

▼ Лабораторная работа №5

"Предобработка и классификация текстовых данных"

Выполнила: Ся Тунтун

Группа: ИУ5-23М

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".

```
import pandas as pd
from nltk import tokenize
```

```
!pip install natasha
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple
Requirement already satisfied: natasha in /usr/local/lib/python3.7/dist-packages (1.4.0)
Requirement already satisfied: yargy>=0.14.0 in /usr/local/lib/python3.7/dist-packages (from natasha)
Requirement already satisfied: slovnet>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from yargy>=0.14.0)
Requirement already satisfied: ipymarkup>=0.8.0 in /usr/local/lib/python3.7/dist-packages (from slovnet>=0.3.0)
Requirement already satisfied: razdel>=0.5.0 in /usr/local/lib/python3.7/dist-packages (from slovnet>=0.3.0)
Requirement already satisfied: navec>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from slovnet>=0.3.0)
Requirement already satisfied: pymorphy2 in /usr/local/lib/python3.7/dist-packages (from slovnet>=0.3.0)
Requirement already satisfied: intervaltree>=3 in /usr/local/lib/python3.7/dist-packages (from slovnet>=0.3.0)
Requirement already satisfied: sortedcontainers<3.0, >=2.0 in /usr/local/lib/python3.7/dist-packages (from slovnet>=0.3.0)
```

Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from navec>=C
 Requirement already satisfied: dawg-python>=0.7.1 in /usr/local/lib/python3.7/dist-packages (
 Requirement already satisfied: pymorphy2-dicts-ru<3.0,>=2.4 in /usr/local/lib/python3.7/dist-
 Requirement already satisfied: docopt>=0.6 in /usr/local/lib/python3.7/dist-packages (from py

```
!pip install razdel
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/si>
 Requirement already satisfied: razdel in /usr/local/lib/python3.7/dist-packages (0.5.0)

Загрузим датасет с классификацией записей в сети Твиттер и предполагаемой тональностью их содержимого:

```
# # Подключение к gogle диску
# from google.colab import drive
# drive.mount('/content/drive')

# # Вывод содержимого папки на диске
# import os
# data_root = '/content/drive/MyDrive/MMO'
# print(os.listdir(data_root))

# # Распаковка архива с датасетом
# !unzip /content/drive/MyDrive/MMO/train.zip
# # Unpack files from zip-file
# # import zipfile
# # with zipfile.ZipFile('/content/drive/MyDrive/MMO/ml-latest-small.zip', 'r') as zip_ref:
# #     zip_ref.extractall(BASE_DIR)

# df_class = pd.read_csv('/content/train.csv', sep=",")
# df_class.head()

text = '''Дмитрий Иванович Менделеев родился 27 января
Они должны были умереть еще три дня назад, когда о
Перси убивал их и своими глазами видел, как они об
Он взобрался на вершину холма и перевел дух. Сколь
В последние дни Перси почти не спал. Ел он то, что
Перси до сих пор был жив только потому, что две зм

text2 = 'МГТУ им. Н. Э. Баумана — российский националь

test_text = 'Праздник весны является самым любимым праз

# # выделим тестовое сообщение, с которым затем буд
# test_val = 100
# texts = df_class['content']
```

```
# test_text = texts.iloc[test_val]
# . . . .
```

▼ Предобработка текста

▼ Токенизация

NLTK

Содержит большое количество токенизаторов. На практике они не всегда стабильно работают для русского языка.

```
import nltk
nltk.download('punkt')

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
True
```

```
from nltk import tokenize
dir(tokenize)[:18]
```

```
['BlanklineTokenizer',
 'LineTokenizer',
 'MWETokenizer',
 'PunktSentenceTokenizer',
 'RegexpTokenizer',
 'ReppTokenizer',
 'SExprTokenizer',
 'SpaceTokenizer',
 'StanfordSegmenter',
 'TabTokenizer',
 'TextTilingTokenizer',
 'ToktokTokenizer',
 'TreebankWordTokenizer',
 'TweetTokenizer',
 'WhitespaceTokenizer',
 'WordPunctTokenizer',
 '__builtins__',
 '__cached__']
```

Токенизация по предложениям:

```
nltk_tk_sents = nltk.tokenize.sent_tokenize(test_text)
print(len(nltk_tk_sents))
nltk_tk_sents
```

```
1
[' П р а з д н и к  в е с н ы  я в л я е т с я  с а м ы м  л ю б и м ы м  п р а з д н и к о м
```

Токенизация по словам:

```
nltk Tk_1 = nltk.WordPunctTokenizer()
nltk Tk_1.tokenize(test_text)
```

```
[' П р а з д н и к',
 ' в е с н ы',
 ' я в л я е т с я',
 ' с а м ы м',
 ' л ю б и м ы м',
 ' п р а з д н и к о м',
 ' к и т а й ц е в',
 ',',
 ' Н а к а н у н е',
 ' е г о',
 ' к и т а й ц ы',
 ',',
 ' к о т о р ы е',
 ' р а б а т а ю т',
 ' и л и',
 ' з а н и м а ю т с я',
 ' в',
 ' д р у г и х',
 ' м е с т а х',
 ',',
 ' н е с м о т р я',
 ' н а',
 ' д а л ё к и й',
 ' п у т ь',
 ',',
 ' в с е г д а',
 ' с п е ш а т',
 ' к',
 ' р о д н ы м',
 ' о ч а г а м',
 ',',
 ' ч т о б ы',
 ' п р о в е с т и',
 ' э т о т',
 ' п р а з д н и к',
 ' д о м а',
 ',',
 ' с о',
 ' с в о и м и',
 ' б л и з к и м и',
 '.']
```

Natasha

Для токенизации используется библиотека <https://github.com/natasha/razdel>

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

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

