Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Лабораторная работа №5
«Предобработка текста»
по дисциплине «Методы машинного обучения»

Выполнил:
студент группы ИУ5-25М
Тураев Г.В.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва 2023

Цель:

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:

```
[1] text1 = '''Московский государственный технический университет им. Н.Э. Баумана — российский национальный исследовательский университет, научный центр, особо ценный объект культурного наследия на
    text2 = 'Предыдущее название университета «Московское высшее техническое училище им. Н.Э. Баумана» было присвоено ему в честь революционера Николая Эрнестовича Баумана, убитого в 1905 году неда
```

```
[2] !pip install natasha
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting natasha
  Downloading natasha-1.5.0-py3-none-any.whl (34.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.4/34.4 MB 28.7 MB/s eta 0:00:00
Collecting pymorphy2 (from natasha)
  Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.5/55.5 kB 3.6 MB/s eta 0:00:00
Collecting razdel>=0.5.0 (from natasha)
  Downloading razdel-0.5.0-py3-none-any.whl (21 kB)
Collecting navec>=0.9.0 (from natasha)
  Downloading navec-0.10.0-py3-none-any.whl (23 kB)
Collecting slovnet>=0.6.0 (from natasha)
  Downloading slovnet-0.6.0-py3-none-any.whl (46 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.7/46.7 kB 1.1 MB/s eta 0:00:00
Collecting yargy>=0.14.0 (from natasha)
  Downloading yargy-0.15.1-py3-none-any.whl (33 kB)
Collecting ipymarkup>=0.8.0 (from natasha)
  Downloading ipymarkup-0.9.0-py3-none-any.whl (14 kB)
Collecting intervaltree>=3 (from ipymarkup>=0.8.0->natasha)
  Downloading intervaltree-3.1.0.tar.gz (32 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from navec>=0.9.0->natasha) (1.22.4)
Collecting dawg-python>=0.7.1 (from pymorphy2->natasha)
  Downloading DAWG_Python-0.7.2-py2.py3-none-any.whl (11 kB)
Collecting pymorphy2-dicts-ru<3.0,>=2.4 (from pymorphy2->natasha)
  Downloading pymorphy2_dicts_ru-2.4.417127.4579844-py2.py3-none-any.whl (8.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.2/8.2 MB 37.3 MB/s eta 0:00:00
Collecting docopt>=0.6 (from pymorphy2->natasha)
  Downloading docopt-0.6.2.tar.gz (25 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: sortedcontainers<3.0,>=2.0 in /usr/local/lib/python3.10/dist-packages (from intervaltree>=3->ipymarkup>=0.8.0->natasha) (2.4.0)
Building wheels for collected packages: docopt, intervaltree
  Building wheel for docopt (setup.py) ... done
  Created wheel for docopt: filename=docopt-0.6.2-py2.py3-none-any.whl size=13707 sha256=a6943b08c66df637620c8bd6339b80813320186e4b90ed953571789327012b2a
  Stored in directory: /root/.cache/pip/wheels/fc/ab/d4/5da2067ac95b36618c629a5f93f809425700506f72c9732fac
  Building wheel for intervaltree (setup.py) ... done
  Created wheel for intervaltree: filename=intervaltree-3.1.0-py2.py3-none-any.whl size=26099 sha256=0042b8c56e9688756c57ee500ff3ba51cbf5ecc0d66ce389833a442cdafe9f4b
  Stored in directory: /root/.cache/pip/wheels/fa/80/8c/43488a924a046b733b64de3fac99252674c892a4c3801c0a61
Successfully built docopt intervaltree
Installing collected packages: razdel, pymorphy2-dicts-ru, docopt, dawg-python, pymorphy2, navec, intervaltree, yargy, slovnet, ipymarkup, natasha
Successfully installed dawg-python-0.7.2 docopt-0.6.2 intervaltree-3.1.0 ipymarkup-0.9.0 natasha-1.5.0 navec-0.10.0 pymorphy2-0.9.1 pymorphy2-dicts-ru-2.4.417127.4579844 razdel-0.5.0 slovnet-0.6
```

