МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №__5__
по дисциплине «Методы машинного обучения»

ИСПОЛНИТЕЛЬ: _Морозенков О.Н___
ФИО
группа ИУ5-23М _____
подпись
"__"_____2022 г.

ПРЕПОДАВАТЕЛЬ: _Гапанюк Ю.Е._____
ФИО
_____
подпись
"__"_____2022 г.

Москва - 2022

Цель лабораторной работы: изучение методов предобработки и классификации текстовых данных.

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".

```
text = '''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.'''
text2 = 'Россия или Российская Федерация — государство в Восточной Европе и Северной Азии со столицей в городе Москва.'
```

Задача токенизации

```
from razdel import tokenize, sentenize

n_tok_text = list(tokenize(text))
n_tok_text
```

```
[Substring(0, 1, 'С'),
 Substring(2, 8, 'другой'),
 Substring(9, 16, 'стороны'),
 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, '.')]
```

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

```
['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',
 '.']
```

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

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

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

```
(['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.'],
 3)

# 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

n_sen_chunk = n_sentenize(text)
n_sen_chunk
```

```
[['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',
  '.']]
```

```
n_sen_chunk_2 = n_sentenize(text2)
n_sen_chunk_2
```

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

Частеречная разметка

```
from navec import Navec
from slovnet import Morph

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

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

```
morph_res = n_morph.navec(navec)

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

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

```
С - 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=A
ct
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 - CCONJ
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=A
ct
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=A
ct
, - PUNCT
что - SCONJ
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=A
ct
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

[None, None, None]
```

```
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 - CCONJ
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 - CCONJ
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

[None]
```

Лемматизация

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

```
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

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

```
{'С': 'с',
 'другой': 'другой',
 'стороны': 'сторона',
 'социально-экономическое': 'социально-экономический',
 'развитие': 'развитие',
 'влечет': 'влечь',
 'за': 'за',
 'собой': 'себя',
 'процесс': 'процесс',
 'внедрения': 'внедрение',
 'и': 'и',
 'модернизации': 'модернизация',
 'модели': 'модель',
 'развития': 'развитие',
 '.': '.',
 'Разнообразный': 'разнообразный',
 'богатый': 'богатый',
 'опыт': 'опыт',
 'начало': 'начало',
 'повседневной': 'повседневный',
 'работы': 'работа',
 'по': 'по',
 'формированию': 'формирование',
 'позиции': 'позиция',
 'представляет': 'представлять',
 'интересный': 'интересный',
 'эксперимент': 'эксперимент',
 'проверки': 'проверка',
 'направлений': 'направление',
 'прогрессивного': 'прогрессивный',
 'Повседневная': 'повседневный',
 'практика': 'практика',
 'показывает': 'показывать',
 ',': ',',
 'что': 'что',
```

```
 'новая': 'новый',
 'модель': 'модель',
 'организационной': 'организационный',
 'деятельности': 'деятельность',
 'играет': 'играть',
 'важную': 'важный',
 'роль': 'роль',
 'в': 'в',
 'формировании': 'формирование'}
```

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

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

Выделение (распознавание) именованных сущностей

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

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

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

```
markup_ner = ner(text2)
markup_ner
```

```
SpanMarkup(
    text='Россия или Российская Федерация — государство в Восточной Европе и
Северной Азии со столицей в городе Москва.',
    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'
        )]
)

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

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

Разбор предложения

```
from natasha import NewsSyntaxParser

emb = NewsEmbedding()
syntax_parser = NewsSyntaxParser(emb)

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

```
        ┌─► С                         case
        │  ┌► другой                  amod
     ┌─►└──└─ стороны                 obl
     │     ┌► социально-экономическое amod
     │  ┌─►└─ развитие                nsubj
┌─┬─┬─└──┴─── влечет
│ │ │  ┌─►┌─ за                       case
│ │ │  └──►└─ собой                   obl
│ │ └─────►┌─ процесс                  obj
│ │     ┌─┬└─ внедрения               nmod
│ │     │ │  ┌► и                     cc
│ │     │ └─►└─ модернизации          conj
```

```
    └─► ┌─ модели                  nmod
        └─► развития               nmod
└─────────► .                      punct
```

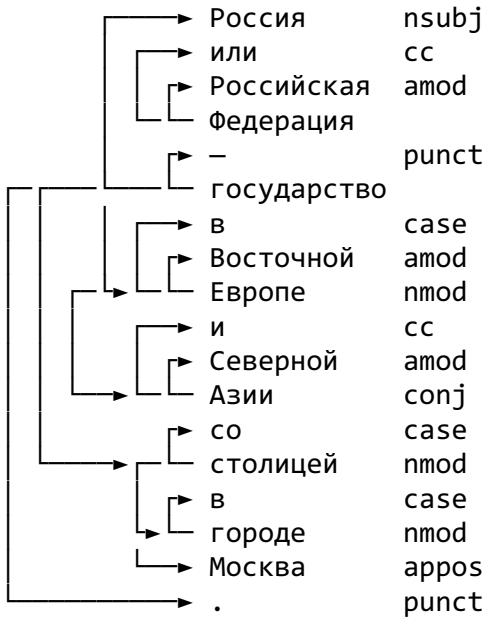
```
n_doc.parse_syntax(syntax_parser)
n_doc.sents[1].syntax.print()
```

