Московский государственный технический университет им. Н.Э.

Баумана Кафедра «Системы обработки информации и управления»

Лабораторная работа №5

по дисциплине

«Методы машинного обучения»

на тему

«Предобработка и классификация текстовых данных»

Выполнил:

студент группы ИУ5И-22М

Чжан Аньци

Москва — 2022 г.

In [1]:

```
text='Николай Эрнестович Бауман родился в семье владельца о
```

Задание1:Для произвольного предложения или текста решите следующие задачи:

Токенизация. Частеречная разметка. Лемматизация. Выделение (распознавание) именованных сущностей. Разбор предложения.

1. Токенизация

Для выполнения работы испоьзована библиотека 'Natasha'

In [2]:

```
from razdel import tokenize, sentenize
n_tok_text = list(tokenize(text))
n_tok_text
```

Out[2]:

```
[Substring(0, 1, 'Н'),
 Substring(1, 5, 'и к о л'),
 Substring(5, 6, 'а'),
 Substring(6, 7, 'й'),
 Substring(8, 11, 'Э р н'),
 Substring(11, 12, 'е'),
 Substring(12, 18, 'с т о в и ч'),
 Substring(19, 20, 'Б'),
 Substring(20, 21, 'а'),
 Substring(21, 23, 'у м'),
 Substring(23, 24, 'а'),
 Substring(24, 25, 'н'),
 Substring(26, 33, 'р о д и л с я'),
 Substring(34, 35, 'в'),
 Substring(36, 41, 'с е м ь е'),
 Substring(42, 51, 'в л а д е л ь ц а'),
 Substring(52, 59, 'о б о й н о й'),
 Substring(60, 61, 'и'),
 Substring(62, 71, 'с т о л я р н о й'),
 Substring(72, 82, 'м а с т е р с к о й'),
 Substring(82, 83, '.'),
 Substring(84, 85, 'В'),
 Substring(86, 95, '1891—1895'),
 Substring(96, 101, 'г о д а х'),
 Substring(102, 105, 'б ы л'),
 Substring(106, 115, 'с т у д е н т о м'),
 Substring(116, 126, 'К а з а н с к о г о'),
 Substring(127, 140, 'в е т е р и н а р н о г о'),
 Substring(141, 150, 'и н с т и т у т а'),
 Substring(150, 151, '.'),
 Substring(152, 153, 'В'),
 Substring(154, 158, 'г о д ы'),
 Substring(159, 164, 'у ч ё б ы'),
 Substring(165, 172, 'у в л ё к с я'),
 Substring(173, 184, 'н е л е г а л ь н о й'),
 Substring(185, 198, 'н а р о д н и ч е с к о й'),
 Substring(199, 200, 'и'),
 Substring(201, 213, 'м а р к с и с т с к о й'),
 Substring(214, 225, 'л и т е р а т у р о й'),
 Substring(225, 226, ','),
 Substring(227, 237, 'у ч а с т в о в а л'),
 Substring(238, 239, 'в'),
 Substring(240, 246, 'р а б о т е'),
 Substring(247, 257, 'п о д п о л ь н ы х'),
 Substring(258, 265, 'р а б о ч и х'),
 Substring(266, 273, 'к р у ж к о в'),
 Substring(273, 274, '.')]
```

In [3]:

```
[_.text for _ in n_tok_text]
```

Out[3]:

```
['Н',
 'и к о л',
 'а',
 'й',
 'Э р н',
 'е',
 'с т о в и ч',
 'Б',
 'а',
 'у м',
 'а',
 'н',
 'р о д и л с я',
 'в',
 'с е м ь е',
 'в л а д е л ь ц а',
 'о б о й н о й',
 'и',
 'с т о л я р н о й',
 'м а с т е р с к о й',
 '.',
 'В',
 '1891—1895',
 'г о д а х',
 'б ы л',
 'с т у д е н т о м',
 'К а з а н с к о г о',
 'в е т е р и н а р н о г о',
 'и н с т и т у т а',
 '.',
 'В',
 'г о д ы',
 'у ч ё б ы',
 'у в л ё к с я',
 'н е л е г а л ь н о й',
 'н а р о д н и ч е с к о й',
 'и',
 'м а р к с и с т с к о й',
 'л и т е р а т у р о й',
 ',',
 'у ч а с т в о в а л',
 'в',
 'р а б о т е',
 'п о д п о л ь н ы х',
 'р а б о ч и х',
 'к р у ж к о в',
 '.']
```