Задача токенизации

```
[4] from razdel import tokenize, sentenize
```

```
[6] n_tok_text = list(tokenize(text1))
    n_tok_text
```

```
[Substring(0, 10, 'Московский'),
 Substring(11, 26, 'государственный'),
 Substring(27, 38, 'технический'),
 Substring(39, 50, 'университет'),
 Substring(51, 53, 'им'),
 Substring(53, 54, '.'),
 Substring(55, 56, 'Н'),
 Substring(56, 57, '.'),
 Substring(57, 58, 'Э'),
 Substring(58, 59, '.'),
 Substring(60, 67, 'Баумана'),
 Substring(68, 69, '—'),
 Substring(70, 80, 'российский'),
 Substring(81, 93, 'национальный'),
 Substring(94, 111, 'исследовательский'),
 Substring(112, 123, 'университет'),
 Substring(123, 124, ','),
 Substring(125, 132, 'научный'),
 Substring(133, 138, 'центр'),
 Substring(138, 139, ','),
 Substring(140, 145, 'особо'),
 Substring(146, 152, 'ценный'),
 Substring(153, 159, 'объект'),
 Substring(160, 171, 'культурного'),
 Substring(172, 180, 'наследия'),
 Substring(181, 188, 'народов'),
 Substring(189, 195, 'России'),
 Substring(195, 196, '.')]
```

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

```
['Московский',
 'государственный',
 'технический',
 'университет',
 'им',
 '.',
 'Н',
 '.',
 'Э',
 '.',
 'Баумана',
 '—',
 'российский',
 'национальный',
 'исследовательский',
 'университет',
 ',',
 'научный',
 'центр',
 ',',
 'особо',
 'ценный',
 'объект',
 'культурного',
 'наследия',
 'народов',
 'России',
 '.']
```

```
[12] n_sen_text = list(sentenize(text1))
     n_sen_text
```

```
[Substring(0,
           196,
           'Московский государственный технический университет им. Н.Э. Баумана — российский национальный исследовательский университет, научный центр, особо ценный объект культурного наследия народов России.')]
```

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

```
(['Московский государственный технический университет им. Н.Э. Баумана — российский национальный исследовательский университет, научный центр, особо ценный объект культурного наследия народов России.'],
 1)
```

Этот вариант токенизации нужен для последующей обработки:

```
[18] 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
```

```
[19] n_sen_chunk = n_sentenize(text1)
     n_sen_chunk
```

```
[['Московский',
  'государственный',
  'технический',
  'университет',
  'им',
  '.',
  'Н',
  '.',
  'Э',
  '.',
  'Баумана',
  '—',
  'российский',
  'национальный',
  'исследовательский',
  'университет',
  ',',
  'научный',
  'центр',
  ',',
  'особо',
  'ценный',
  'объект',
  'культурного',
  'наследия',
  'народов',
  'России',
  '.']]
```

```
[21] n_sen_chunk_2 = n_sentenize(text2)
     n_sen_chunk_2
```

```
[['Предыдущее',
  'название',
  'университета',
  '«',
  'Московское',
  'высшее',
  'техническое',
  'училище',
  'им',
  '.',
  'Н',
  '.',
  'Э',
  '.',
  'Баумана',
  '»',
  'было',
  'присвоено',
  'ему',
  'в',
  'честь',
  'революционера',
  'Николая',
  'Эрнестовича',
  'Баумана',
  ',',
  'убитого',
  'в',
  '1905',
  'году',
  'недалеко',
  'от',
  'главного',
  'здания',
  'в',
  'то',
  'время',
  '—',
  'Императорского',
  'московского',
  'технического',
  'училища',
  '.']]
```

Частеречная разметка

Скачаем файл «navec_news_v1_1B_250K_300d_100q» по ссылке https://github.com/natasha/navec#downloads

```
[28] navec = Navec.load('navec_news_v1_1B_250K_300d_100q.tar')
```

