МГТУ им. Н. Э. Баумана, кафедра ИУ5

курс “Технология машинного обучения”

Рубежный контроль №2

«Технологии использования и оценки моделей машинного обучения»

ВЫПОЛНИЛ:

Матюнин да Вейга Р.А.

Группа: ИУ5-61Б

ПРОВЕРИЛ:

Гапанюк Ю.Е.

Москва 2020

Задание №1:

Данный вариант выполняется на основе материалов лекции часть 1 и часть 2.

Необходимо решить задачу классификации текстов на основе любого выбранного Вами датасета (кроме примера, который рассматривался в лекции). 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 признаки на основе CountVectorizer или TfidfVectorizer.

В качестве классификаторов необходимо использовать два классификатора, не относящихся к наивным Байесовским методам (например, LogisticRegression, LinearSVC), а также Multinomial Naive Bayes (MNB), Complement Naive Bayes (CNB), Bernoulli Naive Bayes.

Для каждого метода необходимо оценить качество классификации с помощью хотя бы одной метрики качества классификации (например, Accuracy).

Сделать выводы о том, какой классификатор осуществляет более качественную классификацию на Вашем наборе данных.

Выполненная работа:

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

Загрузка данных

In [2]:
```
data = pd.read_csv("dataset/sentiment labelled sentences/amazon_cells_labelled.txt",
                   delimiter='\t', header=None, names=['Text', 'Value'])
data.head()
```

Out[2]:

	Text	Value
0	So there is no way for me to plug it in here i...	0
1	Good case, Excellent value.	1
2	Great for the jawbone.	1
3	Tied to charger for conversations lasting more...	0
4	The mic is great.	1

In [3]:
```
data.shape
```

Out[3]: (1000, 2)

Общий словарь для обучения моделей

In [4]:
```
vocab_list = data.Text.tolist()
vocab_list[:10]
```

Out[4]:
```
['So there is no way for me to plug it in here in the US unless I go by a converter.',
 'Good case, Excellent value.',
 'Great for the jawbone.',
 'Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!!',
 'The mic is great.',
 'I have to jiggle the plug to get it to line up right to get decent volume.',
 'If you have several dozen or several hundred contacts, then imagine the fun of sending each of t
hem one by one.',
 'If you are Razr owner...you must have this!',
 'Needless to say, I wasted my money.',
 'What a waste of money and time!.']
```

In [5]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

In [6]:
```
vocabVect = CountVectorizer()
vocabVect.fit_transform(vocab_list)
```

Out[6]:
```
<1000x1847 sparse matrix of type '<class 'numpy.int64'>'
        with 9130 stored elements in Compressed Sparse Row format>
```

Количество признаков = 1847

In [7]:
```
len(vocabVect.get_feature_names())
```

Out[7]:
```
1847
```

In [8]:
```
corpusVocab = vocabVect.vocabulary_
```

Признак и его индекс в словаре

In [9]:
```
for i in list(corpusVocab)[:10]:
    print('{}={}'.format(i, corpusVocab[i]))
```

```
so=1491
there=1609
is=854
no=1074
way=1766
for=653
me=993
to=1640
plug=1212
it=857
```

Векторизация текста

In [10]:
```
test_features = vocabVect.transform(vocab_list)
test_features
```

Out[10]:
```
<1000x1847 sparse matrix of type '<class 'numpy.int64'>'
        with 9130 stored elements in Compressed Sparse Row format>
```

In [11]:
```
test_features.todense()
```

Out[11]:
```
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]], dtype=int64)
```

1000 строк - 1000 предложений в документе

1847 столбцов - 1847 уникальных значений в документе

N-граммы

In [12]:
```
ncv = CountVectorizer(ngram_range=(1, 3))
ngram_features = ncv.fit_transform(vocab_list)
ngram_features
```

Out[12]:
```
<1000x15088 sparse matrix of type '<class 'numpy.int64'>'
        with 25421 stored elements in Compressed Sparse Row format>
```

