Рубежный контроль №2

Румянцев Олег

Группа ИУ5-22М

Тема: Методы обработки текстов.

Решение задачи классификации текстов.

Необходимо решить задачу классификации текстов на основе любого выбранного Вами датасета. Классификация может быть бинарной или многоклассовой. Целевой признак из выбранного Вами датасета может иметь любой физический смысл, примером является задача анализа тональности текста.

Необходимо сформировать два варианта векторизации признаков - на основе CountVectorizer и на основе TfidfVectorizer.

В качестве классификаторов необходимо использовать два классификатора по варианту для Вашей группы: RandomForestClassifier, Complement Naive Bayes (CNB)

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

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries insta
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from typing import Dict, Tuple
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classifi
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squ
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, Linea
from sklearn.naive_bayes import ComplementNB
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will li

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that g
# You can also write temporary files to /kaggle/temp/, but they won't be save

pd.set_option("display.max_columns", None)
```

```
/kaggle/input/amazon-fine-food-reviews/hashes.txt
/kaggle/input/amazon-fine-food-reviews/Reviews.csv
/kaggle/input/amazon-fine-food-reviews/database.sqlite
```

In [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]))
```

In [5]:
```
train = pd.read_csv('/kaggle/input/amazon-fine-food-reviews/Reviews.csv')
```

In [6]:
```
print(train.shape)
```

```
(568454, 10)
```

In [7]:
```
train.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenor
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenor
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

In [8]:
```
train.Score.value_counts()
```

Out[8]:
```
5    363122
4     80655
1     52268
3     42640
2     29769
Name: Score, dtype: int64
```

Очистка данных

In [23]:
```
import re
import nltk
from nltk.corpus import stopwords
stop = set(stopwords.words('english'))

def test(word):
    if word.isalpha() and len(word) > 2 and word.lower() not in stop:
        s=(sno.stem(word.lower()))
        return s
    else:
        pass

#initialising the snowball stemmer
sno = nltk.stem.SnowballStemmer('english')
def preprocess_sentence(w):
    w = re.sub('\t\n', '', w)
    w = re.sub(r'http\S+', '', w)
    w = re.sub(r"([?.!,])", r" \1 ", w)
    w = re.sub(r'[" "]+', " ", w)
    w = re.sub(r"[^a-zA-Za-яA-Я?.!,`']+", " ", w)

    w = w.strip().split()
    text = [test(x) for x in w if test(x)]

    return ' '.join(text)
```

In [24]:
```
train.Text = train.Text.apply(preprocess_sentence)
```

In [25]:
```
vocabVect = CountVectorizer()
vocabVect.fit(train.Text)
corpusVocab = vocabVect.vocabulary_
print('Количество сформированных признаков - {}'.format(len(corpusVocab)))
```

```
Количество сформированных признаков - 73376
```

In [13]:
```
for i in list(corpusVocab)[1:10]:
    print('{}={}'.format(i, corpusVocab[i]))
```

