Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Лабораторная работа №5

«Предобработка и классификация текстовых данных»

по дисциплине

«Методы машинного обучения»

ИСПОЛНИТЕЛЬ:

Крюков Г.М.
Группа ИУ5-21М

_____

"__"_____2022 г.

Москва, 2022

Цель: 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".

Набор данных:

SlovNet is a Python library for deep-learning based NLP modeling for Russian language. Library is integrated with other Natasha projects: Nerus — large automatically annotated corpus, Razdel — sentence segmenter, tokenizer and Navec — compact Russian embeddings.

Текст программы:


```
text = '''С минувшего сезона в Формуле 1 ввели лимит бюджетов, который регулирует затраты команд чемпионата на протяжении вс
Ограничения не включают в себя зарплаты гонщиков и трех главных сотрудников, а также траты на перелеты, отпуска и менеджмент
командам нужно вписываться в денежный «потолок» только по издержкам, связанным с работой над машинами, включая зарплаты пода
text2 = 'Если установить планку максимального дохода пилотов, это может подстегнуть коллективы задуматься о выборе между при
```

```
!pip install natasha
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting natasha
  Downloading natasha-1.4.0-py3-none-any.whl (34.4 MB)
     |████████████████████████████████| 34.4 MB 145 kB/s
Collecting razdel>=0.5.0
  Downloading razdel-0.5.0-py3-none-any.whl (21 kB)
Collecting slovnet>=0.3.0
  Downloading slovnet-0.5.0-py3-none-any.whl (49 kB)
     |████████████████████████████████| 49 kB 3.4 MB/s
Collecting yargy>=0.14.0
  Downloading yargy-0.15.0-py3-none-any.whl (41 kB)
     |████████████████████████████████| 41 kB 97 kB/s
Collecting ipymarkup>=0.8.0
  Downloading ipymarkup-0.9.0-py3-none-any.whl (14 kB)
Collecting navec>=0.9.0
  Downloading navec-0.10.0-py3-none-any.whl (23 kB)
Collecting pymorphy2
  Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
     |████████████████████████████████| 55 kB 3.9 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>
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)
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 42.5 MB/s
Requirement already satisfied: docopt>=0.6 in /usr/local/lib/python3.7/dist-packages (from pymorphy2->natasha) (0.6.2)
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=04f6762984fea3d225
```



```
  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,
  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
```

Задача токенизации

```
from razdel import tokenize, sentenize
```

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

```
 Substring(72, 79, 'затраты'),
 Substring(80, 86, 'команд'),
 Substring(87, 97, 'чемпионата'),
 Substring(98, 100, 'на'),
 Substring(101, 111, 'протяжении'),
 Substring(112, 117, 'всего'),
 Substring(118, 130, 'календарного'),
 Substring(131, 135, 'года'),

 Substring(135, 136, '.'),
 Substring(137, 148, 'Ограничения'),
 Substring(149, 151, 'не'),
 Substring(152, 160, 'включают'),
 Substring(161, 162, 'в'),
 Substring(163, 167, 'себя'),
 Substring(168, 176, 'зарплаты'),
 Substring(177, 185, 'гонщиков'),
 Substring(186, 187, 'и'),
 Substring(188, 192, 'трех'),
 Substring(193, 200, 'главных'),
 Substring(201, 212, 'сотрудников'),
```

```
Substring(212, 213, ' '),
Substring(214, 215, 'а'),
Substring(216, 221, 'также'),
Substring(222, 227, 'траты'),
Substring(228, 230, 'на'),
Substring(231, 239, 'перелеты'),
Substring(239, 240, ','),
Substring(241, 248, 'отпуска'),
Substring(249, 250, 'и'),
Substring(251, 261, 'менеджмент'),
Substring(262, 263, '–'),
Substring(264, 278, 'соответственно'),
Substring(278, 279, ','),
Substring(280, 288, 'командам'),
Substring(289, 294, 'нужно'),
Substring(295, 306, 'вписываться'),
Substring(307, 308, 'в'),
Substring(309, 317, 'денежный'),
Substring(318, 319, '«'),
Substring(319, 326, 'потолок'),
Substring(326, 327, '»'),
Substring(328, 334, 'только'),
Substring(335, 337, 'по'),
Substring(338, 347, 'издержкам'),
Substring(347, 348, ','),
Substring(349, 358, 'связанным'),
Substring(359, 360, 'с'),
Substring(361, 368, 'работой'),
Substring(369, 372, 'над'),
Substring(373, 381, 'машинами'),

Substring(381, 382, ','),
Substring(383, 390, 'включая'),
Substring(391, 399, 'зарплаты'),
Substring(400, 412, 'подавляющего'),
Substring(413, 424, 'большинства'),
Substring(425, 431, 'членов'),
Substring(432, 439, 'команды'),
```