In [4]:

```
n_sen_text = list(sentenize(text))
n_sen_text
```

Out[4]:

```
[Substring(0,
           83,
           'Николай Эрнестович Бауман родился в семье
владельца обойной и столярной мастерской.'),
 Substring(84,
           151,
           'В 1891—1895 годах был студентом\xa0Казанског
о ветеринарного института.'),
 Substring(152,
           274,
           'В годы учёбы увлёкся нелегальной народ
нической и марксистской литературой, участво
вал в работе подпольных рабочих кружков.')]
```

In [5]:

```
[_.text for _ in n_sen_text], len([_.text for _ in n_sen_text])
```

Out[5]:

```
(['Николай Эрнестович Бауман родился в семье влад
ельца обойной и столярной мастерской.',
  'В 1891—1895 годах был студентом\xa0Казанского вет
еринарного института.',
  'В годы учёбы увлёкся нелегальной народничес
кой и марксистской литературой, участвовал в
работе подпольных рабочих кружков.'],
 3)
```

In [6]:

```
def n_sentenize(text):
    n_sen_chunk = []
    for sent in sentenize(text):
        tokens = [_.text for _ in tokenize(sent.text)]
        n_sen_chunk.append(tokens)
    return n_sen_chunk
```

In [7]:

```
n_sen_chunk = n_sentenize(text)
n_sen_chunk
```

Out[7]:

```
[['Н',
  'и к о л',
  'а',
  'й',
  'Э р н',
  'е',
  'с т о в и ч',
  'Б',
  'а',
  'у м',
  'а',
  'н',
  'р о д и л с я',
  'в',
  'с е м ь е',
  'в л а д е л ь ц а',
  'о б о й н о й',
  'и',
  'с т о л я р н о й',
  'м а с т е р с к о й',
  '.'],
 ['В',
  '1891—1895',
  'г о д а х',
  'б ы л',
  'с т у д е н т о м',
  'К а з а н с к о г о',
  'в е т е р и н а р н о г о',
  'и н с т и т у т а',
  '.'],
 ['В',
  'г о д ы',
  'у ч ё б ы',
  'у в л ё к с я',
  'н е л е г а л ь н о й',
  'н а р о д н и ч е с к о й',
  'и',
  'м а р к с и с т с к о й',
  'л и т е р а т у р о й',
  ',',
  'у ч а с т в о в а л',
  'в',
  'р а б о т е',
  'п о д п о л ь н ы х',
  'р а б о ч и х',
  'к р у ж к о в',
  '.']]
```

2.Частеречная разметка

In [8]:

```
from navec import Navec
from slovnet import Morph
```

In [9]:

```
navec = Navec.load('navec_news_v1_1B_250K_300d_100q.tar')
```

In [10]:

```
n_morph = Morph.load('slovnet_morph_news_v1.tar', batch_size=4)
```

In [11]:

```
morph_res = n_morph.navec(navec)
```

In [12]:

```
def print_pos(markup):
    for token in markup.tokens:
        print('{} - {}'.format(token.text, token.tag))
```