```
            ┌─► Разнообразный  amod
            ├─► и              cc
            └─ богатый
┌─────────► └── опыт           conj
│   ┌─────► ┌── начало         obj
│   │       ├─► повседневной  amod
│   │     ┌─└─ работы         nmod
│   │     │ ┌─► по             case
│   │     └─└─ формированию   nmod
│   │       └─► позиции        nmod
│ ┌─└─────┌── представляет
│ │       ├─► собой           fixed
│ │       ├─► интересный      amod
└─└─────► └── эксперимент     obj
          └─► ┌─ проверки      nmod
            ┌─└─ направлений   nmod
            │ ┌─► прогрессивного amod
            └─└─ развития      nmod
┌─────────────► .              punct
```

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

```
              ┌─► Повседневная    amod
            ┌─└─ практика         nsubj
┌─┌─────────└──── показывает
│ │ ┌─────────► ,                punct
│ │ │ ┌───────► что              mark
│ │ │ │       ┌─► новая          amod
│ │ │ │     ┌─└─ модель          nsubj
│ │ │ │     │ ┌─► организационной amod
│ │ │ │     └─└─ деятельности    nmod
│ └─└─└─────┌── играет           ccomp
│           ├─► важную           amod
│         ┌─└─ роль              obj
│         │ ┌─► в                case
│       ┌─└─└── формировании     nmod
│       │   ┌─► модели           nmod
│       │   └─ развития         nmod
└─────────────► .                punct
```

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




```
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_error, r2_score
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"

```
categories = ["rec.motorcycles", "rec.sport.baseball",
"sci.electronics","sci.med"]
```

```
newsgroups = fetch_20newsgroups(subset='train', categories=categories)
data = newsgroups['data']

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]))

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

```
Количество сформированных признаков - 33448
```

```
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

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

```
<2380x33448 sparse matrix of type '<class 'numpy.int64'>'
      with 335176 stored elements in Compressed Sparse Row format>
```

```
test_features.todense()
```

```
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]])
```

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

```
33448
```

```
# 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, 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, 3, 2]
```

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

```
/home/dnikolskiy/anaconda3/lib/python3.9/site-
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)
```

```
['00',
 '000',
 '0000',
 '0000000004',
 '0000000005',
 '0000000667',
 '0000001200',
 '0001',
 '00014',
 '0002']
```

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

```
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=3).mean()
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')

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

```
/home/dnikolskiy/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: 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
  n_iter_i = _check_optimize_result(
/home/dnikolskiy/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: 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
  n_iter_i = _check_optimize_result(
/home/dnikolskiy/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: 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
  n_iter_i = _check_optimize_result(

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, '0005111312na1em': 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 = 0.9382336841146768
============================
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, '0005111312na1em': 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, '0005111312na1em': 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

```
X_train, X_test, y_train, y_test = train_test_split(newsgroups['data'],
newsgroups['target'], test_size=0.5, random_state=1)

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)

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

```
import gensim
from gensim.models import word2vec

model_path = 'ruscorpora_mystem_cbow_300_2_2015.bin.gz'

model = gensim.models.KeyedVectors.load_word2vec_format(model_path,
binary=True)

words = ['холод_S', 'мороз_S', 'береза_S', 'сосна_S']

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 => 0.589036762714386
озноб_S => 0.5776054859161377

СЛОВО - мороз_S
5 ближайших соседей слова:
стужа_S => 0.6425478458404541
морозец_S => 0.5947279930114746
холод_S => 0.589036762714386
жара_S => 0.5522176623344421
снегопад_S => 0.5083199143409729

СЛОВО - береза_S
5 ближайших соседей слова:
сосна_S => 0.7943246960639954
тополь_S => 0.7562226057052612
дуб_S => 0.7440178394317627
дерево_S => 0.7373415231704712
клен_S => 0.7105200886726379

СЛОВО - сосна_S
5 ближайших соседей слова:
береза_S => 0.7943247556686401
дерево_S => 0.758143424987793
лиственница_S => 0.7478148937225342
дуб_S => 0.7412480711936951
ель_S => 0.7363824844360352
```

#Находим близость между словами и строим аналогии

```
print(model.similarity('сосна_S', 'береза_S'))
```

```
0.79432476
```

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

```
[('сырость_S', 0.5040210485458374), ('стылость_S', 0.46336129307746887),
('голод_S', 0.4604816436767578), ('зной_S', 0.45904630422592163), ('скука_S',
0.4489358067512512), ('жара_S', 0.44645124673843384), ('усталость_S',
0.4218570291996002), ('озноб_S', 0.41469815373420715), ('духота_S',
0.4099087715148926), ('неуют_S', 0.40298786759376526)]
```

#Обучим word2vec на наборе данных "fetch_20newsgroups"

```
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]     /home/dnikolskiy/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
True
```

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

```
# 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)
```

```
corpus[:5]
```

```
[['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',
  'asssures',
  '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']]
```

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

```
CPU times: user 3.76 s, sys: 19.1 ms, total: 3.78 s
Wall time: 1.16 s
```

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

```
[('etc', 0.9759538173675537), ('voltage', 0.9697504639625549), ('buy',
0.9697079658508301), ('circuit', 0.9695694446563721), ('work',
0.9691796898841858)]
```

```
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

```
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])

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]))

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

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

```
/home/dnikolskiy/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: 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
  n_iter_i = _check_optimize_result(
```

```
Метка        Accuracy
0      0.868421052631579
1      0.9368932038834952
2      0.8073394495412844
3      0.7719298245614035
```

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