity_csv/
 ↪train.csv",header=None, names=['value', 'header','text'])
imdb_df.head()
```

[3]:

	value	header	text
0	2	Stuning even for the non-gamer	
1	2	The best soundtrack ever to anything.	
2	2	Amazing!	
3	2	Excellent Soundtrack	
4	2	Remember, Pull Your Jaw Off The Floor After He…	

```
0  This sound track was beautiful! It paints the …
1  I'm reading a lot of reviews saying that this …
2  This soundtrack is my favorite music of all ti…
3  I truly like this soundtrack and I enjoy video…
4  If you've played the game, you know how divine…
```

[4]:
```
imdb_df.shape
```

[4]:
```
(3600000, 3)
```

Исходные данные слишком большие, воспользуемся первыми 10000 строками.

[5]:
```
imdb_df = imdb_df[:10000]
```

Попробуем распознать индекс класса по заголовкам.

[6]:
```
imdb_df.drop('text', axis=1, inplace=True)
```

[7]:
```
imdb_df.shape
```

[7]:
```
(10000, 2)
```

[8]:
```
#Сформируем словарь для обучения моделей
vocab_list = imdb_df['header'].tolist()
vocab_list[1:10]
```

[8]:
```
['The best soundtrack ever to anything.',
 'Amazing!',
 'Excellent Soundtrack',
 'Remember, Pull Your Jaw Off The Floor After Hearing it',
 'an absolute masterpiece',
 'Buyer beware',
 'Glorious story',
 'A FIVE STAR BOOK',
 'Whispers of the Wicked Saints']
```

[9]:
```
vocabVect = CountVectorizer()
vocabVect.fit(vocab_list)
corpusVocab = vocabVect.vocabulary_
print('Количество сформированных признаков - {}'.format(len(corpusVocab)))
```

```
Количество сформированных признаков - 6995
```

[10]:
```
for i in list(corpusVocab)[1:10]:
    print('{}={}'.format(i, corpusVocab[i]))
```

```
even=2155
for=2475
the=6159
non=4205
gamer=2581
best=662
```

```
soundtrack=5723
ever=2159
to=6255
```

[11]:
```
test_features = vocabVect.transform(vocab_list)
```

[12]:
```
test_features
```

[12]:
```
<10000x6995 sparse matrix of type '<class 'numpy.int64'>'
        with 40572 stored elements in Compressed Sparse Row format>
```

[13]:
```
test_features.todense()
```

[13]:
```
matrix([[0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        …,
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]])
```

[14]:
```
# Размер нулевой строки
len(test_features.todense()[0].getA1())
```

[14]:
```
6995
```

[15]:
```
def VectorizeAndClassify(vectorizers_list, classifiers_list):
    for v in vectorizers_list:
        for c in classifiers_list:
            pipeline1 = Pipeline([("vectorizer", v), ("classifier", c)])
            score = cross_val_score(pipeline1, imdb_df['header'],␣
 ↪imdb_df['value'], scoring='accuracy', cv=3).mean()
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')
```

С использованием кросс-валидации попробуем применить к корпусу текстов различные варианты векторизации и классификации.

[16]:
```
vectorizers_list = [CountVectorizer(vocabulary = corpusVocab),␣
 ↪TfidfVectorizer(vocabulary = corpusVocab)]
classifiers_list = [ComplementNB(), KNeighborsClassifier()]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '02': 2, '03':
 ↪ 3,
'10': 4,
                               '100': 5, '101': 6, '103': 7, '1059': 8, '11':␣
 ↪9,
                               '12': 10, '121': 11, '13': 12, '133x': 13, '14':
 ↪ 14,
```

```
                            '144': 15, '14th': 16, '15': 17, '16': 18, '17':
 ↪ 19,
                            '18': 20, '1800': 21, '1800s': 22, '1840': 23,
                            '1875': 24, '1890': 25, '1900s': 26, '1911': 27,
                            '1912': 28, '1914': 29, …})
Модель для классификации - ComplementNB()
Accuracy = 0.776699175616422
============================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '02': 2, '03':
 ↪ 3,
'10': 4,
                            '100': 5, '101': 6, '103': 7, '1059': 8, '11':␣
 ↪9,
                            '12': 10, '121': 11, '13': 12, '133x': 13, '14':
 ↪ 14,
                            '144': 15, '14th': 16, '15': 17, '16': 18, '17':
 ↪ 19,
                            '18': 20, '1800': 21, '1800s': 22, '1840': 23,
                            '1875': 24, '1890': 25, '1900s': 26, '1911': 27,
                            '1912': 28, '1914': 29, …})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.7154984844615235
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '02': 2, '03':
 ↪ 3,
'10': 4,
                            '100': 5, '101': 6, '103': 7, '1059': 8, '11':␣
 ↪9,
                            '12': 10, '121': 11, '13': 12, '133x': 13, '14':
 ↪ 14,
                            '144': 15, '14th': 16, '15': 17, '16': 18, '17':
 ↪ 19,
                            '18': 20, '1800': 21, '1800s': 22, '1840': 23,
                            '1875': 24, '1890': 25, '1900s': 26, '1911': 27,
                            '1912': 28, '1914': 29, …})
Модель для классификации - ComplementNB()
Accuracy = 0.7717991955164322
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '02': 2, '03':
 ↪ 3,
'10': 4,
                            '100': 5, '101': 6, '103': 7, '1059': 8, '11':␣
 ↪9,
                            '12': 10, '121': 11, '13': 12, '133x': 13, '14':
 ↪ 14,
                            '144': 15, '14th': 16, '15': 17, '16': 18, '17':
 ↪ 19,
                            '18': 20, '1800': 21, '1800s': 22, '1840': 23,
                            '1875': 24, '1890': 25, '1900s': 26, '1911': 27,
```

```
                            '1912': 28, '1914': 29, …})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.6367997627597193
==========================
```

1.3. Вывод

Лучшую точность показал CountVectorizer и наивный байесовский классификатор (77,7%)