Рубежный контроль №2

Тислюк Дмитрий

ИУ5-22М

В [20]:

```
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
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.naive_bayes import ComplementNB
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
from collections import Counter
from sklearn.datasets import fetch_20newsgroups
import matplotlib.pyplot as plt

%matplotlib inline
sns.set(style="ticks")
```

В [21]:

```
categories = ["comp.sys.ibm.pc.hardware", "rec.motorcycles", "sci.electronics", "alt.atheis
newsgroups = fetch_20newsgroups(subset='train', categories=categories)
data = newsgroups['data']
```

В [22]:

```
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]))
```

В [23]:

```
vocabVect = CountVectorizer()
vocabVect.fit(data)
corpusVocab = vocabVect.vocabulary_
print('Количество сформированных признаков - {}'.format(len(corpusVocab)))
```

```
Количество сформированных признаков - 31660
```

B [24]:

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

```
bil=6823
okcforum=21419
osrhe=21683
edu=11993
bill=6827
conner=9374
subject=27588
re=24066
americans=5315
```

B [25]:

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

Out[25]:

```
<2259x31660 sparse matrix of type '<class 'numpy.int64'>'
        with 310255 stored elements in Compressed Sparse Row format>
```

B [26]:

```
len(test_features.todense()[0].getA1())
```

Out[26]:

```
31660
```

B [29]:

```
vocabVect.get_feature_names()[31655:]
```

Out[29]:

```
['zzr1100', 'zzzz', 'zzzzzz', '³ation', 'ýé']
```

B [30]:

```
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'], sc
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')
```

В [31]:

```
vectorizers_list = [CountVectorizer(vocabulary = corpusVocab), TfidfVectorizer(vocabulary =
classifiers_list = [RandomForestClassifier(), ComplementNB()]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
Векторизация - CountVectorizer(analyzer='word', binary=False, decode_error
='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='conten
t',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None,
                vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                            '0000000004': 4, '0000000005': 5, '0000000667':
6,
                            '0000001200': 7, '000042': 8, '000094': 9,
                            '0001': 10, '0001x7c': 11, '0002': 12, '0003': 1
3,
                            '000406': 14, '0005111312': 15,
                            '0005111312na1em': 16, '000531': 17, '00072': 1
8,
                            '000851': 19, '000rpm': 20, '001': 21,
                            '00100111b': 22, '0011': 23, '001125': 24,
                            '0013': 25, '001813': 26, '002222': 27,
                            '002937': 28, '003029': 29, ...})
Модель для классификации - RandomForestClassifier(bootstrap=True, ccp_alpha=
0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
Accuracy = 0.9198760513501548
============================
Векторизация - CountVectorizer(analyzer='word', binary=False, decode_error
='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='conten
t',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None,
                vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                            '0000000004': 4, '0000000005': 5, '0000000667':
6,
                            '0000001200': 7, '000042': 8, '000094': 9,
                            '0001': 10, '0001x7c': 11, '0002': 12, '0003': 1
3,
                            '000406': 14, '0005111312': 15,
                            '0005111312na1em': 16, '000531': 17, '00072': 1
8,
                            '000851': 19, '000rpm': 20, '001': 21,
                            '00100111b': 22, '0011': 23, '001125': 24,
                            '0013': 25, '001813': 26, '002222': 27,
                            '002937': 28, '003029': 29, ...})
Модель для классификации - ComplementNB(alpha=1.0, class_prior=None, fit_pri
```

```
or=True, norm=False)
Accuracy = 0.9641434262948207
============================
Векторизация - TfidfVectorizer(analyzer='word', binary=False, decode_error
='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=No
ne,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use...
                vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                            '0000000004': 4, '0000000005': 5, '0000000667':
6,
                            '0000001200': 7, '000042': 8, '000094': 9,
                            '0001': 10, '0001x7c': 11, '0002': 12, '0003': 1
3,
                            '000406': 14, '0005111312': 15,
                            '0005111312na1em': 16, '000531': 17, '00072': 1
8,
                            '000851': 19, '000rpm': 20, '001': 21,
                            '00100111b': 22, '0011': 23, '001125': 24,
                            '0013': 25, '001813': 26, '002222': 27,
                            '002937': 28, '003029': 29, ...})
Модель для классификации - RandomForestClassifier(bootstrap=True, ccp_alpha=
0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
Accuracy = 0.9096945551128818
============================
Векторизация - TfidfVectorizer(analyzer='word', binary=False, decode_error
='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=No
ne,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use...
                vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                            '0000000004': 4, '0000000005': 5, '0000000667':
6,
                            '0000001200': 7, '000042': 8, '000094': 9,
                            '0001': 10, '0001x7c': 11, '0002': 12, '0003': 1
3,
                            '000406': 14, '0005111312': 15,
                            '0005111312na1em': 16, '000531': 17, '00072': 1
8,
                            '000851': 19, '000rpm': 20, '001': 21,
                            '00100111b': 22, '0011': 23, '001125': 24,
                            '0013': 25, '001813': 26, '002222': 27,
                            '002937': 28, '003029': 29, ...})
Модель для классификации - ComplementNB(alpha=1.0, class_prior=None, fit_pri
or=True, norm=False)
```

```
Accuracy = 0.9601593625498008
============================
```

B [ ]:

```

```

Лучшую точность показал CountVectorizer и ComplementNB