In [13]:
```
ncv.get_feature_names()[100:120]
```

Out[13]:
```
['able to',
 'able to do',
 'able to roam',
 'able to use',
 'abound',
 'about',
 'about 10',
 'about 10 of',
 'about 18',
 'about 18 months',
 'about inches',
 'about inches above',
 'about it',
 'about it is',
 'about the',
 'about the consumer',
 'about this',
 'about this headset',
 'about this phone',
 'about this product']
```

Векторизация TfidVectorizer

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

Out[14]:
```
<1000x15088 sparse matrix of type '<class 'numpy.float64'>'
        with 25421 stored elements in Compressed Sparse Row format>
```

In [15]:
```
tfidf_ngram_features.todense()
```

Out[15]:
```
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.]])
```

In [16]:
```
# Непустые значения нулевой строки
[i for i in tfidf_ngram_features.todense()[0].getA1() if i>0][:10]
```

Out[16]:
```
[0.12296719867492838,
 0.15534944608172185,
 0.15534944608172185,
 0.06830400100424172,
 0.1255030252282181,
 0.15534944608172185,
 0.1283779082640305,
 0.15534944608172185,
 0.15534944608172185,
 0.13562203495268255]
```

Решение задачи

In [17]:
```
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, data['Text'], data['Value'], scoring='accuracy', cv=3)
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')
```

In [18]:
```
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
```

In [19]:
```
vectorizers_list = [CountVectorizer(vocabulary = corpusVocab), TfidfVectorizer(vocabulary = corpusVoc
classifiers_list = [LogisticRegression(C=3.0), LinearSVC(), KNeighborsClassifier()]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
Векторизация - CountVectorizer(analyzer='word', binary=False,
decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8',
input='content',
                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={'10': 0, '100': 1, '11': 2, '12': 3, '13': 4,
                            '15': 5, '15g': 6, '18': 7, '20': 8, '2000': 9,
                            '2005': 10, '2160': 11, '24': 12, '2mp': 13,
                            '325': 14, '350': 15, '375': 16, '3o': 17, '42':
18,
                            '44': 19, '45': 20, '4s': 21, '50': 22, '5020':
23,
                            '510': 24, '5320': 25, '680': 26, '700w': 27,
                            '8125': 28, '8525': 29, ...})
Модель для классификации - LogisticRegression(C=3.0, class_weight=None,
dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
Accuracy = 0.8069896243548937
============================
Векторизация - CountVectorizer(analyzer='word', binary=False,
decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8',
input='content',
                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={'10': 0, '100': 1, '11': 2, '12': 3, '13': 4,
                            '15': 5, '15g': 6, '18': 7, '20': 8, '2000': 9,
                            '2005': 10, '2160': 11, '24': 12, '2mp': 13,
                            '325': 14, '350': 15, '375': 16, '3o': 17, '42':
18,
                            '44': 19, '45': 20, '4s': 21, '50': 22, '5020':
23,
                            '510': 24, '5320': 25, '680': 26, '700w': 27,
                            '8125': 28, '8525': 29, ...})
Модель для классификации - LinearSVC(C=1.0, class_weight=None, dual=True,
fit_intercept=True,
```

```
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
Accuracy = 0.8249956543369716
============================
Векторизация - CountVectorizer(analyzer='word', binary=False,
decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8',
input='content',
                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={'10': 0, '100': 1, '11': 2, '12': 3, '13': 4,
                            '15': 5, '15g': 6, '18': 7, '20': 8, '2000': 9,
                            '2005': 10, '2160': 11, '24': 12, '2mp': 13,
                            '325': 14, '350': 15, '375': 16, '3o': 17, '42':
18,
                            '44': 19, '45': 20, '4s': 21, '50': 22, '5020':
23,
                            '510': 24, '5320': 25, '680': 26, '700w': 27,
                            '8125': 28, '8525': 29, ...})
Модель для классификации - KNeighborsClassifier(algorithm='auto',
leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
Accuracy = 0.6619733505961051
============================
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=None,
                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_idf=True,
                vocabulary={'10': 0, '100': 1, '11': 2, '12': 3, '13': 4,
                            '15': 5, '15g': 6, '18': 7, '20': 8, '2000': 9,
                            '2005': 10, '2160': 11, '24': 12, '2mp': 13,
                            '325': 14, '350': 15, '375': 16, '3o': 17, '42':
18,
                            '44': 19, '45': 20, '4s': 21, '50': 22, '5020':
23,
                            '510': 24, '5320': 25, '680': 26, '700w': 27,
                            '8125': 28, '8525': 29, ...})
Модель для классификации - LogisticRegression(C=3.0, class_weight=None,
dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
Accuracy = 0.8109936283588978
============================
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=None,
                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_idf=True,
                vocabulary={'10': 0, '100': 1, '11': 2, '12': 3, '13': 4,
```

```
                            '15': 5, '15g': 6, '18': 7, '20': 8, '2000': 9,
                            '2005': 10, '2160': 11, '24': 12, '2mp': 13,
                            '325': 14, '350': 15, '375': 16, '3o': 17, '42':
18,
                            '44': 19, '45': 20, '4s': 21, '50': 22, '5020':
23,
                            '510': 24, '5320': 25, '680': 26, '700w': 27,
                            '8125': 28, '8525': 29, ...})
Модель для классификации - LinearSVC(C=1.0, class_weight=None, dual=True,
fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
Accuracy = 0.8109816403229577
============================
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=None,
                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_idf=True,
                vocabulary={'10': 0, '100': 1, '11': 2, '12': 3, '13': 4,
                            '15': 5, '15g': 6, '18': 7, '20': 8, '2000': 9,
                            '2005': 10, '2160': 11, '24': 12, '2mp': 13,
                            '325': 14, '350': 15, '375': 16, '3o': 17, '42':
18,
                            '44': 19, '45': 20, '4s': 21, '50': 22, '5020':
23,
                            '510': 24, '5320': 25, '680': 26, '700w': 27,
                            '8125': 28, '8525': 29, ...})
Модель для классификации - KNeighborsClassifier(algorithm='auto',
leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
Accuracy = 0.7709895524266782
============================
```