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

```
'регулирует',
'затраты',
```



```
 команд ,
'чемпионата',
'на',
'протяжении',
'всего',
'календарного',
'года',
'.',
'Ограничения',
'не',
'включают',
'в',
'себя',
'зарплаты',
'гонщиков',
'и',
'трех',
'главных',
'сотрудников',
',',
'а',
'также',
'траты',
'на',
'перелеты',
',',
'отпуска',
'и',
'менеджмент',
'–',

'соответственно',
',',
'командам',
'нужно',
'вписываться',
'в',
'денежный',
'«',
'потолок',
'»',
'только',
'по',
```

```
 издержкам ',
',',
'связанным',
'с',
'работой',
'над',
'машинами',
',',
'включая',
'зарплаты',
'подавляющего',
'большинства',
'членов',
'команды',
```

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

```
[Substring(0,
           136,
           'С минувшего сезона в Формуле 1 ввели лимит бюджетов, который регулирует затраты команд чемпионата на протя»
 Substring(137,
           440,
           'Ограничения не включают в себя зарплаты гонщиков и трех главных сотрудников, а также траты на перелеты, отг
```

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

```
(['С минувшего сезона в Формуле 1 ввели лимит бюджетов, который регулирует затраты команд чемпионата на протяжении всег
  'Ограничения не включают в себя зарплаты гонщиков и трех главных сотрудников, а также траты на перелеты, отпуска и ме
 2)
```

```
# 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',
  'минувшего',
  'сезона',
  'в',
  'Формуле',
  '1',
  'ввели',
  'лимит',
  'бюджетов',
  ',',
  'который',
  'регулирует',
  'затраты',
  'команд',
  'чемпионата',
  'на',
  'протяжении',
  'всего',
  'календарного',
  'года',
  '.'],
 ['Ограничения',
  'не',
  'включают',
  'в',
  'себя',
  'зарплаты',
  'гонщиков',
  'и',
  'трех',
  'главных',
  'сотрудников',
  ',',
  'а',
  'также',
  'траты',
```

```
          'на',
          'перелеты',
          ',',
          'отпуска',
          'и',
          'менеджмент',
          '–',
          'соответственно',
          ',',
          'командам',
          'нужно',
          'вписываться',
          'в',
          'денежный',
          '«',
          'потолок',
          '»',
          'только',
          'по',
          'издержкам',
          ',',
          '.          .
```

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

```
     [['Если',
       'установить',
       'планку',
       'максимального',
       'дохода',
       'пилотов',
       ',',
       'это',
       'может',
       'подстегнуть',
       'коллективы',
       'задуматься',
       'о',
       'выборе',
       'между',
       'приглашением',
```



```
          'звезд',
          'спорта',
          'и',
          'дополнительными',
          'тратами',
          'на',
          'развитии',
          'своего',
          'автомобиля',
          '.']]
```

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
     минувшего - ADJ|Case=Gen|Degree=Pos|Gender=Masc|Number=Sing
```

```
сезона - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
в - ADP
Формуле - PROPN|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
1 - NUM
ввели - VERB|Aspect=Perf|Mood=Ind|Number=Plur|Tense=Past|VerbForm=Fin|Voice=Act
лимит - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing
бюджетов - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Plur
, - PUNCT
который - PRON|Case=Nom|Gender=Masc|Number=Sing
регулирует - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
затраты - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Plur
команд - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Plur
чемпионата - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
на - ADP
протяжении - NOUN|Animacy=Inan|Case=Loc|Gender=Neut|Number=Sing
всего - DET|Case=Gen|Gender=Masc|Number=Sing
календарного - ADJ|Case=Gen|Degree=Pos|Gender=Masc|Number=Sing
года - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
. - PUNCT
Ограничения - NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Plur
не - PART|Polarity=Neg
включают - VERB|Aspect=Imp|Mood=Ind|Number=Plur|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
в - ADP
себя - PRON|Case=Acc
зарплаты - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Plur
гонщиков - NOUN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Plur
и - CCONJ
трех - NUM|Animacy=Anim|Case=Acc
главных - ADJ|Case=Gen|Degree=Pos|Number=Plur
сотрудников - NOUN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Plur
, - PUNCT
а - CCONJ
также - ADV|Degree=Pos
траты - NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Plur
на - ADP
перелеты - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Plur
, - PUNCT
отпуска - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Plur
и - CCONJ
менеджмент - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
– - PUNCT
соответственно - ADV|Degree=Pos
```