Также скачаем файл «slovnet_morph_news_v1» по ссылке: https://github.com/natasha/slovnet#downloads

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

```
[32] morph_res = n_morph.navec(navec)
```

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

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

```
Московский - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
государственный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
технический - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
университет - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing
им - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
. - PUNCT
Н - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
. - PUNCT
Э - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
. - PUNCT
Баумана - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
— - PUNCT
российский - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
национальный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
исследовательский - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
университет - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
, - PUNCT
научный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
центр - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
, - PUNCT
особо - ADV|Degree=Pos
ценный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
объект - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
культурного - ADJ|Case=Gen|Degree=Pos|Gender=Neut|Number=Sing
наследия - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
народов - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Plur
России - PROPN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
. - PUNCT
[None]
```

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

```
Предыдущее - ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Gender=Neut|Number=Sing
название - NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
университета - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
« - PUNCT
Московское - ADJ|Case=Nom|Degree=Pos|Gender=Neut|Number=Sing
высшее - ADJ|Case=Nom|Degree=Pos|Gender=Neut|Number=Sing
техническое - ADJ|Case=Nom|Degree=Pos|Gender=Neut|Number=Sing
училище - NOUN|Animacy=Inan|Case=Acc|Gender=Neut|Number=Sing
им - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
. - PUNCT
Н - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
. - PUNCT
Э - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
. - PUNCT
Баумана - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
» - PUNCT
было - AUX|Aspect=Imp|Gender=Neut|Mood=Ind|Number=Sing|Tense=Past|VerbForm=Fin|Voice=Act
присвоено - VERB|Aspect=Perf|Gender=Neut|Number=Sing|Tense=Past|Variant=Short|VerbForm=Part|Voice=Pass
ему - PRON|Case=Dat|Gender=Masc|Number=Sing|Person=3
в - ADP
честь - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Sing
революционера - NOUN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
Николая - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
Эрнестовича - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
Баумана - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
, - PUNCT
убитого - VERB|Aspect=Perf|Case=Gen|Gender=Masc|Number=Sing|Tense=Past|VerbForm=Part|Voice=Pass
в - ADP
1905 - ADJ
году - NOUN|Animacy=Inan|Case=Loc|Gender=Masc|Number=Sing
недалеко - ADV|Degree=Pos
от - ADP
главного - ADJ|Case=Gen|Degree=Pos|Gender=Neut|Number=Sing
здания - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
в - ADP
то - DET|Animacy=Inan|Case=Acc|Gender=Neut|Number=Sing
время - NOUN|Animacy=Inan|Case=Acc|Gender=Neut|Number=Sing
— - PUNCT
Императорского - ADJ|Case=Gen|Degree=Pos|Gender=Neut|Number=Sing
московского - ADJ|Case=Gen|Degree=Pos|Gender=Neut|Number=Sing
технического - ADJ|Case=Gen|Degree=Pos|Gender=Neut|Number=Sing
училища - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
. - PUNCT
[None]
```

Лемматизация

```
[41] from natasha import Doc, Segmenter, NewsEmbedding, NewsMorphTagger, MorphVocab
```

```
[48] 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
```

```
[53] n_doc = n_lemmatize(text1)
     {_.text: _.lemma for _ in n_doc.tokens}
```

```
{'Московский': 'московский',
 'государственный': 'государственный',
 'технический': 'технический',
 'университет': 'университет',
 'им': 'имя',
 '.': '.',
 'Н': 'н',
 'Э': 'э',
 'Баумана': 'бауман',
 '»': '»',
 'российский': 'российский',
 'национальный': 'национальный',
 'исследовательский': 'исследовательский',
 ',': ',',
 'научный': 'научный',
 'центр': 'центр',
 'особо': 'особо',
 'ценный': 'ценный',
 'объект': 'объект',
 'культурного': 'культурный',
 'наследия': 'наследие',
 'народов': 'народ',
 'России': 'россия'}
```