```
bought=10722
several=83580
of=64753
the=94178
vitality=101486
canned=13501
dog=27448
food=35881
products=73379
```

In [14]:
```
tfidfv = TfidfVectorizer(ngram_range=(1,3))
tfidf_ngram_features = tfidfv.fit_transform(train.Text)
tfidf_ngram_features
```

Out[14]:
```
<568454x14619665 sparse matrix of type '<class 'numpy.float64'>'
```

In [19]:
```
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, train.Text[:10000], train.Scor
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')
```

In [26]:
```
vectorizers_list = [CountVectorizer(vocabulary = corpusVocab), TfidfVectorize
classifiers_list = [RandomForestClassifier(), ComplementNB()]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
Векторизация - CountVectorizer(vocabulary={'aa': 0, 'aaa': 1, 'aaaa': 2, 'aaa
aa': 3,
                               'aaaaaa': 4, 'aaaaaaaaaa': 5, 'aaaaaaaaaaa': 6,
                               'aaaaaaaaaaaa': 7, 'aaaaaaaaaaaaa': 8,
                               'aaaaaaaaaaaaaa': 9, 'aaaaaaaaaaaaaaa': 10,
                               'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa': 1
1,
                               'aaaaaaaaaaaaaaaaaargh': 12,
                               'aaaaaaaaaaaaaaaaacccccccckkkkkk': 13,
                               'aaaaaaaaagghh': 14, 'aaaaaaah': 15,
                               'aaaaaaahhhhhh': 16, 'aaaaaaarrrrrggghhh': 17,
                               'aaaaaah': 18, 'aaaaaahhh': 19, 'aaaaaahhhh': 20,
                               'aaaaaahhhhh': 21, 'aaaaaahhhhhyaaaaaa': 22,
                               'aaaaaand': 23, 'aaaaaawwwwwwwww': 24,
                               'aaaaah': 25, 'aaaaahhhhhhhhhhhhhhh': 26,
                               'aaaaallll': 27, 'aaaaawsom': 28, 'aaaah': 29,
...})
Модель для классификации - RandomForestClassifier()
Accuracy = 0.6243000724787536
============================
Векторизация - CountVectorizer(vocabulary={'aa': 0, 'aaa': 1, 'aaaa': 2, 'aaa
aa': 3,
                               'aaaaaa': 4, 'aaaaaaaaaa': 5, 'aaaaaaaaaaa': 6,
                               'aaaaaaaaaaaa': 7, 'aaaaaaaaaaaaa': 8,
                               'aaaaaaaaaaaaaa': 9, 'aaaaaaaaaaaaaaa': 10,
                               'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa': 1
1,
                               'aaaaaaaaaaaaaaaaaargh': 12,
                               'aaaaaaaaaaaaaaaaacccccccckkkkkk': 13,
                               'aaaaaaaaagghh': 14, 'aaaaaaah': 15,
                               'aaaaaaahhhhhh': 16, 'aaaaaaarrrrrggghhh': 17,
                               'aaaaaah': 18, 'aaaaaahhh': 19, 'aaaaaahhhh': 20,
                               'aaaaaahhhhh': 21, 'aaaaaahhhhhyaaaaaa': 22,
                               'aaaaaand': 23, 'aaaaaawwwwwwwww': 24,
                               'aaaaah': 25, 'aaaaahhhhhhhhhhhhhhh': 26,
                               'aaaaallll': 27, 'aaaaawsom': 28, 'aaaah': 29,
...})
Модель для классификации - ComplementNB()
Accuracy = 0.6250998525167454
============================
Векторизация - TfidfVectorizer(vocabulary={'aa': 0, 'aaa': 1, 'aaaa': 2, 'aaa
aa': 3,
```

```
                                'aaaaaa': 4, 'aaaaaaaaaa': 5, 'aaaaaaaaaaa': 6,
                                'aaaaaaaaaaaa': 7, 'aaaaaaaaaaaaa': 8,
                                'aaaaaaaaaaaaaa': 9, 'aaaaaaaaaaaaaaa': 10,
                                'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa': 1
1,
                                'aaaaaaaaaaaaaaaaaaargh': 12,
                                'aaaaaaaaaaaaaaaaacccccccckkkkkk': 13,
                                'aaaaaaaaagghh': 14, 'aaaaaaah': 15,
                                'aaaaaaahhhhhh': 16, 'aaaaaaarrrrrggghhh': 17,
                                'aaaaaah': 18, 'aaaaaahhh': 19, 'aaaaaahhhh': 20,
                                'aaaaaahhhhh': 21, 'aaaaaahhhhhyaaaaaa': 22,
                                'aaaaaand': 23, 'aaaaaawwwwwwwww': 24,
                                'aaaaah': 25, 'aaaaahhhhhhhhhhhhhhh': 26,
                                'aaaaallll': 27, 'aaaaawsom': 28, 'aaaah': 29,
...})
Модель для классификации - RandomForestClassifier()
Accuracy = 0.6224999224577527
==========================
Векторизация - TfidfVectorizer(vocabulary={'aa': 0, 'aaa': 1, 'aaaa': 2, 'aaa
aa': 3,
                                'aaaaaa': 4, 'aaaaaaaaaa': 5, 'aaaaaaaaaaa': 6,
                                'aaaaaaaaaaaa': 7, 'aaaaaaaaaaaaa': 8,
                                'aaaaaaaaaaaaaa': 9, 'aaaaaaaaaaaaaaa': 10,
                                'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa': 1
1,
                                'aaaaaaaaaaaaaaaaaaargh': 12,
                                'aaaaaaaaaaaaaaaaacccccccckkkkkk': 13,
                                'aaaaaaaaagghh': 14, 'aaaaaaah': 15,
                                'aaaaaaahhhhhh': 16, 'aaaaaaarrrrrggghhh': 17,
                                'aaaaaah': 18, 'aaaaaahhh': 19, 'aaaaaahhhh': 20,
                                'aaaaaahhhhh': 21, 'aaaaaahhhhhyaaaaaa': 22,
                                'aaaaaand': 23, 'aaaaaawwwwwwwww': 24,
                                'aaaaah': 25, 'aaaaahhhhhhhhhhhhhhh': 26,
                                'aaaaallll': 27, 'aaaaawsom': 28, 'aaaah': 29,
...})
Модель для классификации - ComplementNB()
Accuracy = 0.6182000023637636
```

Лучший результат показала модель ComplementNB с CountVectorizer

In [ ]: