# Рубежный контроль №2

**Студент**: Хуан Яовэнь
**Группа**: ИУ5И-21М

Тема: Методы обработки текстов.

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 два варианта векторизации признаков - на основе CountVectorizer и на основе TfidfVectorizer.

**Классификатор**:LogisticRegression,Multinomial Naive Bayes - MNB

In [2]:

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB
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
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

In [3]:

```
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]))
```

## Загрузка данных

Работа с наборами данных"Amazon Alexa Reviews" из Kaggle.
About the Data

This dataset consists of a nearly 3000 Amazon customer reviews (input text), star ratings, date of review, variant and feedback of various amazon Alexa products like Alexa Echo, Echo dots, Alexa Firesticks etc. for learning how to train Machine for sentiment analysis.

In [16]:

```
amazon_rev=pd.read_csv("amazon_alexa.tsv", sep='\t')
amazon_rev.head()
```

Out[16]:

| | rating | date | variation | verified_reviews | feedback |
|---|---|---|---|---|---|
| **0** | 5 | 31-Jul-18 | Charcoal Fabric | Love my Echo! | 1 |
| **1** | 5 | 31-Jul-18 | Charcoal Fabric | Loved it! | 1 |
| **2** | 4 | 31-Jul-18 | Walnut Finish | Sometimes while playing a game, you can answer... | 1 |
| **3** | 5 | 31-Jul-18 | Charcoal Fabric | I have had a lot of fun with this thing. My 4 ... | 1 |
| **4** | 5 | 31-Jul-18 | Charcoal Fabric | Music | 1 |

In [17]:

```
amazon_rev.shape
```

Out[17]:

```
(3150, 5)
```

Только держать колонки "verified_reviews" и "feedback".

In [20]:

```
amazon_df = pd.DataFrame(amazon_rev,columns=['verified_reviews','feedback'])
amazon_df.columns = ['text','value']
amazon_df.head()
```

Out[20]:

| | text | value |
|---|---|---|
| **0** | Love my Echo! | 1 |
| **1** | Loved it! | 1 |
| **2** | Sometimes while playing a game, you can answer... | 1 |
| **3** | I have had a lot of fun with this thing. My 4 ... | 1 |
| **4** | Music | 1 |

In [22]:

```
#Сформируем общий словарь
vocab_list = amazon_df['text'].tolist()
vocab_list[1:10]
```

Out[22]:

```
['Loved it!',
 'Sometimes while playing a game, you can answer a question correctly but Alexa says
you got it wrong and answers the same as you.  I like being able to turn lights on a
nd off while away from home.',
 'I have had a lot of fun with this thing. My 4 yr old learns about dinosaurs, i con
trol the lights and play games like categories. Has nice sound when playing music as
well.',
 'Music',
 'I received the echo as a gift. I needed another Bluetooth or something to play mus
ic easily accessible, and found this smart speaker. Can't wait to see what else it
can do.',
 'Without having a cellphone, I cannot use many of her features. I have an iPad but
do not see that of any use.  It IS a great alarm.  If u r almost deaf, you can hear
her alarm in the bedroom from out in the living room, so that is reason enough to ke
ep her.It is fun to ask random questions to hear her response.  She does not seem to
be very smartbon politics yet.',
 "I think this is the 5th one I've purchased. I'm working on getting one in every ro
om of my house. I really like what features they offer specifily playing music on al
l Echos and controlling the lights throughout my house.",
 'looks great',
 'Love it! I've listened to songs I haven't heard since childhood! I get the news,
weather, information! It's great!']
```

In [23]:

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

```
Количество сформированных признаков - 4044
```

In [24]:

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

```
my=2320
echo=1160
loved=2151
it=1933
sometimes=3289
while=3945
playing=2640
game=1504
you=4028
```

## Векторизация текста на основе модели "мешка слов"

Векторизация текста поддерживается библиотекой scikit-learn.

## Использование класса CountVectorizer

Подсчитывает количество слов словаря, входящих в данный текст.

In [25]:

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

In [26]:

```
test_features
```

Out[26]:

```
<3150x4044 sparse matrix of type '<class 'numpy.int64'>'
	with 60852 stored elements in Compressed Sparse Row format>
```

In [27]:

```
test_features.todense()
```

Out[27]:

```
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)
```

In [28]:

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

Out[28]:

```
4044
```

In [29]:

```
# Непустые значения нулевой строки
[i for i in test_features.todense()[0].getA1() if i>0]
```

Out[29]:

```
[1, 1, 1]
```

In [30]:

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

Out[30]:

```
['according',
 'accordingly',
 'account',
 'accounts',
 'accuracy',
 'accurate',
 'accurately',
 'accustom',
 'acknowledge',
 'acoustical',
 'across',
 'act',
 'acting',
 'action',
 'actions',
 'activate',
 'activated',
 'activates',
 'activating',
 'activation']
```

## Использование класса TfidfVectorizer

Вычисляет специфичность текста в корпусе текстов на основе метрики TF-IDF.

