

МГТУ им. Н. Э. Баумана, кафедра ИУ5
курс “Методы машинного обучения”

Лабораторная работа №5

«Предобработка текста»

ВЫПОЛНИЛ:

Сергеев М.К.

Группа: ИУ5-22М

ПРОВЕРИЛ:

Гапанюк Ю.Е.

Москва 2022

Задание:

1. Для произвольного предложения или текста решите следующие задачи:
 - Токенизация.
 - Частеречная разметка.
 - Лемматизация.
 - Выделение (распознавание) именованных сущностей.
 - Разбор предложения.
2. 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 использовать ключевые слова "datasets for text classification".

3. Сформировать отчет и разместить его в своем репозитории на github.

Выполнение работы:

```
In [51]: text = '''С другой стороны социально-экономическое развитие влечет за собой процесс внедрения и модернизации мод
Разнообразный и богатый опыт начало повседневной работы по формированию позиции представляет собой интересный эк
Повседневная практика показывает, что новая модель организационной деятельности играет важную роль в формировани
text2 = 'Россия или Российская Федерация – государство в Восточной Европе и Северной Азии со столицей в городе М
```

```
In [16]: !pip install natasha
```

```
Collecting natasha
  Downloading natasha-1.4.0-py3-none-any.whl (34.4 MB)
    |██████████████████████████████████████| 34.4 MB 127 kB/s
Collecting razdel>=0.5.0
  Downloading razdel-0.5.0-py3-none-any.whl (21 kB)
Collecting yargy>=0.14.0
  Downloading yargy-0.15.0-py3-none-any.whl (41 kB)
    |██████████████████████████████████████| 41 kB 112 kB/s
Collecting navec>=0.9.0
  Downloading navec-0.10.0-py3-none-any.whl (23 kB)
Collecting ipymarkup>=0.8.0
  Downloading ipymarkup-0.9.0-py3-none-any.whl (14 kB)
Collecting pymorphy2
  Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
    |██████████████████████████████████████| 55 kB 3.9 MB/s
Collecting slovnet>=0.3.0
  Downloading slovnet-0.5.0-py3-none-any.whl (49 kB)
    |██████████████████████████████████████| 49 kB 6.1 MB/s
Collecting intervaltree>=3
  Downloading intervaltree-3.1.0.tar.gz (32 kB)
Requirement already satisfied: sortedcontainers<3.0,>=2.0 in /usr/local/lib/python3.7/dist-packages (from intervaltree>=3->ipymarkup>=0.8.0->natasha) (2.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from navec>=0.9.0->natasha) (1.21.6)
Collecting dawg-python>=0.7.1
  Downloading DAWG_Python-0.7.2-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: docopt>=0.6 in /usr/local/lib/python3.7/dist-packages (from pymorphy2->natasha) (0.6.2)
Collecting pymorphy2-dicts-ru<3.0,>=2.4
  Downloading pymorphy2_dicts_ru-2.4.417127.4579844-py2.py3-none-any.whl (8.2 MB)
    |██████████████████████████████████████| 8.2 MB 50.1 MB/s
Building wheels for collected packages: intervaltree
  Building wheel for intervaltree (setup.py) ... done
  Created wheel for intervaltree: filename=intervaltree-3.1.0-py2.py3-none-any.whl size=26119 sha256=12f684935551c0e3e98f66864a833cf57193f206ddd0d931fd92a934424edca2
  Stored in directory: /root/.cache/pip/wheels/16/85/bd/1001cbb46dcfb71c2001cd7401c6fb250392f22a81ce3722f7
Successfully built intervaltree
Installing collected packages: pymorphy2-dicts-ru, dawg-python, razdel, pymorphy2, navec, intervaltree, yargy, slovnet, ipymarkup, natasha
  Attempting uninstall: intervaltree
    Found existing installation: intervaltree 2.1.0
    Uninstalling intervaltree-2.1.0:
      Successfully uninstalled intervaltree-2.1.0
Successfully installed dawg-python-0.7.2 intervaltree-3.1.0 ipymarkup-0.9.0 natasha-1.4.0 navec-0.10.0 pymorphy2-0.9.1 pymorphy2-dicts-ru-2.4.417127.4579844 razdel-0.5.0 slovnet-0.5.0 yargy-0.15.0
```