In [13]:

```
n_text_markup = list(_ for _ in n_morph.map(n_sen_chunk))
[print_pos(x) for x in n_text_markup]
```

```
Н - NOUN
и к о л - X|Foreign=Yes
а - X|Foreign=Yes
й - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
Э р н - PROPN|Foreign=Yes
е - X|Foreign=Yes
с т о в и ч - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
Б - PROPN
а - X|Foreign=Yes
у м - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
а - NOUN
н - X|Foreign=Yes
р о д и л с я - VERB|Aspect=Perf|Gender=Masc|Mood=Ind|Number=Sing|Tense=Past|VerbFo
rm=Fin|Voice=Mid
в - ADP
с е м ь е - NOUN|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
в л а д е л ь ц а - NOUN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
о б о й н о й - NOUN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
и - CCONJ
с т о л я р н о й - ADJ|Case=Gen|Degree=Pos|Gender=Fem|Number=Sing
м а с т е р с к о й - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
. - PUNCT
В - ADP
1891—1895 - ADJ
г о д а х - NOUN|Animacy=Inan|Case=Loc|Gender=Masc|Number=Plur
б ы л - AUX|Aspect=Imp|Gender=Masc|Mood=Ind|Number=Sing|Tense=Past|VerbForm=Fin|Voi
ce=Act
с т у д е н т о м - NOUN|Animacy=Anim|Case=Ins|Gender=Masc|Number=Sing
К а з а н с к о г о - ADJ|Case=Gen|Degree=Pos|Gender=Masc|Number=Sing
в е т е р и н а р н о г о - ADJ|Case=Gen|Degree=Pos|Gender=Masc|Number=Sing
и н с т и т у т а - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
. - PUNCT
В - ADP
г о д ы - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Plur
у ч ё б ы - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
у в л ё к с я - VERB|Aspect=Perf|Gender=Masc|Mood=Ind|Number=Sing|Tense=Past|VerbFo
rm=Fin|Voice=Mid
н е л е г а л ь н о й - ADJ|Case=Ins|Degree=Pos|Gender=Fem|Number=Sing
н а р о д н и ч е с к о й - NOUN|Animacy=Inan|Case=Ins|Gender=Fem|Number=Sing
и - CCONJ
м а р к с и с т с к о й - ADJ|Case=Ins|Degree=Pos|Gender=Fem|Number=Sing
л и т е р а т у р о й - NOUN|Animacy=Inan|Case=Ins|Gender=Fem|Number=Sing
, - PUNCT
у ч а с т в о в а л - VERB|Aspect=Imp|Gender=Masc|Mood=Ind|Number=Sing|Tense=Past|V
erbForm=Fin|Voice=Act
в - ADP
р а б о т е - NOUN|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
п о д п о л ь н ы х - ADJ|Case=Gen|Degree=Pos|Number=Plur
р а б о ч и х - ADJ|Case=Gen|Degree=Pos|Number=Plur
к р у ж к о в - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Plur
. - PUNCT
```

Out[13]:

```
[None, None, None]
```

3.Лемматизация

In [14]:

```
from natasha import Doc, Segmenter, NewsEmbedding, NewsMorphTagger, MorphVocab
```

In [15]:

```
def n_lemmatize(text):
    emb = NewsEmbedding()
    morph_tagger = NewsMorphTagger(emb)
    segmenter = Segmenter()
    morph_vocab = MorphVocab()
    doc = Doc(text)
    doc.segment(segmenter)
    doc.tag_morph(morph_tagger)
    for token in doc.tokens:
        token.lemmatize(morph_vocab)
    return doc
```

In [16]:

```
n_doc = n_lemmatize(text)
{_.text: _.lemma for _ in n_doc.tokens}
```

Out[16]:

```
{'Н': 'h',
 'икол': 'икол',
 'а': 'а',
 'й': 'й',
 'Эрн': 'эрна',
 'е': 'е',
 'стович': 'стович',
 'Б': 'б',
 'ум': 'ум',
 'н': 'н',
 'родился': 'родиться',
 'в': 'в',
 'семье': 'семья',
 'владельца': 'владелец',
 'обойной': 'обойный',
 'и': 'и',
 'столярной': 'столярный',
 'мастерской': 'мастерская',
 '.': '.',
 'В': 'в',
 '1891—1895': '1891—1895',
 'годах': 'год',
 'был': 'быть',
 'студентом': 'студент',
 'Казанского': 'казанский',
 'ветеринарного': 'ветеринарный',
 'института': 'институт',
 'годы': 'год',
 'учёбы': 'учеба',
 'увлёкся': 'увлечься',
 'нелегальной': 'нелегальный',
 'народнической': 'народнический',
 'марксистской': 'марксистский',
 'литературой': 'литература',
 ',': ',',
 'участвовал': 'участвовать',
 'работе': 'работа',
 'подпольных': 'подпольный',
 'рабочих': 'рабочий',
 'кружков': 'кружок'}
```