In [31]:

```
tfidfv = TfidfVectorizer(ngram_range=(1,3))
tfidf_ngram_features = tfidfv.fit_transform(vocab_list)
tfidf_ngram_features
```

Out[31]:

```
<3150x78649 sparse matrix of type '<class 'numpy.float64'>'
	with 200215 stored elements in Compressed Sparse Row format>
```

In [32]:

```
tfidf_ngram_features.todense()
```

Out[32]:

```
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 [33]:

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

Out[33]:

```
78649
```

In [34]:

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

Out[34]:

```
[0.29583592663701436,
 0.25907502163192725,
 0.49177728048214797,
 0.5583340776813701,
 0.25070598596889654,
 0.47846202605627475]
```

## Решение задачи анализа тональности текста на основе модели "мешка слов"

С использованием кросс-валидации попробуем применить к корпусу текстов различные варианты векторизации и классификации.

In [39]:

```
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, amazon_df['text'], amazon_df['value'], scoring='accur
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('===========================')
```

## Классификаторы

По варианту используем классификаторы "LogisticRegression" и "Multinomial Naive Bayes - MNB"

In [40]:

```
vectorizers_list = [CountVectorizer(vocabulary = corpusVocab), TfidfVectorizer(vocabulary = corpusVo
classifiers_list = [LogisticRegression(C=3.0), MultinomialNB(alpha=0.5)]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '07': 2,
'10': 3, '100': 4,
                            '100x': 5, '11': 6, '1100sf': 7, '12': 8, '129': 9,
                            '12am': 10, '15': 11, '150': 12, '18': 13, '19': 14,
                            '1964': 15, '1990': 16, '1gb': 17, '1rst': 18,
                            '1st': 19, '20': 20, '200': 21, '2000': 22,
                            '2017': 23, '229': 24, '23': 25, '24': 26, '25': 27,
                            '29': 28, '2nd': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.9257142857142857
===========================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '07': 2,
'10': 3, '100': 4,
                            '100x': 5, '11': 6, '1100sf': 7, '12': 8, '129': 9,
                            '12am': 10, '15': 11, '150': 12, '18': 13, '19': 14,
                            '1964': 15, '1990': 16, '1gb': 17, '1rst': 18,
                            '1st': 19, '20': 20, '200': 21, '2000': 22,
                            '2017': 23, '229': 24, '23': 25, '24': 26, '25': 27,
                            '29': 28, '2nd': 29, ...})
Модель для классификации - MultinomialNB(alpha=0.5)
Accuracy = 0.9219047619047619
===========================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '07': 2,
'10': 3, '100': 4,
                            '100x': 5, '11': 6, '1100sf': 7, '12': 8, '129': 9,
                            '12am': 10, '15': 11, '150': 12, '18': 13, '19': 14,
                            '1964': 15, '1990': 16, '1gb': 17, '1rst': 18,
                            '1st': 19, '20': 20, '200': 21, '2000': 22,
                            '2017': 23, '229': 24, '23': 25, '24': 26, '25': 27,
                            '29': 28, '2nd': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.921904761904762
===========================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '07': 2,
'10': 3, '100': 4,
                            '100x': 5, '11': 6, '1100sf': 7, '12': 8, '129': 9,
                            '12am': 10, '15': 11, '150': 12, '18': 13, '19': 14,
                            '1964': 15, '1990': 16, '1gb': 17, '1rst': 18,
                            '1st': 19, '20': 20, '200': 21, '2000': 22,
                            '2017': 23, '229': 24, '23': 25, '24': 26, '25': 27,
                            '29': 28, '2nd': 29, ...})
Модель для классификации - MultinomialNB(alpha=0.5)
Accuracy = 0.9168253968253968
===========================
```

## Разделим выборку на обучающую и тестовую и проверим решение для лучшей модели

In [41]:

```
X_train, X_test, y_train, y_test = train_test_split(amazon_df['text'], amazon_df['value'], test_size
```

In [42]:

```
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 [45]:

```
sentiment(TfidfVectorizer(), LogisticRegression(C=3.0))
```

```
Метка      Accuracy
0        0.1171875
1        0.9986178299930891
```

In [46]:

```
sentiment(CountVectorizer(), LogisticRegression(C=3.0))
```

```
Метка      Accuracy
0        0.4375
1        0.9854872149274361
```

In [47]:

```
sentiment(TfidfVectorizer(), MultinomialNB(alpha=0.5))
```

```
Метка      Accuracy
0        0.0078125
1        0.9993089149965446
```

In [48]:

```
sentiment(CountVectorizer(), MultinomialNB(alpha=0.5))
```

```
Метка      Accuracy
0        0.3671875
1        0.9847961299239807
```

Лучшую точность показал CountVectorizer и LogisticRegression

In [ ]:

```

```