

▼ Лабораторная работа №5

"Предобработка и классификация текстовых данных"

Выполнила: Пинская Н.М.

Группа: ИУ5-21М

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/
Collecting natasha
  Downloading natasha-1.4.0-py3-none-any.whl (34.4 MB)
    |████████████████████████████████████████| 34.4 MB 87.6 MB/s
Collecting slovnet>=0.3.0
  Downloading slovnet-0.5.0-py3-none-any.whl (49 kB)
    |████████████████████████████████████████| 49 kB 7.3 MB/s
Collecting razdel>=0.5.0
  Downloading razdel-0.5.0-py3-none-any.whl (21 kB)
Collecting pymorphy2
  Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
    |████████████████████████████████████████| 55 kB 3.8 MB/s
Collecting yargy>=0.14.0
  Downloading yargy-0.15.0-py3-none-any.whl (41 kB)
    |████████████████████████████████████████| 41 kB 111 kB/s
Collecting navec>=0.9.0
  Downloading navec-0.10.0-py3-none-any.whl (23 kB)
Collecting ipymarkup>=0.8.0
  Downloading ipymarkup-0.9.0-py3-none-any.whl (14 kB)
Collecting 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 (f
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from navec>=0.9.0->nat
Requirement already satisfied: docopt>=0.6 in /usr/local/lib/python3.7/dist-packages (from pymorphy2->
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 55.1 MB/s
Building wheels for collected packages: intervaltree
  Building wheel for intervaltree (setup.py) ... done
  Created wheel for intervaltree: filename=intervaltree-3.1.0-py2.py3-none-any.whl size=26119 sha256=9
  Stored in directory: /root/.cache/pip/wheels/16/85/bd/1001cbb46dcfb71c2001cd7401c6fb250392f22a81ce37
Successfully built intervaltree
Installing collected packages: pymorphy2-dicts-ru, dawg-python, razdel, pymorphy2, navec, intervaltree
  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

```

```
!pip install razdel
```

```

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
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 google 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 = '''Эти змееволосые дамочки уже начали раздражать Перси.
Они должны были умереть еще три дня назад, когда он сбросил на них ящик с шарами для боулинга в «Баргин-Мар»
Перси убивал их и своими глазами видел, как они обращаются в прах, но эти гнусные тетки неизменно возвращали
Он взобрался на вершину холма и перевел дух. Сколько времени прошло с тех пор, как он прикончил их в последи
В последние дни Перси почти не спал. Ел он то, что удавалось стянуть по дороге, – жевательную конфету из той
Перси до сих пор был жив только потому, что две змееволосые дамочки (они называли себя горгонами) тоже, похи

text2 = 'Сам факт Посещения является наиболее важным открытием не только за истекшие тринадцать лет, но и за

test_text = 'Ранним майским утром к гостинице «Дубки» подкатил светло-серый автомобиль. Распахнулась дверца.

# # выделим тестовое сообщение, с которым затем будем выполнять задачи предобработки текста
# test_val = 100
# texts = df_class['content']
# test_text = texts.iloc[test_val]
# test_text

```

▼ 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]   Unzipping tokenizers/punkt.zip.
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

5
['Ранним майским утром к гостинице «Дубки» подкатил светло-серый автомобиль.',
 'Распахнулась дверца, из машины выскочил человек с трубкой в зубах.',
 'Увидев приветливые лица, букеты цветов, он смущённо улыбнулся.',
 'Это был профессор Громов.',
 'Почётный гость конгресса кибернетиков приехал из Синегорска, сибирского научного городка, и, как все
```



Токенизация по словам:

```
nltk_tk_1 = nltk.WordPunctTokenizer()
nltk_tk_1.tokenize(test_text)

'»',
'подкатил'
```



```
подкатил',
'светло',
'-',
'серый',
'автомобиль',
'.',
'Распахнулась',
'дверца',
',',
'из',
'машины',
'выскочил',
'человек',
'с',
'трубкой',
'в',
'зубах',
'.',
'Увидев',
'приветливые',
'лица',
',',
'букеты',
'цветов',
',',
'он',
'смущённо',
'улыбнулся',
'.',
'Это',
'был',
'профессор',
'Громов',
'.',
'Почётный',
'гость',
'конгресса',
'кибернетиков',
'приехал',
'из',
'Синегорска',
',',
'сибирского',
'научного',
'городка',
',',
'и',
',',
'как',
'всегда',
',',
'решил',
'остановиться',
'в',
'«',
'Дубках',
'».']
```

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(1142, 1144, 'не'),
Substring(1145, 1146, 'в'),
Substring(1147, 1156, 'состоянии'),
```