4.Выделение (распознавание) именованных сущностей

named-entity recognition (NER)

In [17]:

```
from slovnet import NER
from ipymarkup import show_span_ascii_markup as show_markup
```

In [18]:

```
ner = NER.load('slovnet_ner_news_v1.tar')
```

In [19]:

```
ner_res = ner.navec(navec)
```

In [20]:

```
markup_ner = ner(text)
markup_ner
```

Out[20]:

```
SpanMarkup(
    text='Николай Эрнестович Бауман родился в семье
владельца обойной и столярной мастерской. В 189
1—1895 годах был студентом\xa0Казанского ветерин
арного института. В годы учёбы увлёкся нелега
льной народнической и марксистской литератур
ой, участвовал в работе подпольных рабочих кр
ужков.',
    spans=[Span(
         start=116,
         stop=150,
         type='ORG'
     )]
)
```

In [21]:

```
show_markup(markup_ner.text, markup_ner.spans)
```

```
Николай Эрнестович Бауман родился в семье владе
льца обойной и
столярной мастерской. В 1891—1895 годах был студе
нтом Казанского
                                                ORG————————
ветеринарного института. В годы учёбы увлёкся
нелегальной
————————————————————————————————————————————————
народнической и марксистской литературой, уч
аствовал в работе
подпольных рабочих кружков.
```

5.Разбор предложения

In [22]:

```
from natasha import NewsSyntaxParser
```

In [23]:

```
emb = NewsEmbedding()
syntax_parser = NewsSyntaxParser(emb)
```

In [24]:

```
n_doc.parse_syntax(syntax_parser)
n_doc.sents[0].syntax.print()
```

```
┌──────────────────────► Н          advmod
│  ┌►┌──┌──┌──┌──┌──── и к о л      obl
│  │ │  │  │  │  └►а          appos
│  │ │  │  │  └►┌── й          appos
│  │ │  │  │    └► Э р н       nmod
│  │ │  │  │  ┌►е             appos
│  │ │  └►┌──└── с т о в и ч  appos
│  │ │  ┌──└►┌── Б            appos
│  │ │  │    └►а             appos
│  │ └►│  ┌── у м            appos
│  │   └──►│  а             flat:foreign
│  │       └► н             appos
└──└──┌──────┌──────── р о д и л с я
   │  │  ┌►в             case
   │  └►└── с е м ь е     obl
   │  ┌──└►┌── в л а д е л ь ц а  nmod
   │  │  ┌──└► о б о й н о й    nmod
   │  │  │  ┌►и             cc
   │  │  └►└── с т о л я р н о й  conj
   │  └──────► м а с т е р с к о й nmod
   └─────────► .             punct
```

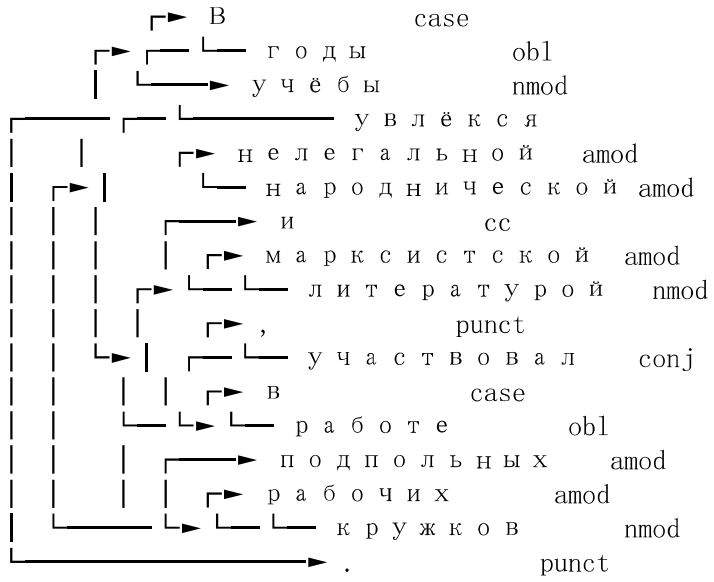
In [25]:

```
n_doc.sents[1].syntax.print()
```

```
      ┌──► В                case
      │  ┌► 1891—1895        amod
   ┌► └──└── г о д а х          nmod
   │  ┌► б ы л               cop
┌──└──└──── с т у д е н т о м
│  │  ┌──► К а з а н с к о г о    amod
│  │  │  ┌► в е т е р и н а р н о г о amod
│  └─►└──└── и н с т и т у т а     nmod
└────────► .                punct
```