Задача токенизации

```
In [18]: from razdel import tokenize, sentenize
```

```
In [19]: n_tok_text = list(tokenize(text))
n_tok_text
```

```
Out[19]: [Substring(0, 1, 'С'),
Substring(2, 8, 'другой'),
Substring(9, 16, 'сторонн'),
Substring(17, 40, 'социально-экономическое'),
Substring(41, 49, 'развитие'),
Substring(50, 56, 'влечет'),
Substring(57, 59, 'за'),
Substring(60, 65, 'собой'),
Substring(66, 73, 'процесс'),
Substring(74, 83, 'внедрения'),
Substring(84, 85, 'и'),
Substring(86, 98, 'модернизации'),
Substring(99, 105, 'модели'),
Substring(106, 114, 'развития'),
Substring(114, 115, '.'),
Substring(117, 130, 'Разнообразный'),
Substring(131, 132, 'и'),
Substring(133, 140, 'богатый'),
```

```

Substring(141, 145, 'опыт'),
Substring(146, 152, 'начало'),
Substring(153, 165, 'повседневной'),
Substring(166, 172, 'работы'),
Substring(173, 175, 'по'),
Substring(176, 188, 'формированию'),
Substring(189, 196, 'позиции'),
Substring(197, 209, 'представляет'),
Substring(210, 215, 'собой'),
Substring(216, 226, 'интересный'),
Substring(227, 238, 'эксперимент'),
Substring(239, 247, 'проверки'),
Substring(248, 259, 'направлений'),
Substring(260, 274, 'прогрессивного'),
Substring(275, 283, 'развития'),
Substring(283, 284, '.'),
Substring(286, 298, 'Повседневная'),
Substring(299, 307, 'практика'),
Substring(308, 318, 'показывает'),
Substring(318, 319, ','),
Substring(320, 323, 'что'),
Substring(324, 329, 'новая'),
Substring(330, 336, 'модель'),
Substring(337, 352, 'организационной'),
Substring(353, 365, 'деятельности'),
Substring(366, 372, 'играет'),
Substring(373, 379, 'важную'),
Substring(380, 384, 'роль'),
Substring(385, 386, 'в'),
Substring(387, 399, 'формировании'),
Substring(400, 406, 'модели'),
Substring(407, 415, 'развития'),
Substring(415, 416, '.') ]

```

```

In [21]: [_.text for _ in n_tok_text]

```

```

Out[21]: ['С',
'другой',
'стороны',
'социально-экономическое',
'развитие',
'влечет',
'за',
'собой',
'процесс',
'внедрения',
'и',
'модернизации',
'модели',
'развития',
'.',
'Разнообразный',
'и',
'богатый',
'опыт',
'начало',
'повседневной',
'работы',
'по',
'формированию',
'позиции',
'представляет',
'собой',
'интересный',
'эксперимент',
'проверки',
'направлений',
'прогрессивного',
'развития',
'.',
'Повседневная',
'практика',
'показывает',
',',
'что',
'новая',
'модель',
'организационной',
'деятельности',
'играет',
'важную',
'роль',
'в',
'формировании',
'модели',

```

```
'развития',  
'.']
```

```
In [22]: n_sen_text = list(sentenize(text))  
n_sen_text
```

```
Out[22]: [Substring(0,  
115,  
'С другой стороны социально-экономическое развитие влечет за собой процесс внедрения и модернизации мо  
дели развития.'),  
Substring(117,  
284,  
'Разнообразный и богатый опыт начало повседневной работы по формированию позиции представляет собой ин  
тересный эксперимент проверки направлений прогрессивного развития.'),  
Substring(286,  
416,  
'Повседневная практика показывает, что новая модель организационной деятельности играет важную роль в  
формировании модели развития.')] ]
```

```
In [23]: [_.text for _ in n_sen_text], len([_.text for _ in n_sen_text])
```

```
Out[23]: (['С другой стороны социально-экономическое развитие влечет за собой процесс внедрения и модернизации модели разв  
ития.',  
'Разнообразный и богатый опыт начало повседневной работы по формированию позиции представляет собой интересный  
эксперимент проверки направлений прогрессивного развития.',  
'Повседневная практика показывает, что новая модель организационной деятельности играет важную роль в формирова  
нии модели развития.'],  
3)
```

```
In [24]: # Этот вариант токенизации нужен для последующей обработки  
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 [25]: n_sen_chunk = n_sentenize(text)  
n_sen_chunk
```

```
Out[25]: [['С',  
'другой',  
'сторон',  
'социально-экономическое',  
'развитие',  
'влечет',  
'за',  
'собой',  
'процесс',  
'внедрения',  
'и',  
'модернизации',  
'модели',  
'развития',  
'.'],  
['Разнообразный',  
'и',  
'богатый',  
'опыт',  
'начало',  
'повседневной',  
'работы',  
'по',  
'формированию',  
'позиции',  
'представляет',  
'собой',  
'интересный',  
'эксперимент',  
'проверки',  
'направлений',  
'прогрессивного',  
'развития',  
'.'],  
['Повседневная',  
'практика',  
'показывает',
```

```

',',
'что',
'новая',
'модель',
'организационной',
'деятельности',
'играет',
'важную',
'роль',
'в',
'формировании',
'модели',
'развития',
'.']]