```

[Substring(0, 7, 'Д м и т р и й'),
 Substring(8, 16, 'И в а н о в и ч'),
 Substring(17, 26, 'М е н д е л е е в'),
 Substring(27, 34, 'р о д и л с я'),
 Substring(35, 37, '27'),
 Substring(38, 44, 'я н в а р я'),
 Substring(45, 46, '('),
 Substring(46, 47, '8'),
 Substring(48, 55, 'ф е в р а л я'),
 Substring(55, 56, ')'),
 Substring(57, 61, '1834'),
 Substring(62, 66, 'г о д а'),
 Substring(67, 68, 'в'),
 Substring(69, 78, 'Т о б о л ь с к е'),
 Substring(79, 80, 'в'),
 Substring(81, 86, 'с е м ь е'),
 Substring(87, 92, 'И в а н а'),
 Substring(93, 102, 'П а в л о в и ч а'),
 Substring(103, 113, 'М е н д е л е е в а'),
 Substring(113, 114, ','),
 Substring(115, 116, 'в'),
 Substring(117, 119, 'т о'),
 Substring(120, 125, 'в р е м я'),
 Substring(126, 137, 'з а н и м а в ш е г о'),
 Substring(138, 147, 'д о л ж н о с т ь'),
 Substring(148, 157, 'д и р е к т о р а'),
 Substring(158, 168, 'Т о б о л ь с к о й'),
 Substring(169, 177, 'г и м н а з и и'),
 Substring(178, 179, 'и'),
 Substring(180, 186, 'у ч и л и щ'),
 Substring(187, 198, 'Т о б о л ь с к о г о'),
 Substring(199, 205, 'о к р у г а'),
 Substring(205, 206, ','),
 Substring(207, 208, 'и'),
 Substring(209, 214, 'М а р и и'),
 Substring(215, 225, 'Д м и т р и е в н ы'),
 Substring(226, 237, 'М е н д е л е е в о й'),
 Substring(238, 239, '('),
 Substring(239, 250, 'К о р н и л ь е в о й'),
 Substring(250, 251, ')'),
 Substring(251, 252, '.'),
 Substring(252, 255, 'Э т и'),
 Substring(256, 267, 'з м е е в о л о с ы е'),
 Substring(268, 275, 'д а м о ч к и'),
 Substring(276, 279, 'у ж е'),
 Substring(280, 286, 'н а ч а л и'),
 Substring(287, 297, 'р а з д р а ж а т ь'),
 Substring(298, 303, 'П е р с и'),
 Substring(303, 304, '.'),
 Substring(305, 308, 'О н и'),
 Substring(309, 315, 'д о л ж н ы'),
 Substring(316, 320, 'б ы л и'),
 Substring(321, 328, 'у м е р е т ь'),
 Substring(329, 332, 'е щ е'),
 Substring(333, 336, 'т р и'),
 Substring(337, 340, 'д н я'),
 Substring(341, 346, 'н а з а д'),
 Substring(346, 347, ',')

```

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

```
['Д м и т р и й',  
'И в а н о в и ч',  
'М е н д е л е е в',  
'р о д и л с я',  
'27',  
'я н в а р я',  
'(',  
'8',  
'ф е в р а л я',  
)',  
'1834',  
'г о д а',  
'в',  
'Т о б о л ь с к е',  
'в',  
'с е м ь е',  
'И в а н а',  
'П а в л о в и ч а',  
'М е н д е л е е в а',  
,,  
,,  
'в',  
'т о',  
'в р е м я',  
'з а н и м а в ш е г о',  
'д о л ж н о с т ь',  
'д и р е к т о р а',  
'Т о б о л ь с к о й',  
'г и м н а з и и',  
'и',  
'у ч и л и щ',  
'Т о б о л ь с к о г о',  
'о к р у г а',  
,,  
,,  
'и',  
'М а р и и',  
'Д м и т р и е в н ы',  
'М е н д е л е е в о й',  
'(',  
'К о р н и л ь е в о й',  
)',  
,,  
,,  
'Э т и',  
'з м е е в о л о с ы е',  
'д а м о ч к и',  
'у ж е',  
'н а ч а л и',  
'р а з д р а ж а т ь',  
'П е р с и',  
,,  
,,  
'О н и',  
'д о л ж н ы',  
'б ы л и',  
'у м е р е т ь',  
'е щ е',  
'т р и',  
'д н я',  
'н а з а д',
```

