Буклин С. В. ИУ5-21М

РК №2 по курсу "Методы машинного обучения"

Вариант №1. Классификация текстов на основе методов наивного Байеса

- 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 признаки на основе CountVectorizer или TfidfVectorizer. В качестве классификаторов необходимо использовать один из классификаторов, не относящихся к наивным Байесовским методам (например, LogisticRegression), 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);
- Сделате выводы о том, какой классификатор осуществляет более качественную классификацию на Вашем наборе данных.

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/con
tent/drive", force_remount=True).
```

In [0]:

```
from google.colab import files
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
os.listdir()
data = pd.read_csv('drive/My Drive/Colab Notebooks/AppleStore.csv', sep=",")
```

Выбранный датасет

Датасет по приложениям в Apple Store

Предобработка датасета

In [26]:

```
rating = data[['prime_genre', 'user_rating']]
rating.head()
```

Out[26]:

	prime_genre	user_rating
0	Games	4.0
1	Productivity	4.0
2	Weather	3.5
3	Shopping	4.0
4	Reference	4.5

In [27]:

```
rating_cleaned = rating.dropna(axis=0, how='any')
float_rating = rating_cleaned['user_rating']
(rating.shape, rating_cleaned.shape)
```

Out[27]:

```
((7197, 2), (7197, 2))
```

In [28]:

```
rating_cleaned['user_rating'] = rating_cleaned['user_rating'].astype(int)
rating_cleaned.head()
```

Out[28]:

	prime_genre	user_rating
0	Games	4
1	Productivity	4
2	Weather	3
3	Shopping	4
4	Reference	4

Непосредственное обучение на различных классификаторах

In [0]:

```
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
```

In [0]:

```
X_train, X_test, y_train, y_test = train_test_split(rating_cleaned['prime_genre'], rating_cleaned['user_rati
ng'], test_size=0.4, random_state=1)
```

In [0]:

```
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, v, c):
    """
    Вывод метрики accuracy для каждого класса
    """
    print("Признаки сформированы на\n{}".format(v))
    print("\nКлассификатор\n{}".format(c))
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
      if i > 5:
        pass
      else:
        print('{} \t {:.2%}'.format(i, accs[i]))
    print('\n\n')
```

In [0]:

```
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, v, c)
```

In [0]:

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

In [35]:

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

```
Признаки сформированы на
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)

Классификатор
LogisticRegression(C=5.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
Метка    Accuracy
0        3.77%
1        0.00%
2        0.00%
3        3.82%
4        98.03%
5        0.00%



Признаки сформированы на
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)

Классификатор
LogisticRegression(C=5.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
Метка    Accuracy
0        3.77%
1        0.00%
2        0.00%
3        3.82%
4        98.03%
5        0.00%



Признаки сформированы на
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)

Классификатор
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Метка    Accuracy
0        3.77%
1        0.00%
2        0.00%
3        3.82%
4        98.03%
5        0.00%



Признаки сформированы на
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)

Классификатор
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Метка    Accuracy
0        3.77%
1        0.00%
2        0.00%
3        8.31%
4        95.94%
```

```
5        0.00%



Признаки сформированы на
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)

Классификатор
ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
Метка    Accuracy
0        12.94%
1        0.00%
2        9.60%
3        35.51%
4        71.54%
5        7.34%



Признаки сформированы на
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)

Классификатор
ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
Метка    Accuracy
0        12.94%
1        0.00%
2        9.60%
3        35.51%
4        71.54%
5        7.34%



Признаки сформированы на
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)

Классификатор
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
Метка    Accuracy
0        8.89%
1        0.00%
2        0.00%
3        8.31%
4        95.36%
5        0.00%



Признаки сформированы на
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)

Классификатор
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
Метка    Accuracy
0        8.89%
1        0.00%
2        0.00%
```

```
3       8.31%
4       95.36%
5       0.00%
```

Вывод

На основе полученного можно сделать вывод, что лучшим методом в данной ситуации является CountVectorizer с ComplementNB, который показал более равномерную оценку, по сравнению с другими, определявшеми только 4.