```

```

In [59]: n_sen_chunk_2 = n_sentenize(text2)
n_sen_chunk_2

```

```

Out[59]: [['Россия',
'или',
'Российская',
'Федерация',
'—',
'государство',
'в',
'Восточной',
'Европе',
'и',
'Северной',
'Азии',
'со',
'столицей',
'в',
'городе',
'Москва',
'.']]

```

Частеречная разметка

```

In [26]: from navec import Navec
from slovnet import Morph

```

```

In [29]: # Файл необходимо скачать по ссылке https://github.com/natasha/navec#downloads
navec = Navec.load('navec_news_v1_1B_250K_300d_100q.tar')

```

```

In [30]: # Файл необходимо скачать по ссылке https://github.com/natasha/slovnet#downloads
n_morph = Morph.load('slovnet_morph_news_v1.tar', batch_size=4)

```

```

In [31]: morph_res = n_morph.navec(navec)

```

```

In [32]: def print_pos(markup):
    for token in markup.tokens:
        print('{} - {}'.format(token.text, token.tag))

```

```

In [33]: n_text_markup = list(_ for _ in n_morph.map(n_sen_chunk))
[print_pos(x) for x in n_text_markup]

```

```

C - ADP
другой - ADJ|Case=Gen|Degree=Pos|Gender=Fem|Number=Sing
стороны - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
социально-экономическое - ADJ|Case=Nom|Degree=Pos|Gender=Neut|Number=Sing
развитие - NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
влечет - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
за - ADP
собой - PRON|Case=Ins
процесс - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing
внедрения - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
и - CCONJ
модернизации - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
модели - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
развития - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
. - PUNCT
Разнообразный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing

```

и - CONJ
 богатый - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
 опыт - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
 начало - NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
 повседневной - ADJ|Case=Gen|Degree=Pos|Gender=Fem|Number=Sing
 работы - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
 по - ADP
 формированию - NOUN|Animacy=Inan|Case=Dat|Gender=Neut|Number=Sing
 позиции - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
 представляет - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
 собой - PRON|Case=Ins
 интересный - ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Gender=Masc|Number=Sing
 эксперимент - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing
 проверки - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
 направлений - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Plur
 прогрессивного - ADJ|Case=Gen|Degree=Pos|Gender=Neut|Number=Sing
 развития - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
 . - PUNCT
 Повседневная - ADJ|Case=Nom|Degree=Pos|Gender=Fem|Number=Sing
 практика - NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing
 показывает - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
 , - PUNCT
 что - CONJ
 новая - ADJ|Case=Nom|Degree=Pos|Gender=Fem|Number=Sing
 модель - NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing
 организационной - ADJ|Case=Gen|Degree=Pos|Gender=Fem|Number=Sing
 деятельности - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
 играет - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
 важную - ADJ|Case=Acc|Degree=Pos|Gender=Fem|Number=Sing
 роль - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Sing
 в - ADP
 формировании - NOUN|Animacy=Inan|Case=Loc|Gender=Neut|Number=Sing
 модели - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
 развития - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
 . - PUNCT
 Out[33]: [None, None, None]

```
In [60]: n_text2_markup = list(n_morph.map(n_sen_chunk_2))
[print_pos(x) for x in n_text2_markup]
```

Россия - PROPN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing
 или - CONJ
 Российская - ADJ|Case=Nom|Degree=Pos|Gender=Fem|Number=Sing
 Федерация - PROPN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing
 — - PUNCT
 государство - NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
 в - ADP
 Восточной - ADJ|Case=Loc|Degree=Pos|Gender=Fem|Number=Sing
 Европе - PROPN|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
 и - CONJ
 Северной - ADJ|Case=Gen|Degree=Pos|Gender=Fem|Number=Sing
 Азии - PROPN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
 со - ADP
 столицей - NOUN|Animacy=Inan|Case=Ins|Gender=Fem|Number=Sing
 в - ADP
 городе - NOUN|Animacy=Inan|Case=Loc|Gender=Masc|Number=Sing
 Москва - PROPN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing
 . - PUNCT