```

n_sen_text = list(sentenize(text))
n_sen_text

[Substring(0,
          304,
          'Д м и т р и й \xa0И в а н о в и ч \xa0М е н д е л е е в \xa0р о д и л с я \xa027\
Substring(305,
          423,
          'О н и д о л ж н ы б ы л и у м е р е т ь е щ е т р и д н я н а з а д , к о
Substring(424,
          534,
          'О н и д о л ж н ы б ы л и о т д а т ь к о н ц ы д в а д н я н а з а d , п
Substring(535,
          628,
          'И у ж т о ч н о о н и д о л ж н ы б ы л и с д о х н у т ь , к о г д а о н
Substring(629,
          748,
          'П е р с и у б и в а л и х и с в о и м и г л а з а м и в и д е л , к а к о
Substring(749, 799, 'О н , п о х о ж е , д а ж е н е м о г н а д о л г о о т н и х о
Substring(800, 844, 'О н в з о б р а л с я н а в е р ш и н у х о л м а и п е р е в е
Substring(845,
          915,
          'С к о л ь к о в р е м е н и п р о ш л о с т е х п о р , к а к о н п р и к
Substring(916, 935, 'Ч а с а д в а , н а в е р н о е .'),
Substring(936, 990, 'К а ж е т с я , о н и т е п е р ь н е у м и р а ю т б о л ь ш е ч
Substring(991, 1027, 'В п о с л е д н и е д н и П е р с и п о ч т и н е с п а л .'),
Substring(1028,
          1195,
          'Е л о н т о , ч т о у д а в а л о с ь с т я н у т ь п о д о р о г е , - ж
Substring(1196,
          1271,
          'О д е ж д а е г о п о р в а л а с ь , м е с т а м и о б г о р е л а и в с я
Substring(1272,
          1419,
          'П е р с и д о с и х п о р б ы л ж и в т о л ь к о п о т о м у , ч т о д
Substring(1420, 1460, 'И х к о г т и н е о с т а в л я л и с л е д а н а е г о к о ж
Substring(1461, 1513, 'Е с л и о н и п ы т а л и с ь е г о у к у с и т ь - з у б ы у
Substring(1514, 1542, 'Н о П е р с и у ж е б ы л н а п р е д е л е .'),
Substring(1543,
          1631,
          'С к о р о о н с в а л и т с я о т и с т о щ е н и я , а т о г д а ... х о т
Substring(1632, 1656, 'О н в э т о м н е с о м н е в а л с я .')]

```

```

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

```

```

(['Д м и т р и й \xa0И в а н о в и ч \xa0М е н д е л е е в \xa0р о д и л с я \xa027\xa0я н в
'О н и д о л ж н ы б ы л и у м е р е т ь е щ е т р и д н я н а з а d , к о г д а о
'О н и д о л ж н ы б ы л и o т d а т ь к o n c ы d v a d n я n a z a d , п o c л e
'И у ж т o ч н o o н и д o л ж н ы б ы л и s d o x n u t ь , к o г d a o n o t r e
'П е р с и у б и в а л и х и c v o i m i g l a z a m i v i d e l , к a k o n i o b
'О н , п o х o ж e , d a ж e n e m o г n a d o л г o o t n i x o t o p в а т ь с я .'
'О н в з o б р a л s я n a в e р ш и n у x o л m a и п e р e v e л d y x .',
'С к o л ь к o v р e m e n и p р o ш l o c t e x p o r , к a k o n п р и k o n ч и .
'Ч а с a d v a , n a v e р n o e .',
'К а ж e т s я , o n и t e п e р ь n e y m и r a ю т б o л ь ш e ч e m n a d v a ч
'В п o c л e d n и e d n и П e р s и п o ч t и n e c п a л .',

```

```
'Ел он то, что удавалось стянуть по дороге, - жевать
'Одежда его порвалась, местами обгорела и вся была
'Перси до сих пор был жив только потому, что две змеи
'Их когти не оставляли следа на его коже.',
'Если они пытались его укусить - зубы у них ломались
'Но Перси уже был на пределе.',
'Скоро он свалится от истощения, а тогда... хоть его
'Он в этом не сомневался.'],
19)
```

Этот вариант токенизации нужен для последующей обработки

```
def n_sentence(text):
    n_sentence_chunk = []
    for sent in sentence(text):
        tokens = [_.text for _ in tokenize(sent.text)]
        n_sentence_chunk.append(tokens)
    return n_sentence_chunk
```

```
n_sentence_chunk = n_sentence(text)
n_sentence_chunk
```

```
[['Д м и т р и й',
'И в а н о в и ч',
'М е н д е л е е в',
'р о д и л с я',
'27',
'я н в а р я',
'(',
'8',
'ф е в р а л я',
')',
'1834',
'г о д а',
'в',
'Т о б о л ь с к е',
'в',
'с е м ь е',
'И в а н а',
'П а в л о в и ч а',
'М е н д е л е е в а',
',',
',',
'в',
'т о',
'в р е м я',
'з а н и м а в ш е г о',
'д о л ж н о с т ь',
'д и р е к т о р а',
'Т о б о л ь с к о й',
'г и м н а з и и',
'и',
'у ч и л и щ',
'Т о б о л ь с к о г о',
'о к р у г а',
',',
',',
'и',
'М а р и и',
```



```

'Д м и т р и е в н ы',
'М е н д е л е е в о й',
'(',
'К о р н и л ь е в о й',
')',
',',
',',
'Э т и',
'з м е е в о л о с ы е',
'д а м о ч к и',
'у ж е',
'н а ч а л и',
'р а з д р а ж а т ь',
'П е р с и',
'.'],
['О н и',
'д о л ж н ы',
'б ы л и',
'у м е р е т ь',
'е щ е',
'т р и',
'д н я',
'т а з а п',

```

```

n_sen_chunk_2 = n_sentenize(text2)
n_sen_chunk_2

```

```

[['М Г Т У',
'и м',
',',
',',
'Н',
',',
',',
'Э',
',',
',',
'Бáу м а н а',
'—',
',',
'р о с с и й с к и й',
'н а ц и о н а л ь н ы й',
'и с с л е д о в а т е л ь с к и й',
'у н и в е р с и т е т',
',',
',',
'П р е д ы д у щ е е',
'н а з в а н и е',
'у н и в е р с и т е т а',
'«',
'М о с к ó в с к о е',
'в ы с ш е е',
'т е х н í ч е с к о е',
'у ч í л и щ е',
'и м',
',',
',',
',',
'Н',
',',
',',
'Э',
',',
',',
'Бáу м а н а',
'»',
'б ы л о',
'п р и с в о е н о',
'е м у',

```