In [26]:

```
n_doc.sents[2].syntax.print()
```



Задание2: Для произвольного набора данных, предназначенного для классификации текстов, решите задачу классификации текста двумя способами:

Способ 1. На основе CountVectorizer или TfidfVectorizer. Способ 2. На основе моделей word2vec или Glove или fastText. Сравните качество полученных моделей. Для поиска наборов данных в поисковой системе можно

In [27]:

```
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, median
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
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")
```

In [28]:

```
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 [29]:

```
dataset = pd.read_csv("news_articles.csv")
dataset.head()
```

Out[29]:

	author	published	title	text	language	site_url	
0	Barracuda Brigade	2016-10-26T21:41:00.000+03:00	muslims busted they stole millions in govt ben...	print they should pay all the back all the mon...	english	100percentfedup.com	cor
1	reasoning with facts	2016-10-29T08:47:11.259+03:00	re why did attorney general loretta lynch plea...	why did attorney general loretta lynch plead t...	english	100percentfedup.com	cor
2	Barracuda Brigade	2016-10-31T01:41:49.479+02:00	breaking weiner cooperating with fbi on hillar...	red state \nfox news sunday reported this mor...	english	100percentfedup.com	cor
3	Fed Up	2016-11-01T05:22:00.000+02:00	pin drop speech by father of daughter kidnappe...	email kayla mueller was a prisoner and torture...	english	100percentfedup.com	http:/
4	Fed Up	2016-11-01T21:56:00.000+02:00	fantastic trumps point plan to reform healthc...	email healthcare reform to make america great ...	english	100percentfedup.com	http:/

In [30]:

```
dataset1=dataset[['text','hasImage']]
dataset1.head()
```

Out[30]:

	text	hasImage
0	print they should pay all the back all the mon...	1.0
1	why did attorney general loretta lynch plead t...	1.0
2	red state \nfox news sunday reported this mor...	1.0
3	email kayla mueller was a prisoner and torture...	1.0
4	email healthcare reform to make america great ...	1.0

In [31]:

```
dataset1=dataset1.dropna()
dataset1.head()
```

Out[31]:

	text	hasImage
0	print they should pay all the back all the mon...	1.0
1	why did attorney general loretta lynch plead t...	1.0
2	red state \nfox news sunday reported this mor...	1.0
3	email kayla mueller was a prisoner and torture...	1.0
4	email healthcare reform to make america great ...	1.0

In [32]:

```
dataset1=dataset1.sample(frac=1)
dataset1.head()
```

Out[32]:

	text	hasImage
1397	by sarah jones on sat oct th at pm donald tr...	1.0
492	email \n\nhillary clinton personally ordered a...	1.0
237	by vin armani hillary clinton continues to bla...	0.0
1178	ignoring the law to rig elections democrats ar...	1.0
516	email president barack obama admonished donald...	1.0

In [33]:

```
dataset1.describe()
```

Out[33]:

	hasImage
count	2050.000000
mean	0.772683
std	0.419201
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	1.000000

In [34]:

```
train_df=(dataset1.iloc[0:1500,:])
test_df=(dataset1.iloc[1500:2050,:])
```

In [35]:

```
train_df.describe()
```

Out[35]:

	hasImage
count	1500.000000
mean	0.779333
std	0.414834
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	1.000000

In [36]:

```
test_df.describe()
```

Out[36]:

	hasImage
count	550.000000
mean	0.754545
std	0.430748
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	1.000000

Способ 1. На основе CountVectorizer или TfidfVectorizer.

Сформируем общий словарь для обучения моделей из обучающей и тестовой выборки

In [37]:

