Рубежный контроль № 2

Тема: Методы обработки текстов

Студент и вариант:

Студент ИУ5-21М Ханмурзин Тагир
Вариант №1: CountVectorizer, TfidfVectorizer, LogisticRegression, Multinomial Naive Bayes (MNB)

Датасет:

Набор текстовых данных разделённых по **20** группам

Описание:

```
The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups.
The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.
```

In [34]:

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
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, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
import seaborn as sns
from collections import Counter
from sklearn.datasets import fetch_20newsgroups
import matplotlib.pyplot as plt

# Отключим предупреждения
import warnings
warnings.filterwarnings('ignore')
```

In [23]:

```
# Обозначим категории, которые желаем загрузить
categories = ["sci.crypt",
              "sci.electronics",
              "talk.religion.misc",
              "rec.sport.baseball"]
newsgroups = fetch_20newsgroups(subset='train', categories=categories)
data = newsgroups['data']
```

In [24]:

```
# Используемся отработанной функцией вычисления метрики
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 [25]:

```
# С помощью CountVectorizer преобразуем коллекцию текстовых данных в матрицу счётчиков токенов
vocabVect = CountVectorizer()
vocabVect.fit(data)
corpusVocab = vocabVect.vocabulary_
print('Количество сформированных признаков - {}'.format(len(corpusVocab)))
```

```
Количество сформированных признаков - 33282
```

In [26]:

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

```
philly=23632
ravel=25268
udel=30929
edu=12456
robert=26356
hite=16186
subject=29047
re=25308
dave=10693
```

In [36]:

```
test_features = vocabVect.transform(data)
```

In [37]:

```
test_features
```

Out[37]:

```
<2160x33282 sparse matrix of type '<class 'numpy.int64'>'
 with 339881 stored elements in Compressed Sparse Row format>
```

In [38]:

```
test_features.todense()
```

Out[38]:

```
matrix([[0, 3, 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]], dtype=int64)
```

In [28]:

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

Out[28]:

```
33282
```

In [42]:

```
vocabVect.get_feature_names()[100:120]
```

Out[42]:

```
['00101101b',
 '00101110',
 '00101110b',
 '00101111',
 '00101111b',
 '0011',
 '00110000',
 '00110000b',
 '00110001',
 '00110001b',
 '00110010',
 '00110010b',
 '00110011',
 '00110011b',
 '00110100',
 '00110100b',
 '00110101',
 '00110101b',
 '00110110',
 '00110110b']
```

In [30]:

```
# Функция применения к текстовым данных различных вариантов векторизации и классификации
с использованием кросс-валидации
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, newsgroups['data'], newsgroups['target'],
scoring='accuracy', cv=3).mean()
            print('Векторизация - {}'.format(v))
```

```
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')
```

In [33]:

```
# Определяем варианты векторизации и классификации и устанавливаем преднастройки
vectorizers_list = [CountVectorizer(vocabulary = corpusVocab), TfidfVectorizer(vocabular
y = corpusVocab)]
classifiers_list = [LogisticRegression(), MultinomialNB()]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '00000000': 2, '00000000b':
3,
                               '00000001': 4, '00000001b': 5, '00000010': 6,
                               '00000010b': 7, '00000011': 8, '00000011b': 9,
                               '00000100': 10, '00000100b': 11, '00000101': 12,
                               '00000101b': 13, '00000110': 14, '00000110b': 15,
                               '00000111': 16, '00000111b': 17, '00001000': 18,
                               '00001000b': 19, '00001001': 20, '00001001b': 21,
                               '00001010': 22, '00001010b': 23, '00001011': 24,
                               '00001011b': 25, '00001100': 26, '00001100b': 27,
                               '00001101': 28, '00001101b': 29, ...})
Модель для классификации - LogisticRegression()
Accuracy = 0.9564814814814815
============================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '00000000': 2, '00000000b':
3,
                               '00000001': 4, '00000001b': 5, '00000010': 6,
                               '00000010b': 7, '00000011': 8, '00000011b': 9,
                               '00000100': 10, '00000100b': 11, '00000101': 12,
                               '00000101b': 13, '00000110': 14, '00000110b': 15,
                               '00000111': 16, '00000111b': 17, '00001000': 18,
                               '00001000b': 19, '00001001': 20, '00001001b': 21,
                               '00001010': 22, '00001010b': 23, '00001011': 24,
                               '00001011b': 25, '00001100': 26, '00001100b': 27,
                               '00001101': 28, '00001101b': 29, ...})
Модель для классификации - MultinomialNB()
Accuracy = 0.9805555555555556
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '00000000': 2, '00000000b':
3,
                               '00000001': 4, '00000001b': 5, '00000010': 6,
                               '00000010b': 7, '00000011': 8, '00000011b': 9,
                               '00000100': 10, '00000100b': 11, '00000101': 12,
                               '00000101b': 13, '00000110': 14, '00000110b': 15,
                               '00000111': 16, '00000111b': 17, '00001000': 18,
                               '00001000b': 19, '00001001': 20, '00001001b': 21,
                               '00001010': 22, '00001010b': 23, '00001011': 24,
                               '00001011b': 25, '00001100': 26, '00001100b': 27,
                               '00001101': 28, '00001101b': 29, ...})
Модель для классификации - LogisticRegression()
Accuracy = 0.9523148148148147
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '00000000': 2, '00000000b':
3,
                               '00000001': 4, '00000001b': 5, '00000010': 6,
                               '00000010b': 7, '00000011': 8, '00000011b': 9,
                               '00000100': 10, '00000100b': 11, '00000101': 12,
                               '00000101b': 13, '00000110': 14, '00000110b': 15,
                               '00000111': 16, '00000111b': 17, '00001000': 18,
                               '00001000b': 19, '00001001': 20, '00001001b': 21,
                               '00001010': 22, '00001010b': 23, '00001011': 24,
                               '00001011b': 25, '00001100': 26, '00001100b': 27,
                               '00001101': 28, '00001101b': 29, ...})
Модель для классификации - MultinomialNB()
Accuracy = 0.8898148148148147
============================
```

`CountVectorizer` **с** `Multinomial Naive Bayes` **показали лучший результат - 0.9805**

In [ ]:

```
```