```
' в',
' честь',
' революционера',
' Николая',
' Эрнестовича',
' Баумана',
', ']]
```

▼ Частеричная разметка (Part-Of-Speech tagging, POS-tagging)

В некоторых библиотеках вначале выполняется частеречная разметка, а далее на ее основе выполняется лемматизация.

Spacy

```
from spacy.lang.en import English
import spacy
nlp = spacy.load('en_core_web_sm')
spacy_test = nlp(test_text)
# from spacy.lang.ru import Russian
# import spacy
# nlp = spacy.load('ru_core_news_sm')
# spacy_test = nlp(test_text)
# spacy_test
```

Просмотрим какие части речи присутствуют в тестовом твите:

```
for token in spacy_test:
    print('{} - {} - {}'.format(token.text, token.pos_, token.dep_))
```

```
П р а з д н и к - PROPN - compound
в е с н ы - VERB - compound
я в л я е т с я - ADJ - compound
с а м ы м - PROPN - compound
л ю б и м ы м - VERB - compound
п р а з д н и к о м - NOUN - compound
к и т а й ц е в - PROPN - ROOT
. Н а к а н у н е - PUNCT - punct
е г о - PROPN - compound
к и т а й ц ы - NOUN - nsubj
, - PUNCT - punct
к о т о р ы е - PROPN - ROOT
- SPACE -
р а б а т а ю т - PROPN - nsubj
и л и - PROPN - ccomp
з а н и м а ю т с я - ADV - amod
в - PROPN - compound
д р у г и х - NOUN - compound
```

```

м е с т а х - ADJ - dobj
, - PUNCT - punct
н е с м о т р я - PROPN - compound
н а - PROPN - compound
д а л ё к и й - PROPN - compound
п у т ь - PROPN - conj
, - PUNCT - punct
в с е г д а - NOUN - compound
с п е ш а т - NOUN - compound
к - PROPN - compound
р о д н ы м - NOUN - compound
о ч а г а м - PROPN - conj
, - PUNCT - punct
ч т о б ы - PROPN - compound
п р о в е с т и - PROPN - nsubj
э т о т - VERB - conj
п р а з д н и к - PROPN - compound
д о м а - PROPN - dobj
, - PUNCT - punct
с о - PROPN - intj
с в о и м и - PROPN - compound
б л и з к и м и - PROPN - appos
. - PUNCT - punct

```

Natasha

```

from navec import Navec
from slovnet import Morph

```

```

# Ф а й л  н е о б х о д и м о  с к а ч а т ь  п о  с с ы л к е  https://github.com/natasha/n
navec = Navec.load('/content/navec_news_v1_1B_250K_300d_100q (1).tar')

```

```

# Ф а й л  н е о б х о д и м о  с к а ч а т ь  п о  с с ы л к е  https://github.com/natasha/s
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]

```

```

Д м и т р и й - PROPN|Animacy=Anim|Case=Nom|Gender=Masc|Number=Sing
И в а н о в и ч - PROPN|Animacy=Anim|Case=Nom|Gender=Masc|Number=Sing
М е н д е л е е в - PROPN|Animacy=Anim|Case=Nom|Gender=Masc|Number=Sing
р о д и л с я - VERB|Aspect=Perf|Gender=Masc|Mood=Ind|Number=Sing|Tense=Past|VerbForm=Fin
27 - ADJ
я н в а р я - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
( - PUNCT
8 - ADJ

```

М Г Т У - PROPN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
и м - 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
Ба́у ма́на - PROPN|Animacy=Anim|Case=Nom|Gender=Masc|Number=Sing
— - PUNCT
ро́с сий ский - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
на́ цио на́ль ный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
ис сле до ва те лья́ ский - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
у ни ве р си те т - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
, - PUNCT
Пре́ ды ду щее - ADJ|Case=Nom|Degree=Pos|Gender=Neut|Number=Sing
на́з ва́ ние - NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
у ни ве р си те та́ - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
« - PUNCT
Мо́с ко́в ское - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
вы́с шее - X|Foreign=Yes
тех́ни́чес ко́е - ADJ|Case=Dat|Degree=Pos|Gender=Masc|Number=Sing
учи́лище - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
и́м - PRON|Case=Ins|Gender=Masc|Number=Sing|Person=3
. - PUNCT
Н - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
. - PUNCT
Э - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
. - PUNCT
Ба́у ма́на - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
» - PUNCT
бы́ло - AUX|Aspect=Imp|Gender=Neut|Mood=Ind|Number=Sing|Tense=Past|VerbForm=Fin|Voice=Act
при́своено́ - VERB|Aspect=Perf|Gender=Neut|Number=Sing|Tense=Past|Variant=Short|VerbFc
ему́ - PRON|Case=Dat|Gender=Masc|Number=Sing|Person=3
в - ADP
че́сть - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Sing
рево́люцио́нера - NOUN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
Ни́ко ла́я - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
Э́р не́сто вича́ - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
Ба́у ма́на - PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
, - PUNCT
[None]
```

▼ Лемматизация

Spacy

```
for token in spacy_test:
    print(token, token.lemma, token.lemma_)
```

```
П р а з д н и к 1840354460546522912 П р а з д н и к
в е с н ы 14721510421981876170 в е с н ы
я в л я е т с я 11228620018285619923 я в л я е т с я
с а м ы м 131047934457931729 с а м ы м
лю б и м ы м 14607805921186229421 лю б и м ы м
п р а з д н и к о м 2380343434576960769 п р а з д н и к о м
к и т а й ц е в 17453396853267337055 к и т а й ц е в
. Н а к а н у н е 11641952432637114577 . Н а к а н у н е
е г о 6166162853077514292 е г о
к и т а й ц ы 7787469979702150596 к и т а й ц ы
, 2593208677638477497 ,
```