Out[60]: [None]

Лемматизация

```
In [34]: from natasha import Doc, Segmenter, NewsEmbedding, NewsMorphTagger, MorphVocab
```

```
In [35]: 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 [36]: n_doc = n_lemmatize(text)
{_.text: _.lemma for _ in n_doc.tokens}
```

```
Out[36]: {'.': '.',
'.': '.',
'Повседневная': 'повседневный',
'Разнообразный': 'разнообразный',
'С': 'с',
'богатый': 'богатый',
'в': 'в',
'важную': 'важный',
'влечет': 'влечь',
'внедрения': 'внедрение',
'деятельности': 'деятельность',
'другой': 'другой',
'за': 'за',
'и': 'и',
'играет': 'играть',
'интересный': 'интересный',
'модели': 'модель',
'модель': 'модель',
'модернизации': 'модернизация',
'направлений': 'направление',
'начало': 'начало',
'новая': 'новый',
'опыт': 'опыт',
'организационной': 'организационный',
'по': 'по',
'повседневной': 'повседневный',
'позиции': 'позиция',
'показывает': 'показывать',
'практика': 'практика',
'представляет': 'представлять',
'проверки': 'проверка',
'прогрессивного': 'прогрессивный',
'процесс': 'процесс',
'работы': 'работа',
'развитие': 'развитие',
'развития': 'развитие',
'роль': 'роль',
'собой': 'себя',
'социально-экономическое': 'социально-экономический',
'стороны': 'сторона',
'формировании': 'формирование',
'формированию': 'формирование',
'что': 'что',
'эксперимент': 'эксперимент'}
```

```
In [61]: n_doc2 = n_lemmatize(text2)
{_.text: _.lemma for _ in n_doc2.tokens}
```

```
Out[61]: {'.': '.',
'Азии': 'азия',
'Восточной': 'восточный',
'Европе': 'европа',
'Москва': 'москва',
'Российская': 'российский',
'Россия': 'россия',
'Северной': 'северный',
'Федерация': 'федерация',
'в': 'в',
'городе': 'город',
'государство': 'государство',
'и': 'и',
'или': 'или',
'со': 'с',
'столицей': 'столица',
'—': '—'}
```

Выделение (распознавание) именованных сущностей

```
In [47]: from slovnet import NER
from ipymarkup import show_span_ascii_markup as show_markup
```

```
In [48]: ner = NER.load('slovnet_ner_news_v1.tar')
```



```
In [49]: ner_res = ner.navec(navec)
```

```
In [52]: markup_ner = ner(text2)
markup_ner
```

```
Out[52]: SpanMarkup(
  text='Россия или Российская Федерация – государство в Восточной Европе и Северной Азии со столицей в городе М
осква.',
  spans=[Span(
    start=0,
    stop=6,
    type='LOC'
  ), Span(
    start=11,
    stop=31,
    type='LOC'
  ), Span(
    start=48,
    stop=64,
    type='LOC'
  ), Span(
    start=67,
    stop=80,
    type='LOC'
  ), Span(
    start=102,
    stop=108,
    type='LOC'
  )]
)
```

```
In [53]: show_markup(markup_ner.text, markup_ner.spans)
```

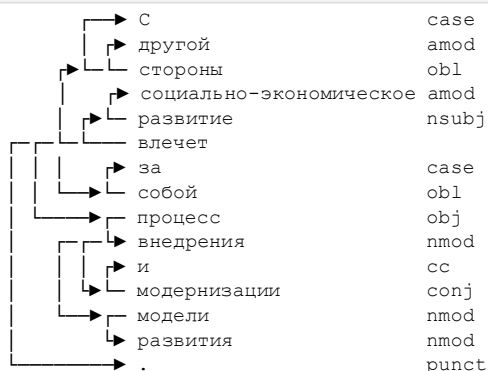
Россия или Российская Федерация – государство в Восточной Европе и
LOC— LOC— LOC— LOC—
Северной Азии со столицей в городе Москва.
LOC— LOC—

Разбор предложения

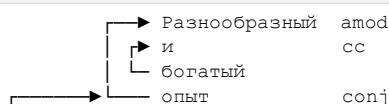
```
In [54]: from natasha import NewsSyntaxParser
```