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

```
{'Предыдущее': 'предыдущий',
 'название': 'название',
 'университета': 'университет',
 '«': '«',
 'Московское': 'московский',
 'высшее': 'высокий',
 'техническое': 'технический',
 'училище': 'училище',
 'им': 'имя',
 '.': '.',
 'Н': 'н',
 'Э': 'э',
 'Баумана': 'бауман',
 '»': '»',
 'было': 'быть',
 'присвоено': 'присвоить',
 'ему': 'он',
 'в': 'в',
 'честь': 'честь',
 'революционера': 'революционер',
 'Николая': 'николай',
 'Эрнестовича': 'эрнестович',
 ',': ',',
 'убитого': 'убить',
 '1905': '1905',
 'году': 'год',
 'недалеко': 'недалеко',
 'от': 'от',
 'главного': 'главный',
 'здания': 'здание',
 'то': 'тот',
 'время': 'время',
 '—': '—',
 'Императорского': 'императорский',
 'московского': 'московский',
 'технического': 'технический',
 'училища': 'училище'}
```

Выделение (распознавание) именованных сущностей

Скачаем файл «slovnet_ner_news_v1» по ссылке: https://github.com/natasha/slovnet#downloads, импортируя перед этим библиотеки:

```
[58] from slovnet import NER
     from ipymarkup import show_span_ascii_markup as show_markup
```

```
[60] ner = NER.load('slovnet_ner_news_v1.tar')
```

```
[62] ner_res = ner.navec(navec)
```

```
[64] markup_ner = ner(text2)
     markup_ner
```

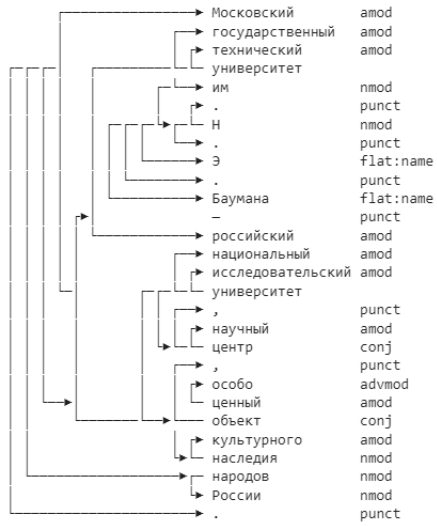
```
SpanMarkup(
    text='Предыдущее название университета «Московское высшее техническое училище им. Н.Э. Баумана» было присвоено ему в честь революционера Николая Эрнестовича Баумана, убитого в 1905 году недалеко от главного здания в то время — Императорского московского технического училища.',
    spans=[Span(
         start=34,
         stop=88,
         type='ORG'
     ),
     Span(
         start=131,
         stop=158,
         type='PER'
     ),
     Span(
         start=221,
         stop=268,
         type='ORG'
     )]
)
```

```
[67] show_markup(markup_ner.text, markup_ner.spans)
```

```
Предыдущее название университета «Московское высшее техническое
                                  ORG─────────────────────────────
училище им. Н.Э. Баумана» было присвоено ему в честь революционера
──────────────────────
Николая Эрнестовича Баумана, убитого в 1905 году недалеко от главного
PER─────────────────────
здания в то время — Императорского московского технического училища.
                   ORG─────────────────────────────────────────────
```

Разбор предложения

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




```
[79] 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"

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

```
[85] 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]))
```

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

     Количество сформированных признаков - 33448
```

```
[91] 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 (способ 1)

```
[95] 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:

```
[96] len(test_features.todense()[0].getA1())
```

```
33448
```

Непустые значения нулевой строки:

```
[100] 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, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2]
```

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

```
[104] 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('============================')
```

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

```
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(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: 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
```

...

```
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 = 0.937813339432037
============================
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, '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

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

```
[109] 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)
```

```
[110] sentiment(CountVectorizer(), LinearSVC())
```

Метка	Accuracy
0	0.9290322580645162
1	0.9675090252707581
2	0.9026845637583892
3	0.9245901639344263

Вывод: по результатам лабораторной работы, а именно проверки качества моделей, можно сделать вывод о том, что лучшее качество показал CountVectorizer.