к о т о р ы е 10821169212437491113 к о т о р ы е
 13912603054523871734
 р а б а т а ю т 12496267239876250009 р а б а т а ю т
 и л и 1530020831762146143 и л и
 з а н и м а ю т с я 5289187509458912700 з а н и м а ю т с я
 в 15939375860797385675 в
 д р у г и х 5568520122224931142 д р у г и х
 м е с т а х 14922183477643145597 м е с т а х
 , 2593208677638477497 ,
 н е с м о т р я 4313172793392934716 н е с м о т р я
 н а 16191904166009283104 н а
 д а л ё к и й 6723292147973124723 д а л ё к и й
 п у т ь 10530679371794504240 п у т ь
 , 2593208677638477497 ,
 в с е г д а 10633257961924346802 в с е г д а
 с п е ш а т 14656338628323285350 с п е ш а т
 к 2390146911029080849 к
 р о д н ы м 18367528641597240676 р о д н ы м
 о ч а г а м 6535154930049018912 о ч а г а м
 , 2593208677638477497 ,
 ч т о б ы 10327972121992521358 ч т о б ы
 п р о в е с т и 5160007530980309479 п р о в е с т и
 э т о т 13138259693353519549 э т о т
 п р а з д н и к 8745251629403612823 п р а з д н и к
 д о м а 1579389597620111726 д о м а
 , 2593208677638477497 ,
 с о 12039906729841018817 с о
 с в о и м и 12718946594216898768 с в о и м и
 б л и з к и м и 6698017153815111151 б л и з к и м и
 . 12646065887601541794 .

Natasha

```
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}
```

```
{ '(': '(',
  ')': ')',
  ',': ',',
  '.': '.',
  ':': ':',
  '1834': '1834',
```

'27': '27',
 '8': '8',
 '?: '?',
 '«': '«',
 '»': '»',
 'Баргин-Марте': 'баргин-март',
 'В': 'в',
 'Дмитриевны': 'дмитриевич',
 'Дмитрий': 'дмитрий',
 'Ел': 'есть',
 'Если': 'если',
 'И': 'и',
 'Ивана': 'иван',
 'Иванович': 'иванович',
 'Их': 'их',
 'Кажется': 'казаться',
 'Корнильевой': 'корнильевой',
 'Марии': 'мария',
 'Мартинесе': 'мартинес',
 'Менделеев': 'менделеев',
 'Менделеева': 'менделеев',
 'Менделеевой': 'менделеев',
 'Напе': 'напе',
 'Но': 'но',
 'Одежда': 'одежда',
 'Он': 'он',
 'Они': 'они',
 'Павловича': 'павлович',
 'Перси': 'перси',
 'Сколько': 'сколько',
 'Скоро': 'скоро',
 'Тилден-парке': 'тилден-парк',
 'Тобольске': 'тобольск',
 'Тобольского': 'тобольский',
 'Тобольской': 'тобольский',
 'Часа': 'час',
 'Эти': 'этот',
 'а': 'а',
 'автомата': 'автомат',
 'автомобилем': 'автомобиль',
 'больше': 'большой',
 'боулинга': 'боулинг',
 'бублик': 'бублик',
 'буррито': 'буррито',
 'был': 'быть',
 'была': 'быть',
 'были': 'быть',
 'в': 'в',
 'вершину': 'вершина',
 'взобрался': 'взобратся',
 'видел': 'видеть',
 'возвращались': 'возвращаться',
 'времени': 'время',

```

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

```

```

{',': ',',
 '.': '.',

```

```
'«': '«',
'»': '»',
'Ба́у ма на': 'ба́у ма н',
'Ба у ма на': 'ба у ма н',
'МГТУ': 'м г т у',
'Мо скóв ско е': 'мо скóв ски й',
'Н': 'н',
'Ни ко ла я': 'ни ко ла й',
'Пр е ды ду щ е е': 'пр е ды ду щ и й',
'Э': 'э',
'Э р не ст о ви ча': 'э р не ст о ви ч',
'бы ло': 'бы ть',
'в': 'в',
'вЫс ше е': 'вЫс ше е',
'е му': 'о н',
'им': 'о н',
'и с с л е до в а т е л ь с к и й': 'и с с л е до в а т е л ь с к и й',
'на з ва ни е': 'на з ва ни е',
'на ци о на л ь н ы й': 'на ци о на л ь н ы й',
'пр ис во е но': 'пр ис во и т ь',
'ре во лю ци о не ра': 'ре во лю ци о не р',
'ро с с и й с к и й': 'ро с с и й с к и й',
'те х н и ч е с к о е': 'те х н и ч е с к и й',
'у ни ве р с и т е т': 'у ни ве р с и т е т',
'у ни ве р с и т е та': 'у ни ве р с и т е т',
'у ч и л и щ е': 'у ч и л и щ е',
'че ст ь': 'че ст ь',
'—': '—'}

```

Выделение (распознавание) именованных сущностей, named-entity recognition (NER)

Spacy

```
for ent in spacy_test.ents:
    print(ent.text, ent.label_)

не с мо т ря на да л ё к и й п у т ь PERSON
ч то б ы п ро ве с т и э т о т п ра з д н и к д о м а PERSON
с о с во и м и PERSON

print(spacy.explain("ORDINAL"))

"first", "second", etc.

print(spacy.explain("PRODUCT"))

Objects, vehicles, foods, etc. (not services)

print(spacy.explain("LOC"))