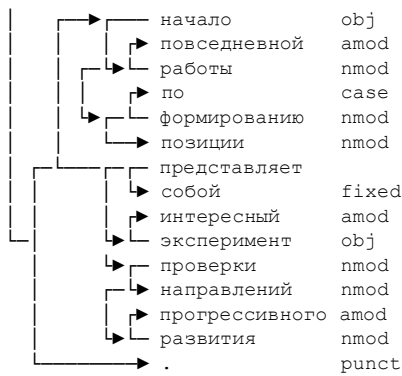
```
In [55]: emb = NewsEmbedding()
syntax_parser = NewsSyntaxParser(emb)
```

```
In [56]: n_doc.parse_syntax(syntax_parser)
n_doc.sents[0].syntax.print()
```



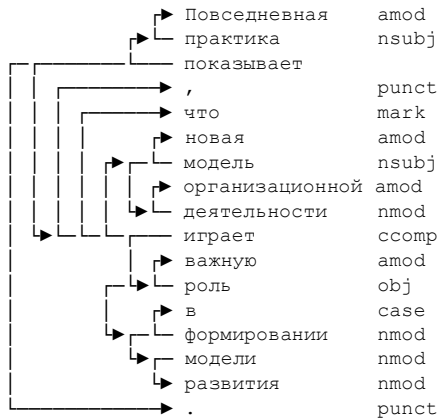
```
In [57]: n_doc.parse_syntax(syntax_parser)
n_doc.sents[1].syntax.print()
```





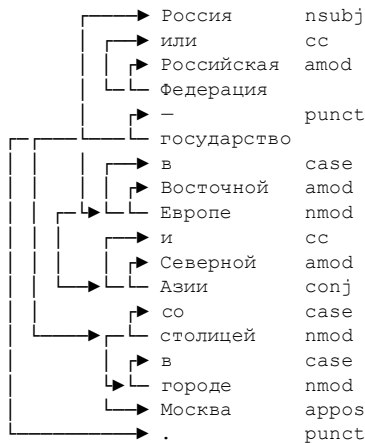
In [58]:

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



In [62]:

```
n_doc2.parse_syntax(syntax_parser)
n_doc2.sents[0].syntax.print()
```



In [84]:

```
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_absolute_err
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
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")
```

Векторизация текста на основе модели "мешка слов"

```
In [65]: categories = ["rec.motorcycles", "rec.sport.baseball", "sci.electronics", "sci.med"]
newsgrps = fetch_20newsgroups(subset='train', categories=categories)
data = newsgroups['data']
```

```
In [66]: 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 [67]: vocabVect = CountVectorizer()
vocabVect.fit(data)
corpusVocab = vocabVect.vocabulary_
print('Количество сформированных признаков - {}'.format(len(corpusVocab)))
```

Количество сформированных признаков - 33448

```
In [68]: for i in list(corpusVocab)[1:10]:
    print('{}={}'.format(i, corpusVocab[i]))
```

```
nrmendel=22213
unix=31462
amherst=5287
edu=12444
nathaniel=21624
mendell=20477
subject=29220
re=25369
bike=6898
```

Использование класса CountVectorizer

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

```
In [71]: test_features.todense()
```

```
Out[71]: matrix([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [2, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]])
```

```
In [72]: # Размер нулевой строки
len(test_features.todense()[0].getA1())
```

```
Out[72]: 33448
```

```
In [74]: # Непустые значения нулевой строки
print([i for i in test_features.todense()[0].getA1() if i>0])
```

```
[1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2]
```

```
In [77]: vocabVect.get_feature_names()[0:10]
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out
instead.
  warnings.warn(msg, category=FutureWarning)
```

```
Out[77]: ['00',
          '000',
          '0000',
          '00000000004',
          '00000000005',
          '00000000667',
          '0000001200',
          '0001',
          '00014',
          '0002']
```