```
, - PUNCT
командам - NOUN|Animacy=Inan|Case=Dat|Gender=Fem|Number=Plur
нужно - ADJ|Degree=Pos|Gender=Neut|Number=Sing|Variant=Short
вписываться - VERB|Aspect=Imp|VerbForm=Inf|Voice=Mid
в - ADP
денежный - ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Gender=Masc|Number=Sing
« - PUNCT
потолок - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing
» - PUNCT
только - PART
по - ADP
издержкам - NOUN|Animacy=Inan|Case=Dat|Gender=Fem|Number=Plur
, - PUNCT
```

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

```
Если - SCONJ
установить - VERB|Aspect=Perf|VerbForm=Inf|Voice=Act
планку - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Sing
максимального - ADJ|Case=Gen|Degree=Pos|Gender=Masc|Number=Sing
дохода - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
пилотов - NOUN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Plur
, - PUNCT
это - PRON|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
может - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
подстегнуть - VERB|Aspect=Perf|VerbForm=Inf|Voice=Act
коллективы - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Plur
задуматься - VERB|Aspect=Perf|VerbForm=Inf|Voice=Mid
о - ADP
выборе - NOUN|Animacy=Inan|Case=Loc|Gender=Masc|Number=Sing
между - ADP
приглашением - NOUN|Animacy=Inan|Case=Ins|Gender=Neut|Number=Sing
звезд - NOUN|Animacy=Anim|Case=Gen|Gender=Fem|Number=Plur
спорта - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
и - CCONJ
дополнительными - ADJ|Case=Ins|Degree=Pos|Number=Plur
тратами - NOUN|Animacy=Inan|Case=Ins|Gender=Fem|Number=Plur
на - ADP
развитии - NOUN|Animacy=Inan|Case=Loc|Gender=Neut|Number=Sing
своего - DET|Case=Gen|Gender=Masc|Number=Sing
автомобиля - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|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}
```

```
{',': ',',
 '.': '.',
 '1': '1',
 '«': '«',
 '»': '»',
 'Ограничения': 'ограничение',
 'С': 'с',
 'Формуле': 'формула',
 'а': 'а',
 'большинства': 'большинство',
 'бюджетов': 'бюджет',
 'в': 'в',
```



```
 'ввели': 'ввести',
 'включают': 'включать',
 'включая': 'включая',
 'вписываться': 'вписываться',
 'всего': 'весь',
 'главных': 'главный',
 'года': 'год',
 'гонщиков': 'гонщик',
 'денежный': 'денежный',
 'зарплаты': 'зарплата',
 'затраты': 'затрата',
 'и': 'и',
 'издержкам': 'издержка',
 'календарного': 'календарный',
 'команд': 'команда',
 'командам': 'команда',
 'команды': 'команда',
 'который': 'который',
 'лимит': 'лимит',
 'машинами': 'машина',
 'менеджмент': 'менеджмент',
 'минувшего': 'минувший',
 'на': 'на',
 'над': 'над',
 'не': 'не',
 'нужно': 'нужный',
 'отпуска': 'отпуск',
 'перелеты': 'перелет',
 'по': 'по',
 'подавляющего': 'подавлять',
 'потолок': 'потолок',
 'протяжении': 'протяжение',
 'работой': 'работа',
 'регулирует': 'регулировать',
 'с': 'с',
 'связанным': 'связать',
 'себя': 'себя',
 'сезона': 'сезон',
 'соответственно': 'соответственно',
 'сотрудников': 'сотрудник',
 'также': 'также',
 'только': 'только',
```

```
 'траты': 'трата',
 'трех': 'три',
 'чемпионата': 'чемпионат'
```

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

```
{',': ',',
 '.': '.',
 'Если': 'если',
 'автомобиля': 'автомобиль',
 'выборе': 'выбор',
 'дополнительными': 'дополнительный',
 'дохода': 'доход',
 'задуматься': 'задуматься',
 'звезд': 'звезда',
 'и': 'и',
 'коллективы': 'коллектив',
 'максимального': 'максимальный',
 'между': 'между',
 'может': 'мочь',
 'на': 'на',
 'о': 'о',
 'пилотов': 'пилот',
 'планку': 'планка',
 'подстегнуть': 'подстегнуть',
 'приглашением': 'приглашение',
 'развитии': 'развитие',
 'своего': 'свой',
 'спорта': 'спорт',
 'тратами': 'трата',
 'установить': 'установить',
 'это': 'это'}
```

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='Если установить планку максимального дохода пилотов, это может подстегнуть коллективы задуматься о выборе меж
    spans=[]
)
```