```


Non-GPE locations, mountain ranges, bodies of water

```
print(spacy.explain("PER"))
```

Named person or family.

```
from spacy import displacy
displacy.render(spacy_test, style='ent', jupyter=True)
```

Natasha

```
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='М Г Т У им. Н. Э. Баумана — российский национальн
  spans=[Span(
    start=0,
    stop=23,
    type='ORG'
  ), Span(
    start=116,
    stop=176,
    type='ORG'
  ), Span(
    start=219,
    stop=246,
    type='PER'
  )]
)
```

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

```
М Г Т У им. Н. Э. Баумана — российский национальн
ORG_____
университет, Предыдущее название университета «Мо
ORG_____
```

техническое училище им. Н. Э. Баумана» было присвоено
революционера Николая Эрнестовича Баумана,
PER

▼ Разбор предложения

Spacy

```
from spacy import displacy
```

```
displacy.render(spacy_test, style='dep', jupyter=True)
```

```
print(spacy.explain("NOUN"))
```

noun

```
print(spacy.explain("amod"))
```

adjectival modifier

Natasha

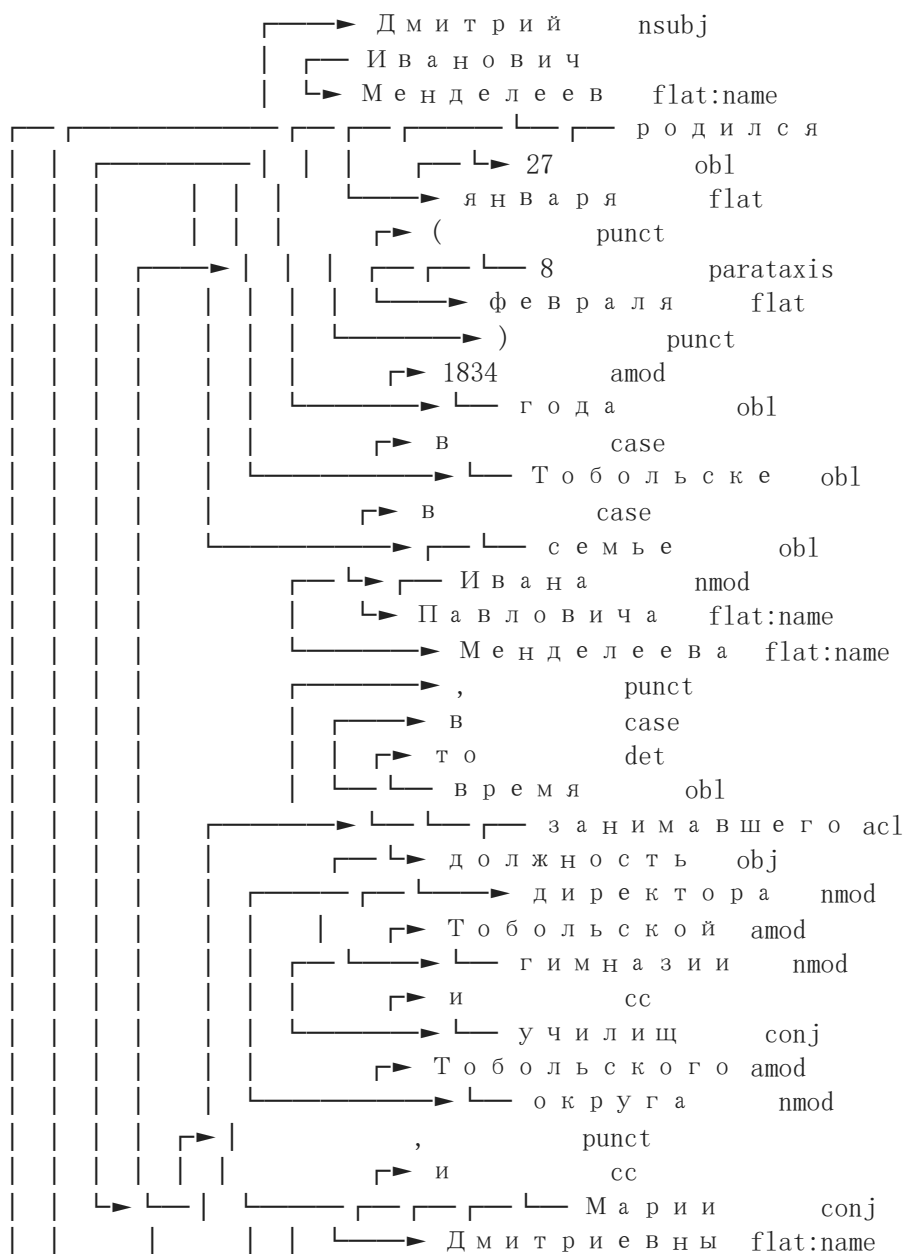
```
from natasha import NewsSyntaxParser
```

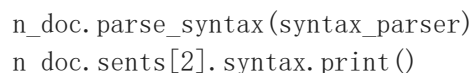
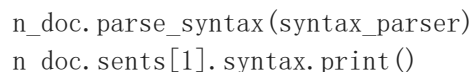
```
emb = NewsEmbedding()
```

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

```
n_doc.parse_syntax(syntax_parser)
```