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

```
In [79]: 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, newsgroups['data'], newsgroups['target'], scoring='accuracy', cv=
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('=====')
```

```
In [82]: vectorizers_list = [CountVectorizer(vocabulary = corpusVocab)]
classifiers_list = [LogisticRegression(C=3.0), LinearSVC(), KNeighborsClassifier()]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
  https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '0000000004': 3,
                                         '0000000005': 4, '0000000667': 5, '0000001200': 6,
                                         '0001': 7, '00014': 8, '0002': 9, '0003': 10,
                                         '0005111312': 11, '0005111312nalem': 12,
                                         '00072': 13, '000851': 14, '000rpm': 15,
                                         '000th': 16, '001': 17, '0010': 18, '001004': 19,
                                         '0011': 20, '001211': 21, '0013': 22, '001642': 23,
                                         '001813': 24, '002': 25, '002222': 26,
                                         '002251w': 27, '0023': 28, '002937': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.937813339432037
=====
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '0000000004': 3,
                                         '0000000005': 4, '0000000667': 5, '0000001200': 6,
                                         '0001': 7, '00014': 8, '0002': 9, '0003': 10,
                                         '0005111312': 11, '0005111312nalem': 12,
                                         '00072': 13, '000851': 14, '000rpm': 15,
                                         '000th': 16, '001': 17, '0010': 18, '001004': 19,
                                         '0011': 20, '001211': 21, '0013': 22, '001642': 23,
                                         '001813': 24, '002': 25, '002222': 26,
                                         '002251w': 27, '0023': 28, '002937': 29, ...})
Модель для классификации - LinearSVC()
Accuracy = 0.9453742497059174
=====
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '0000000004': 3,
                                         '0000000005': 4, '0000000667': 5, '0000001200': 6,
                                         '0001': 7, '00014': 8, '0002': 9, '0003': 10,
                                         '0005111312': 11, '0005111312nalem': 12,
                                         '00072': 13, '000851': 14, '000rpm': 15,
                                         '000th': 16, '001': 17, '0010': 18, '001004': 19,
                                         '0011': 20, '001211': 21, '0013': 22, '001642': 23,
                                         '001813': 24, '002': 25, '002222': 26,
                                         '002251w': 27, '0023': 28, '002937': 29, ...})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.6655358653541747
=====

```

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

```
In [85]: X_train, X_test, y_train, y_test = train_test_split(newsgroups['data'], newsgroups['target'], test_size=0.5, ran
```

```
In [86]: 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 [88]: sentiment(CountVectorizer(), LinearSVC())
```

Метка	Accuracy
0	0.9290322580645162
1	0.9675090252707581
2	0.9026845637583892
3	0.9245901639344263

Работа с векторными представлениями слов с использованием word2vec

```
In [96]: import gensim
from gensim.models import word2vec
```

```
In [93]: model_path = 'ruscorpora_mystem_cbow_300_2_2015.bin.gz'
```

```
In [94]: model = gensim.models.KeyedVectors.load_word2vec_format(model_path, binary=True)
```

```
In [108]: words = ['холод_S', 'мороз_S', 'береза_S', 'сосна_S']
```

```
In [109]: for word in words:
    if word in model:
        print('\nСЛОВО - {}'.format(word))
        print('5 ближайших соседей слова:')
        for word, sim in model.most_similar(positive=[word], topn=5):
            print('{} => {}'.format(word, sim))
    else:
        print('Слово "{}" не найдено в модели'.format(word))
```

```
СЛОВО - холод_S
5 ближайших соседей слова:
стужа_S => 0.7676383852958679
сырость_S => 0.6338975429534912
жара_S => 0.6089427471160889
мороз_S => 0.5890367031097412
озноб_S => 0.5776054859161377
```

```
СЛОВО - мороз_S
5 ближайших соседей слова:
стужа_S => 0.6425479650497437
морозец_S => 0.5947279930114746
холод_S => 0.5890367031097412
жара_S => 0.5522176623344421
снегопад_S => 0.5083199143409729
```

```
СЛОВО - береза_S
5 ближайших соседей слова:
сосна_S => 0.7943247556686401
тополь_S => 0.7562226057052612
дуб_S => 0.7440178394317627
дерево_S => 0.7373415231704712
клен_S => 0.7105200290679932
```

```
СЛОВО - сосна_S
5 ближайших соседей слова:
береза_S => 0.7943247556686401
дерево_S => 0.7581434845924377
лиственница_S => 0.747814953327179
дуб_S => 0.7412480711936951
ель_S => 0.7363824248313904
```

Находим близость между словами и строим аналогии

```
In [110]: print(model.similarity('сосна_S', 'береза_S'))
```

```
0.7943247
```

```
In [111]: print(model.most_similar(positive=['холод_S', 'стужа_S'], negative=['мороз_S']))
```

```
[('сырость_S', 0.5040211081504822), ('стылость_S', 0.46336129307746887), ('голод_S', 0.4604816436767578), ('зной_S', 0.45904627442359924), ('скука_S', 0.4489358067512512), ('жара_S', 0.44645121693611145), ('усталость_S', 0.4218570291996002), ('озноб_S', 0.41469818353652954), ('духота_S', 0.4099087715148926), ('неуют_S', 0.40298789739608765)]
```