```
vocab_list = train_df['text'].tolist()
vocab_list[1:10]
```

```
ong as they benefit infowars nightly news  october  comments \nimmigration laws
are being ignored to an unprecedented extent \nthe democrats are worried about ru
ssia interfering with our elections but are ignoring the law being broken and dom
estic interference via illegal immigrants newsletter sign up get the latest break
ing news  specials from alex jones and the infowars crew related articles downloa
d on your mobile device now for free today on the show get the latest breaking ne
ws  specials from alex jones and the infowars crew from the store expert trump ha
s already won election  see the rest on the alex jones youtube channel  the most
offensive halloween ever  see the rest on the alex jones youtube channel  illustr
ation how much will your healthcare premiums rise in  infowarscom is a free sp
eech systems llc company all rights reserved digital millennium copyright act not
ice  flip the switch and supercharge your state of mind with brain force the nex
t generation of neural activation from infowars life httpwwwinfowarscomwpcontentu
ploadsbrainforceejpg httpwwwinfowarsstorecomhealthandwellnessinfowarslifebrainfor
cehtmlimstzrwuutm_campaigninfowarsplacementutm_sourceinfowarscomutm_mediumwidgetu
tm_contentbrainforce httpwwwinfowarsstorecomhealthandwellnessinfowarslifebrainfor
cehtmlimstzrwuutm_campaigninfowarsplacementutm_sourceinfowarscomutm_mediumwidgetu
tm_contentbrainforce brain force  off  flip the switch and supercharge your sta
te of mind with brain force the next generation of neural activation from infowar
s life httpwwwinfowarscomwpcontentuploadsbrainforceejpg httpwwwinfowarsstorecomhe
```

In [38]:

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

In [39]:

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

```
Количество сформированных признаков - 40455
```

In [40]:

```
import warnings
warnings.filterwarnings("ignore")

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

```
Векторизация - CountVectorizer(vocabulary={'___': 0, 'aa': 1, 'aab': 2,
'aadmi': 3, 'aafe': 4,
                               'aah': 5, 'aaja': 6, 'aali': 7, 'aam': 8, 'aand': 9,
                               'aap': 10, 'aaps': 11, 'aaron': 12, 'aas': 13,
                               'ab': 14, 'aba': 15, 'aback': 16, 'abajo': 17,
                               'abandon': 18, 'abandoned': 19, 'abandoning': 20,
                               'abandonment': 21, 'abandons': 22, 'abated': 23,
                               'abb': 24, 'abbas': 25, 'abbasside': 26,
                               'abbekommen': 27, 'abbey': 28, 'abbott': 29, ...})
Модель для классификации - LogisticRegression()
Accuracy = 0.7893333333333334
============================
Векторизация - CountVectorizer(vocabulary={'___': 0, 'aa': 1, 'aab': 2,
'aadmi': 3, 'aafe': 4,
                               'aah': 5, 'aaja': 6, 'aali': 7, 'aam': 8, 'aand': 9,
                               'aap': 10, 'aaps': 11, 'aaron': 12, 'aas': 13,
                               'ab': 14, 'aba': 15, 'aback': 16, 'abajo': 17,
                               'abandon': 18, 'abandoned': 19, 'abandoning': 20,
                               'abandonment': 21, 'abandons': 22, 'abated': 23,
                               'abb': 24, 'abbas': 25, 'abbasside': 26,
                               'abbekommen': 27, 'abbey': 28, 'abbott': 29, ...})
Модель для классификации - MultinomialNB()
Accuracy = 0.7766666666666667
============================
Векторизация - TfidfVectorizer(vocabulary={'___': 0, 'aa': 1, 'aab': 2,
'aadmi': 3, 'aafe': 4,
                               'aah': 5, 'aaja': 6, 'aali': 7, 'aam': 8, 'aand': 9,
                               'aap': 10, 'aaps': 11, 'aaron': 12, 'aas': 13,
                               'ab': 14, 'aba': 15, 'aback': 16, 'abajo': 17,
                               'abandon': 18, 'abandoned': 19, 'abandoning': 20,
                               'abandonment': 21, 'abandons': 22, 'abated': 23,
                               'abb': 24, 'abbas': 25, 'abbasside': 26,
                               'abbekommen': 27, 'abbey': 28, 'abbott': 29, ...})
Модель для классификации - LogisticRegression()
Accuracy = 0.7840000000000001
============================
Векторизация - TfidfVectorizer(vocabulary={'___': 0, 'aa': 1, 'aab': 2,
'aadmi': 3, 'aafe': 4,
                               'aah': 5, 'aaja': 6, 'aali': 7, 'aam': 8, 'aand': 9,
                               'aap': 10, 'aaps': 11, 'aaron': 12, 'aas': 13,
                               'ab': 14, 'aba': 15, 'aback': 16, 'abajo': 17,
                               'abandon': 18, 'abandoned': 19, 'abandoning': 20,
                               'abandonment': 21, 'abandons': 22, 'abated': 23,
                               'abb': 24, 'abbas': 25, 'abbasside': 26,
                               'abbekommen': 27, 'abbey': 28, 'abbott': 29, ...})
Модель для классификации - MultinomialNB()
Accuracy = 0.7793333333333333
============================
```