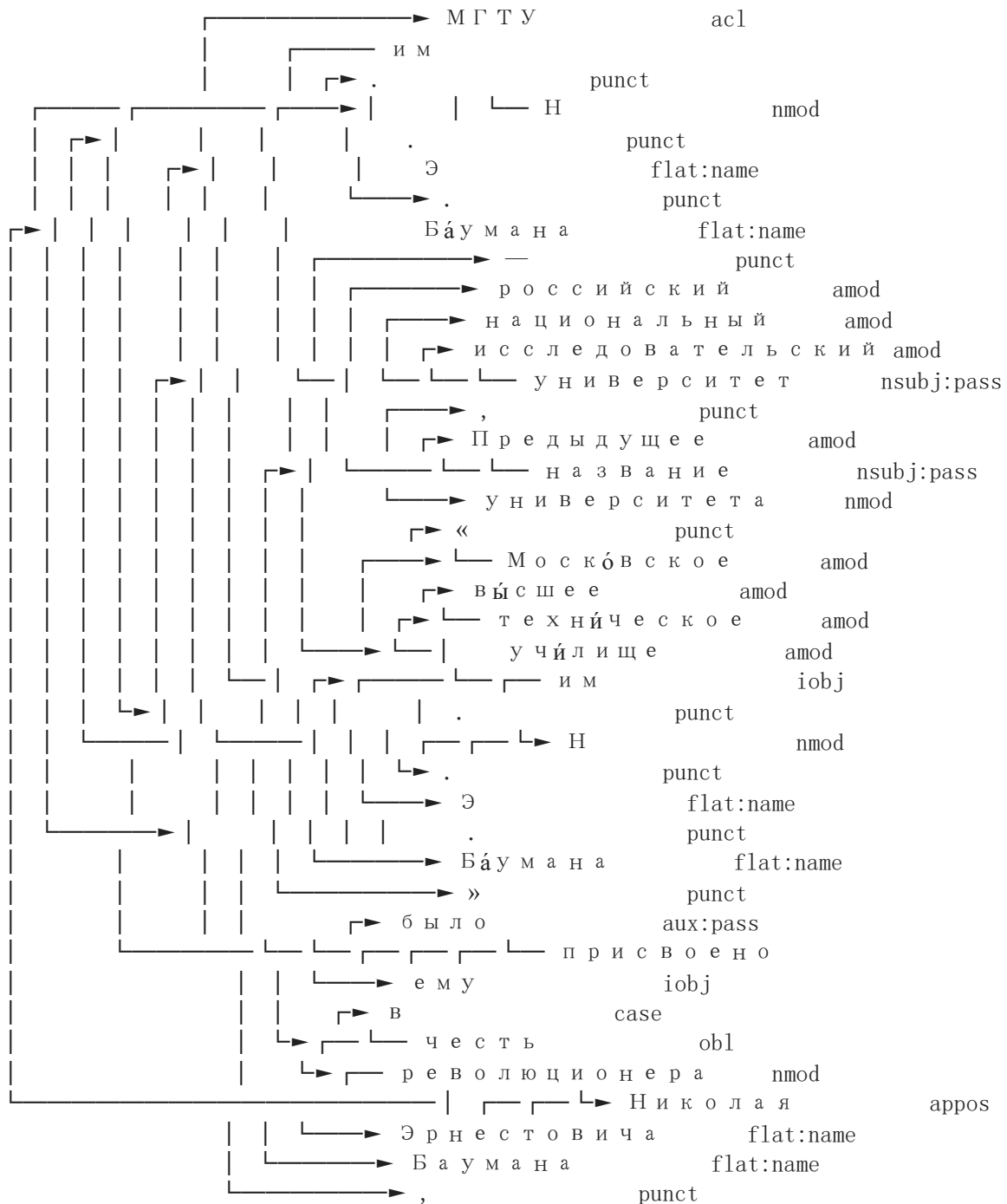
```
n_doc.sents[0].syntax.print()
```







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



▼ Решение задачи классификации текста

```
# Подключение библиотек
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,
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")
```

Способ 1. 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_dataflt = df[df['t']==c]
    # расчет accuracy для заданной метки класса
    temp_acc = accuracy_score(
        temp_dataflt['t'].values,
        temp_dataflt['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, 2, 1, 1, 2, 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
warnings.warn(msg, category=FutureWarning)
```

```
['00',
 '000',
 '0000',
 '00000000004',
 '00000000005',
 '00000000667',
 '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'], cv=5)
            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:
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:
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:
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,
'0000000005': 4, '0000000667': 5, '0000001200': 6,
'0001': 7, '00014': 8, '0002': 9, '0003': 10,
'0005111312': 11, '0005111312nalem': 12,
'00072': 13, '000851': 14, '000rpm': 15,
'000th': 16, '001': 17, '0010': 18, '001004': 19,
'0011': 20, '001211': 21, '0013': 22, '001642': 23,
'001813': 24, '002': 25, '002222': 26,
'002251w': 27, '0023': 28, '002937': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
```

Accuracy = 0.937813339432037

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
'0000000005': 4, '0000000667': 5, '0000001200': 6,
'0001': 7, '00014': 8, '0002': 9, '0003': 10,
'0005111312': 11, '0005111312nalem': 12,
'00072': 13, '000851': 14, '000rpm': 15,
'000th': 16, '001': 17, '0010': 18, '001004': 19,
'0011': 20, '001211': 21, '0013': 22, '001642': 23,
'001813': 24, '002': 25, '002222': 26,
'002251w': 27, '0023': 28, '002937': 29, ...})
```

Модель для классификации - LinearSVC()

Accuracy = 0.9453742497059174

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
'0000000005': 4, '0000000667': 5, '0000001200': 6,
'0001': 7, '00014': 8, '0002': 9, '0003': 10,
'0005111312': 11, '0005111312nalem': 12,
'00072': 13, '000851': 14, '000rpm': 15,
'000th': 16, '001': 17, '0010': 18, '001004': 19,
```