Разделение выборки ¶

In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(data['Text'], data['Value'], test_size=0.5, rando
```

In [22]:
```
from typing import Dict, Tuple
from sklearn.metrics import accuracy_score, balanced_accuracy_score

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 [23]:
```
def sentiment(v, c):
    model = Pipeline(
        [("vectorizer", v),
         ("classifier", c)])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print_accuracy_score_for_classes(y_test, y_pred)
```

In [24]:
```
types = [[TfidfVectorizer(), LogisticRegression(C=5.0)],
         [TfidfVectorizer(ngram_range=(1,3)), LogisticRegression(C=5.0)],
         [TfidfVectorizer(ngram_range=(2,3)), LogisticRegression(C=5.0)],
         [TfidfVectorizer(ngram_range=(1,4)), LogisticRegression(C=5.0)],
         [TfidfVectorizer(ngram_range=(2,4)), LogisticRegression(C=5.0)]]
for type_ in types:
    sentiment(*type_)
    print("============================")
```

```
Метка    Accuracy
0        0.8099173553719008
1        0.8023255813953488
============================
Метка    Accuracy
0        0.7975206611570248
1        0.7984496124031008
============================
Метка    Accuracy
0        0.7727272727272727
1        0.6162790697674418
============================
Метка    Accuracy
0        0.7975206611570248
1        0.8178294573643411
============================
Метка    Accuracy
0        0.768595041322314
1        0.6162790697674418
============================
```


- Ноутбук с выполненной работой и отчет размещены в репозитории на github: https://github.com/Yorati/TMO