Обучим word2vec на наборе данных "fetch_20newsgroups"

In [112]

```
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 /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[112] True

In []:

```
categories = ["rec.motorcycles", "rec.sport.baseball", "sci.electronics", "sci.med"]
newsgroups = fetch_20newsgroups(subset='train', categories=categories)
data = newsgroups['data']
```

In [120]

```
# Подготовим корпус
corpus = []
stop_words = stopwords.words('english')
tok = WordPunctTokenizer()
for line in newsgroups['data']:
    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 [121]

```
corpus[:5]
```

Out[121]

```
[['nrmendel',
  'unix',
  'amherst',
  'edu',
  'nathaniel',
  'mendell',
  'subject',
  'bike',
  'advice',
  'organization',
  'amherst',
  'college',
  'x',
  'newsreader',
  'tin',
  'version',
  'pl',
  'lines',
  'ummm',
  'bikes',
  'kx',
  'suggest',
  'look',
  'zx',
  'since',
  'horsepower',
  'whereas',
  'might',
  'bit',
  'much',
  'sincerely',
  'nathaniel',
  'zx',
  'dod',
  'ama'],
 ['grante',
  'aquarius',
  'rosemount',
  'com',
  'grant',
  'edwards',
```

'subject',
'krillean',
'photography',
'reply',
'grante',
'aquarius',
'rosemount',
'com',
'grant',
'edwards',
'organization',
'rosemount',
'inc',
'lines',
'nntp',
'posting',
'host',
'aquarius',
'stgprao',
'st',
'unocal',
'com',
'richard',
'ottolini',
'writes',
'living',
'things',
'maintain',
'small',
'electric',
'fields',
'enhance',
'certain',
'chemical',
'reactions',
'promote',
'communication',
'states',
'cell',
'communicate',
'cells',
'nervous',
'system',
'specialized',
'example',
'perhaps',
'uses',
'true',
'electric',
'fields',
'change',
'location',
'time',
'large',
'organism',
'also',
'true',
'special',
'photographic',
'techniques',
'applying',
'external',
'fields',
'kirillian',
'photography',
'interact',
'fields',
'resistances',
'caused',
'fields',
'make',
'interesting',
'pictures',
'really',
'kirlian',
'photography',
'taking',
'pictures',
'corona',
'discharge',
'objects',
'animate',
'inanimate',
'fields',
'applied',
'objects',
'millions',
'times',
'larger',

'biologically',
'created',
'fields',
'want',
'record',
'biologically',
'created',
'electric',
'fields',
'got',
'use',
'low',
'noise',
'high',
'gain',
'sensors',
'typical',
'eegs',
'ekgs',
'kirlian',
'photography',
'phun',
'physics',
'type',
'stuff',
'right',
'soaking',
'chunks',
'extra',
'fine',
'steel',
'wool',
'liquid',
'oxygen',
'hitting',
'hammer',
'like',
'kirlean',
'setup',
'fun',
'possibly',
'dangerous',
'perhaps',
'pictures',
'diagonistic',
'disease',
'problems',
'organisms',
'better',
'understood',
'perhaps',
'probably',
'grant',
'edwards',
'yow',
'vote',
'rosemount',
'inc',
'well',
'tapered',
'half',
'cocked',
'ill',
'conceived',
'grante',
'aquarius',
'rosemount',
'com',
'tax',
'deferred'],
['liny',
'sun',
'scri',
'fsu',
'edu',
'nemo',
'subject',
'bates',
'method',
'myopia',
'reply',
'lin',
'ray',
'met',
'fsu',
'edu',
'distribution',
'na',
'organization',

'scri',
'florida',
'state',
'university',
'lines',
'bates',
'method',
'work',
'first',
'heard',
'newsgroup',
'several',
'years',
'ago',
'got',
'hold',
'book',
'improve',
'sight',
'simple',
'daily',
'drills',
'relaxation',
'margaret',
'corbett',
'authorized',
'instructor',
'bates',
'method',
'published',
'talks',
'vision',
'improvement',
'relaxation',
'exercise',
'study',
'whether',
'method',
'actually',
'works',
'works',
'actually',
'shortening',
'previously',
'elongated',
'eyeball',
'increasing',
'lens',
'ability',
'flatten',
'order',
'compensate',
'long',
'eyeball',
'since',
'myopia',
'result',
'eyeball',
'elongation',
'seems',
'logical',
'approach',
'correction',
'find',
'way',
'reverse',
'process',
'e',
'shorten',
'somehow',
'preferably',
'non',
'surgically',
'recent',
'studies',
'find',
'know',
'rk',
'works',
'changing',
'curvature',
'cornea',
'compensate',
'shape',
'eyeball',
'way',
'train',
'muscles',
'shorten',