```
'0011': 20, '001211': 21, '0013': 22, '001642': 23,
'001813': 24, '002': 25, '002222': 26,
'002251w': 27, '0023': 28, '002937': 29, ...})
```

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'],
```

```
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())
```

М е т к а	Accuracy
0	0.9290322580645162
1	0.9675090252707581
2	0.9026845637583892
3	0.9245901639344263

Способ 2. Работа с векторными представлениями слов с использованием word2vec

```
import gensim
from gensim.models import word2vec
```

```
!pip install gensim
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/si
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.6.0)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages (from gensim)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (from gensim)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packages (from gensim)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/dist-packages (from gensim)
```

```
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] Package stopwords is already up-to-date!
True

```

```

model_path = '/content/ruscorpora_mystem_cbow_300_2_2015 (1).bin.gz'

```

```

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

```

```

words = ['х о л о д_S', 'м о р о з_S', 'б е р е з а_S', 'с о с н а_S']

```

```

for word in words:
    if word in model:
        print('\nС Л О В О - {}'.format(word))
        print('5 б л и ж а й ш и х с о с е д е й с л о в а :')
        for word, sim in model.most_similar(positive=[word], topn=5):
            print('{} => {}'.format(word, sim))
    else:
        print('С л о в о "{}" н е н а й д е н о в м о д е л и'.format(word))

```

```

С Л О В О - х о л о д_S
5 б л и ж а й ш и х с о с е д е й с л о в а :
с т у ж а_S => 0.7676383852958679
с ы р о с т ь_S => 0.6338975429534912
ж а р а_S => 0.6089427471160889
м о р о з_S => 0.5890367031097412
о з н о б_S => 0.5776054859161377

```

```

С Л О В О - м о р о з_S
5 б л и ж а й ш и х с о с е д е й с л о в а :
с т у ж а_S => 0.6425479650497437
м о р о з е ц_S => 0.5947279930114746
х о л о д_S => 0.5890367031097412
ж а р а_S => 0.5522176623344421
с н е г о п а д_S => 0.5083199143409729

```

```

С Л О В О - б е р е з а_S
5 б л и ж а й ш и х с о с е д е й с л о в а :
с о с н а_S => 0.7943247556686401
т о п о л ь_S => 0.7562226057052612
д у б_S => 0.7440178394317627
д е р е в о_S => 0.7373415231704712
к л е н_S => 0.7105200290679932

```

```

С Л О В О - с о с н а_S
5 б л и ж а й ш и х с о с е д е й с л о в а :

```

```
б е р е з а _S => 0.7943247556686401
д е р е в о _S => 0.7581434845924377
л и с т в е н н и ц а _S => 0.747814953327179
д у б _S => 0.7412480711936951
е л ь _S => 0.7363824248313904
```

▼ Находим близость между словами и строим аналогии

```
print(model.similarity('с о с н а _S', 'б е р е з а _S'))
```

```
0.7943247
```

```
print(model.most_similar(positive=['х о л о д _S', 'с т у ж а _S'], negative=['м о р о з _S']))
```

```
[('с ы р о с т ь _S', 0.5040211081504822), ('с т ы л о с т ь _S', 0.46336129307746887), ('г о л о д _S', 0.4581434845924377), ('ж е л т о с т ь _S', 0.4581434845924377), ('ж е л т о с т ь _S', 0.4581434845924377)]
```

▼ 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] Package stopwords is already up-to-date!
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'.
```

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

```
CPU times: user 5.62 s, sys: 36.3 ms, total: 5.66 s
```

```
Wall time: 4.81 s
```

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

```
('voltage', 0.9900681972503662), ('circuit', 0.9896678924560547), ('high', 0.988849997520446
```

```
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)
```

▼ Проверка качества работы модели word2vec

```
class EmbeddingVectorizer(object):
    """
    Для текста усредним вектора входящих в него сл
    """
    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_dataflt = df[df['t']==c]
    # расчет accuracy для заданной метки класса
    temp_acc = accuracy_score(
        temp_dataflt['t'].values,
        temp_dataflt['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 = 1000
X_train = corpus[:boundary]
X_test = corpus[boundary:]
y_train = newsgroups['target'][:boundary]
y_test = newsgroups['target'][boundary:]

%%time
sentiment(EmbeddingVectorizer(model_dz.wv), LogisticRegression(C=5.0))

```

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning
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:

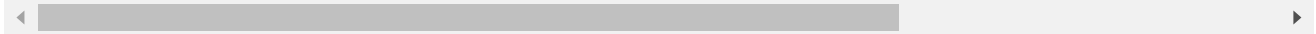
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```

extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Метка Accuracy
0 0.8005698005698005
1 0.9046153846153846

```

```
2      0.7391304347826086
3      0.7381615598885793
CPU times: user 1.33 s, sys: 336 ms, total: 1.67 s
Wall time: 1.73 s
```



Результаты:

Модель CountVectorizer

М е т к а	Accuracy
0	0.9290322580645162
1	0.9675090252707581
2	0.9026845637583892
3	0.9245901639344263

Модель word2vec

М е т к а	Accuracy
0	0.8233618233618234
1	0.9015384615384615
2	0.736231884057971
3	0.7214484679665738

Выводы

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

Результаты, полученные с помощью word2vec не очень хорошие, скорее всего здесь нестандартность лексики ещё больше влияет на работу уже предобученной на более-менее формальных корпусах модели. Короткие неформальные сообщения скорее всего требуют немного других подходов.