Лучшую точность показал CountVectorizer и LogisticRegression(78.93%)

In [41]:

```
X_train=train_df['text']
y_train=train_df['hasImage']
X_test=test_df['text']
y_test=test_df['hasImage']
```

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 [43]:

```
sentiment(CountVectorizer(), LogisticRegression(C=5.0))
```

```
Метка       Accuracy
0.0      0.5259259259259259
1.0      0.9132530120481928
```

Способ 2. На основе моделей word2vec

In [44]:

```
import re
import pandas as pd
import numpy as np
from typing import Dict, Tuple
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from nltk import WordPunctTokenizer
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\12391\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[44]:

```
True
```

Подготовим корпус

In [45]:

```
corpus = []
stop_words = stopwords.words('english')
tok = WordPunctTokenizer()
for line in dataset1['text'].values:
    line1 = line.strip().lower()
    line1 = re.sub("[^a-zA-Z]"," ", line1)
    text_tok = tok.tokenize(line1)
    text_tok1 = [w for w in text_tok if not w in stop_words]
    corpus.append(text_tok1)
```

In [46]:

```
corpus[:5]
```

```
 'left',
 'spurn',
 'newly',
 'surfaced',
 'video',
 'via',
 'ryan',
 'grim',
 'shows',
 'trump',
 'humiliation',
 'game',
 'action',
 'walk',
 'everyone',
 'pov',
 'woman',
 'share',
 'twitter',
 'print',
```

Количество текстов в корпусе не изменилось и соответствует целевому признаку

In [47]:

```
assert dataset1.shape[0]==len(corpus)
```

In [48]:

```
import gensim
from gensim.models import word2vec
%time model = word2vec.Word2Vec(corpus, workers=4, min_count=10, window=10, sample=1e-3)
```

```
CPU times: total: 6.91 s
Wall time: 2.08 s
```

Проверим, что модель обучилась

In [49]:

```
print(model.wv.most_similar(positive=['find'], topn=5))
```

```
[('good', 0.9726372361183167), ('little', 0.9723168611526489), ('always', 0.97155606
74667358), ('whole', 0.9670679569244385), ('understand', 0.9665523767471313)]
```

In [50]:

```
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 [51]:

```
class EmbeddingVectorizer(object):
    def __init__(self, model):
        self.model = model
        self.size = model.vector_size

    def fit(self, X, y):
        return self

    def transform(self, X):
        return np.array([np.mean(
            [self.model[w] for w in words if w in self.model]
            or [np.zeros(self.size)], axis=0)
            for words in X])
```

In [52]:

```
boundary = 1500
X_train = corpus[:boundary]
X_test = corpus[boundary:]
y_train = dataset1['hasImage'][:boundary]
y_test = dataset1['hasImage'][boundary:]
```

In [53]:

```
sentiment(EmbeddingVectorizer(model.wv), LogisticRegression(C=5.0))
```

```
Метка        Accuracy
0.0     0.08888888888888889
1.0     0.980722891566265
```

Лучшую точность показал word2vec

источник базы данных - https://www.kaggle.com/datasets/ruchi798/source-based-news-classification (https://www.kaggle.com/datasets/ruchi798/source-based-news-classification)