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

```
Если установить планку максимального дохода пилотов, это может
подстегнуть коллективы задуматься о выборе между приглашением звезд
спорта и дополнительными тратами на развитии своего автомобиля.
```

Разбор предложения

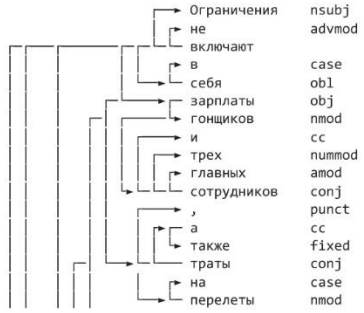
```
from natasha import NewsSyntaxParser
```

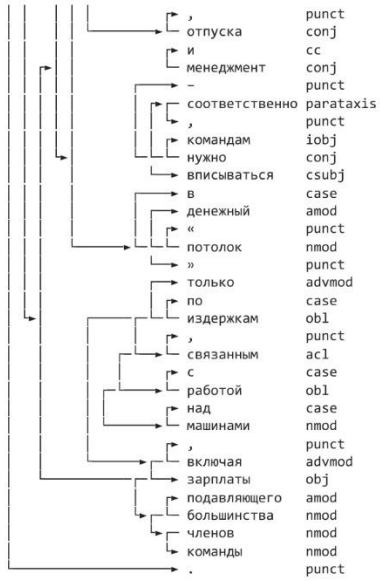
```
emb = NewsEmbedding()
syntax_parser = NewsSyntaxParser(emb)
```

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

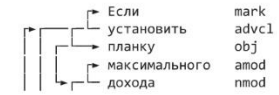
```
            ┌─> C            case
```


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






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



```
    │  │    └──► пилотов        nmod
    │  └─────────► ,            punct
    │      ┌► это              nsubj
 ┌──┴───┌─┌┴─ может
 │      │ └►┌─ подстегнуть     xcomp
 │      │   └► коллективы      obj
 │      └►┌─── задуматься      xcomp
 │        │ ┌► о               case
 │  ┌───┌─└►└─ выборе          obl
 │  │   │   ┌► между           case
 │  │ ┌─└►┌─└─ приглашением    nmod
 │  │ │   └►┌─ звезд           nmod
 │  │ │     └► спорта          nmod
 │  │ │   ┌──► и               cc
 │  │ │   │ ┌► дополнительными amod
 │  │ └──►└─└─ тратами         conj
 │  │       ┌► на              case
 │  └────►┌─└─ развитии        nmod
 │        │ ┌► своего          det
 │        └►└─ автомобиля      nmod
 └──────────► .                punct
```

```
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.autos", "sci.space","comp.windows.x"]
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 - 35770
```

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

```
nicho=22808
vnet=33598
ibm=17519
com=10099
greg=16232
stewart=30046
nicholls=22810
```



```
subject=30313
re=26313
```

Использование класса CountVectorizer

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

```
<2378x35770 sparse matrix of type '<class 'numpy.int64'>'
        with 347775 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],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 1, 0],
        [0, 0, 0, ..., 0, 0, 0]])
```

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

```
35770
```

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

```
[1, 1, 1, 1, 2, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 6, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1
```

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

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is de
```

```
  warnings.warn(msg, category=FutureWarning)
['00',
 '000',
 '0000',
 '00000',
 '000000',
 '00000000',
 '0000000004',
 '0000000005',
 '0000000667',
 '0000001200']
```

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)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to co
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 co
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 co
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, '00000': 3,
                            '000000': 4, '00000000': 5, '0000000004': 6,
                            '0000000005': 7, '0000000667': 8, '0000001200': 9,
                            '00000074': 10, '00000093': 11, '000000e5': 12,
                            '00000315': 13, '000005102000': 14,
                            '00000510200001': 15, '00000ee5': 16,
                            '000010af': 17, '000021': 18, '000062david42': 19,
                            '0001': 20, '0001mpc': 21, '0002': 22, '0003': 23,
                            '00041032': 24, '0004136': 25, '0004246': 26,
                            '0004422': 27, '00044513': 28, '0004847546': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.9516412549199434
============================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                            '000000': 4, '00000000': 5, '0000000004': 6,
                            '0000000005': 7, '0000000667': 8, '0000001200': 9,
                            '00000074': 10, '00000093': 11, '000000e5': 12,
                            '00000315': 13, '000005102000': 14,
                            '00000510200001': 15, '00000ee5': 16,
                            '000010af': 17, '000021': 18, '000062david42': 19,
                            '0001': 20, '0001mpc': 21, '0002': 22, '0003': 23,
                            '00041032': 24, '0004136': 25, '0004246': 26,
                            '0004422': 27, '00044513': 28, '0004847546': 29, ...})
```

```
Модель для классификации - LinearSVC()
Accuracy = 0.9529001660149201
==========================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                               '000000': 4, '00000000': 5, '0000000004': 6,
                               '0000000005': 7, '0000000667': 8, '0000001200': 9,
                               '00000074': 10, '00000093': 11, '000000e5': 12,
                               '00000315': 13, '000005102000': 14,
                               '00000510200001': 15, '00000ee5': 16,
```

