Рубежный контроль №2

Выполнил: Березин Иван Сергеевич, группа ИУ5-23м

Вариант №1. Классификация текстов на основе методов наивного Байеса

Набор данных

На Kaggle.com найден Amazon Fine Food Reviews - набор данных содержащий отзывы покупателей. Задачей будет получение параметра score (оценка по 5 балльной шкале) на основе написанного отзыва. Для этого необходимо выделить два признака, обработать их и затем отправить на обучение

In [2]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from typing import Dict, Tuple
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.naive_bayes import MultinomialNB, ComplementNB, BernoulliNB
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
%matplotlib inline
data = pd.read_csv('data/reviews.csv')
```

In [7]:

```
data = data[['Score', 'Text']]
data=data.dropna(axis=0, how='any')
data = data[:100000]
print(data.shape)
print('data type: \n{}'.format(data.dtypes))
```

```
(100000, 2)
data type:
Score     int64
Text     object
dtype: object
```

In [8]:

```
X_train, X_test, y_train, y_test = train_test_split(data['Text'], data['Score'], test_size=0.4, ran
dom_state=1)
```

In [9]:

```
def calc(v, c):
    model = Pipeline(
        [("vectorizer", v),
         ("classifier", c)])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(v)
    print("+{}".format(c))
    d = {'t': y_test, 'p': y_pred}
    df = pd.DataFrame(data=d)
    classes = np.unique(y_test)
    res = dict()
    for c in classes:
        temp_data_flt = df[df['t']==c]
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        res[c] = temp_acc
```

```
    if len(res)>0:
        print('Points \t Accuracy')
    for i in res:
        print('{} \t {:.2%}'.format(i, res[i]))
    print('\n\n')
```

In [10]:

```
classificators = [LogisticRegression(C=5.0), MultinomialNB(), ComplementNB(), BernoulliNB()]
vectorizers = [TfidfVectorizer(), CountVectorizer()]
```

In [11]:

```
for classificator in classificators:
  for vectorizer in vectorizers:
    calc(vectorizer, classificator)
```

```
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=None)
+LogisticRegression(C=5.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
Points   Accuracy
1   61.53%
2   17.79%
3   28.31%
4   27.29%
5   94.59%
```

```
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=None)
+LogisticRegression(C=5.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
Points   Accuracy
1   58.96%
2   22.05%
3   34.11%
4   30.71%
5   90.58%
```

```
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,
```

```
        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=None)
+MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Points   Accuracy
1   0.58%
2   0.00%
3   0.03%
4   0.14%
5   99.99%



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=None)
+MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Points   Accuracy
1   57.68%
2   8.61%
3   21.45%
4   32.20%
5   90.60%



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=None)
+ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
Points   Accuracy
1   53.33%
2   5.42%
3   9.80%
4   8.73%
5   96.66%



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=None)
+ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
Points   Accuracy
1   73.38%
2   11.79%
3   26.28%
4   29.73%
5   87.41%



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=None)
+BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
Points   Accuracy
1   35.90%
2   4.39%
3   18.92%
4   28.10%
5   84.49%
```

```
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=None)
+BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
Points   Accuracy
1   35.90%
2   4.39%
3   18.92%
4   28.10%
5   84.49%
```

Вывод

На основе полученного можно сделать вывод, что лучшим методом для определения высшей оценки является TfidfVectorizer с MultinomialNB с точностью 99,99%. Более точный для опредления наилучшей и наихудшей оценок является TfidfVectorizer с LogisticRegression определил оценки 1 и 5 с точностью 62 и 95 соответственно.