'eyeball',
'back',
'correct',
'length',
'would',
'even',
'better',
'bates',
'idea',
'right',
'thanks',
'information'],
['mcovingt',
'aisun',
'ai',
'uga',
'edu',
'michael',
'covington',
'subject',
'buy',
'parts',
'time',
'nntp',
'posting',
'host',
'aisun',
'ai',
'uga',
'edu',
'organization',
'ai',
'programs',
'university',
'georgia',
'athens',
'lines',
'pricing',
'parts',
'reminds',
'something',
'chemist',
'said',
'gram',
'dye',
'costs',
'dollar',
'comes',
'liter',
'jar',
'also',
'costs',
'dollar',
'want',
'whole',
'barrel',
'also',
'costs',
'dollar',
'e',
'charge',
'almost',
'exclusively',
'packaging',
'delivering',
'chemical',
'particular',
'case',
'byproduct',
'cost',
'almost',
'nothing',
'intrinsically',
'michael',
'covington',
'associate',
'research',
'scientist',
'artificial',
'intelligence',
'programs',
'mcovingt',
'ai',
'uga',
'edu',
'university',
'georgia',
'phone',
'athens',

'georgia',
'u',
'amateur',
'radio',
'n',
'tmi'],
['tammy',
'vandenboom',
'launchpad',
'unc',
'edu',
'tammy',
'vandenboom',
'subject',
'sore',
'spot',
'testicles',
'nntp',
'posting',
'host',
'lambada',
'oit',
'unc',
'edu',
'organization',
'university',
'north',
'carolina',
'extended',
'bulletin',
'board',
'service',
'distribution',
'na',
'lines',
'husband',
'woke',
'three',
'days',
'ago',
'small',
'sore',
'spot',
'spot',
'size',
'nickel',
'one',
'testicles',
'bottom',
'side',
'knots',
'lumps',
'little',
'sore',
'spot',
'says',
'reminds',
'bruise',
'feels',
'recollection',
'hitting',
'anything',
'like',
'would',
'cause',
'bruise',
'assures',
'remember',
'something',
'like',
'clues',
'might',
'somewhat',
'hypochondriac',
'sp',
'sure',
'gonna',
'die',
'thanks',
'opinions',
'expressed',
'necessarily',
'university',
'north',
'carolina',
'chapel',
'hill',
'campus',
'office',

```
'information',
'technology',
'experimental',
'bulletin',
'board',
'service',
'internet',
'launchpad',
'unc',
'edu']]
```

In [123]

```
%time model_imdb = word2vec.Word2Vec(corpus, workers=4, min_count=10, window=10, sample=1e-3)
```

```
CPU times: user 5.76 s, sys: 34.9 ms, total: 5.79 s
Wall time: 3.61 s
```

In [124]

```
# Проверим, что модель обучилась
print(model_imdb.wv.most_similar(positive=['find'], topn=5))
```

```
[('work', 0.992060661315918), ('voltage', 0.9903362989425659), ('using', 0.9889512658119202), ('high', 0.9884157180786133), ('circuits', 0.9863718748092651)]
```

In [132]

```
def sentiment_2(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)
```

Проверка качества работы модели word2vec

In [127]

```
class EmbeddingVectorizer(object):
    """
    Для текста усредним вектора входящих в него слов
    """
    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 [128]

```
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_dataflt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_dataflt['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 [136..

```

# Обучающая и тестовая выборки
boundary = 1500
X_train = corpus[:boundary]
X_test = corpus[boundary:]
y_train = newsgroups['target'][:boundary]
y_test = newsgroups['target'][boundary:]

```

In [137..

```

sentiment_2(EmbeddingVectorizer(model_imdb.wv), LogisticRegression(C=5.0))

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
 Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
 extra_warning_msg= LOGISTIC_SOLVER_CONVERGENCE_MSG,

Метка	Accuracy
0	0.8552631578947368
1	0.9320388349514563
2	0.7568807339449541
3	0.7368421052631579

Как видно из результатов проверки качества моделей, лучшее качество показал CountVectorizer