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        0.9710610932475884
 1        0.9161073825503355
 2        0.9137931034482759
 3        0.9413793103448276
```


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 = ['формула_S', 'гонщик_S', 'трасса_S', 'пилот_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 - формула_S
5 ближайших соседей слова:
формулировка_S => 0.5704624652862549
схема_S => 0.5608954429626465
термин_S => 0.5511266589164734
уравнение_S => 0.541883111000061
дефиниция_S => 0.48302578926086426

СЛОВО - гонщик_S
5 ближайших соседей слова:
пилот_S => 0.6352430582046509
спортсмен_S => 0.6265261173248291
футболист_S => 0.5708452463150024
теннисист_S => 0.5618873834609985
хоккеист_S => 0.5613346099853516
```

```
СЛОВО - трасса_S
5 ближайших соседей слова:
шоссе_S => 0.6257457733154297
автобан_S => 0.5925841331481934
магистраль_S => 0.5805814862251282
автомагистраль_S => 0.5543080568313599
дорога_S => 0.5536060929298401

СЛОВО - пилот_S
5 ближайших соседей слова:
летчик_S => 0.6687040328979492
гонщик_S => 0.6352430582046509
самолет_S => 0.5455822944641113
космонавт_S => 0.5025683641433716
авиатор_S => 0.49828794598579407
```

Находим близость между словами и строим аналогии

```
print(model.similarity('гонщик_S', 'пилот_S'))
```

```
0.635243
```

```
print(model.most_similar(positive=['гонщик_S', 'пилот_S'], negative=['зритель_S']))
```

```
[('летчик_S', 0.501598060131073), ('williams_UNKN', 0.46746107935905457), ('ferrari_UNKN', 0.4663965404033661), ('jorda
```

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

```
categories = ["rec.motorcycles", "rec.autos", "sci.space","comp.windows.x"]
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]
```

```
[['nicho',
  'vnet',
  'ibm',
  'com',
  'greg',
  'stewart',
  'nicholls',
  'subject',
```

```
          'biosphere',
          'ii',
          'reply',
          'nicho',
          'vnet',
          'ibm',
          'com',
          'disclaimer',
          'posting',
          'represents',
          'poster',
          'views',
          'ibm',
          'news',
          'software',
          'ureply',
          'x',
          'x',
          'nicho',
          'vnet',
          'ibm',
          'com',
          'q',
          'kia',
          'gg',
          'access',
          'digex',
          'net',
          'lines',
          'q',
          'kia',
          'gg',
          'access',
          'digex',
          'net',
          'pat',
          'writes',
          'article',
          'almaden',
          'ibm',
          'com',
          'nicho',
```



```
          'vnet',
          'ibm',
          'com',
          'writes',
          'q',
          'ud',
          'ji',
```

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

```
CPU times: user 6.79 s, sys: 33.6 ms, total: 6.82 s
Wall time: 4.21 s
```

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

```
[('try', 0.9775561094284058), ('make', 0.9725176095962524), ('used', 0.9700536727905273), ('look', 0.9699898362159729),
```

```
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))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to conve
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        0.9571428571428572
1        0.7252252252252253
2        0.7746478873239436
3        0.9055793991416309
```

Вывод:

При выполнении работы для произвольного предложения или текста решены задачи токенизация, частеречная разметка, лемматизация, выделение (распознавание) именованных сущностей, разбор предложения.

Для произвольного набора данных, предназначенного для классификации текстов, решена задача классификации текста двумя способами:

- На основе CountVectorizer
- На основе моделей word2vec

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