```
Substring(1157, 1160, 'его'),
Substring(1161, 1166, 'убить'),
Substring(1166, 1167, '.'),
Substring(1168, 1170, 'Их'),
Substring(1171, 1176, 'когти'),
Substring(1177, 1179, 'не'),
Substring(1180, 1189, 'оставляли'),
Substring(1190, 1195, 'следа'),
Substring(1196, 1198, 'на'),
Substring(1199, 1202, 'его'),
Substring(1203, 1207, 'коже'),
Substring(1207, 1208, '.'),
Substring(1209, 1213, 'Если'),
Substring(1214, 1217, 'они'),
Substring(1218, 1226, 'пытались'),
Substring(1227, 1230, 'его'),
Substring(1231, 1238, 'укусить'),
Substring(1239, 1240, '-'),
Substring(1241, 1245, 'зубы'),
Substring(1246, 1247, 'у'),
Substring(1248, 1251, 'них'),
Substring(1252, 1260, 'ломались'),
Substring(1260, 1261, '.'),
Substring(1262, 1264, 'Но'),
Substring(1265, 1270, 'Перси'),
Substring(1271, 1274, 'уже'),
Substring(1275, 1278, 'был'),
Substring(1279, 1281, 'на'),
Substring(1282, 1289, 'пределе'),
Substring(1289, 1290, '.'),
Substring(1291, 1296, 'Скоро'),
Substring(1297, 1299, 'он'),
Substring(1300, 1308, 'свалится'),
Substring(1309, 1311, 'от'),
Substring(1312, 1321, 'истощения'),
Substring(1321, 1322, ','),
Substring(1323, 1324, 'а'),
Substring(1325, 1330, 'тогда'),
Substring(1330, 1331, '...'),
Substring(1332, 1336, 'хоть'),
Substring(1337, 1340, 'его'),
Substring(1341, 1342, 'и'),
Substring(1343, 1349, 'трудно'),
Substring(1350, 1355, 'убить'),
Substring(1355, 1356, ','),
Substring(1357, 1364, 'горгоны'),
Substring(1365, 1371, 'найдут'),
Substring(1372, 1378, 'способ'),
Substring(1378, 1379, '.'),
Substring(1380, 1382, 'Он'),
Substring(1383, 1384, 'в'),
Substring(1385, 1389, 'этом'),
Substring(1390, 1392, 'не'),
Substring(1393, 1403, 'сомневался'),
Substring(1403, 1404, '.')]

```

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

```
'не',
'в',
'состоянии',
'его',
'убить',
'.',
'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',  
'и',  
'трудно',  
'убить',  
'...',  
'горгоны',  
'найдут',  
'способ',  
'...',  
'Он',  
'в',  
'этом',  
'не',  
'сомневался',  
'...']
```

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

```
[Substring(0, 52, 'Эти змееволосые дамочки уже начали раздражать Перси.'),  
 Substring(53,  
    171,  
    'Они должны были умереть еще три дня назад, когда он сбросил на них ящик с шарами для боули  
 Substring(172,  
    282,  
    'Они должны были отдать концы два дня назад, после того как он переехал их полицейским авто  
 Substring(283,  
    376,  
    'И уж точно они должны были сдохнуть, когда он отрезал им головы сегодня утром в Тилден-пар  
 Substring(377,  
    496,  
    'Перси убивал их и своими глазами видел, как они обращаются в прах, но эти гнусные тетki не  
 Substring(497, 547, 'Он, похоже, даже не мог надолго от них оторваться.'),  
 Substring(548, 592, 'Он взобрался на вершину холма и перевел дух.'),  
 Substring(593,  
    663,  
    'Сколько времени прошло с тех пор, как он прикончил их в последний раз?'),  
 Substring(664, 683, 'Часа два, наверное.'),  
 Substring(684, 738, 'Кажется, они теперь не умирают больше чем на два часа...'),  
 Substring(739, 775, 'В последние дни Перси почти не спал.'),
```

```

Substring(776,
    943,
    'Ел он то, что удавалось стянуть по дороге, – жевательную конфету из торгового автомата, че
Substring(944,
    1019,
    'Одежда его порвалась, местами обгорела и вся была заляпана слюной монстров.'),
Substring(1020,
    1167,
    'Перси до сих пор был жив только потому, что две змееволосые дамочки (они называли себя гор
Substring(1168, 1208, 'Их когти не оставляли следа на его коже.'),
Substring(1209, 1261, 'Если они пытались его укусить – зубы у них ломались.'),
Substring(1262, 1290, 'Но Перси уже был на пределе.'),
Substring(1291,
    1379,
    'Скоро он свалится от истощения, а тогда... хоть его и трудно убить, горгоны найдут способ.')
Substring(1380, 1404, 'Он в этом не сомневался.')]

```

```

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

```

```

(['Эти змееволосые дамочки уже начали раздражать Перси.',
 'Они должны были умереть еще три дня назад, когда он сбросил на них ящик с шарами для боулинга в «Ба
 'Они должны были отдать концы два дня назад, после того как он переехал их полицейским автомобилем в
 'И уж точно они должны были сдохнуть, когда он отрезал им головы сегодня утром в Тилден-парке.',
 'Перси убивал их и своими глазами видел, как они обращаются в прах, но эти гнусные тетki неизменно в
 'Он, похоже, даже не мог надолго от них оторваться.',
 'Он взобрался на вершину холма и перевел дух.',
 'Сколько времени прошло с тех пор, как он прикончил их в последний раз?',
 'Часа два, наверное.',
 'Кажется, они теперь не умирают больше чем на два часа...',
 'В последние дни Перси почти не спал.',
 'Ел он то, что удавалось стянуть по дороге, – жевательную конфету из торгового автомата, черствый бу
 'Одежда его порвалась, местами обгорела и вся была заляпана слюной монстров.',
 'Перси до сих пор был жив только потому, что две змееволосые дамочки (они называли себя горгонами) т
 '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 и трудно убить, горгоны найдут способ.',
 'Он в этом не сомневался.'],
19)

```

Этот вариант токенизации нужен для последующей обработки

```

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

```

```

n_sen_chunk = n_sentence(text)
n_sen_chunk

```

```

..... ,
'только',
'потому',
',',
'что',
'две',
'змееволосые',
'дамочки',
'(',
'они',
'называли',
'себя',
'горгонами',
')',
'тоже',
.

```

```
,',
'похоже',
',',
'оказались',
'не',
'в',
'состоянии',
'его',
'убить',
'.'],
['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',
'и',
'трудно',
'убить',
',',
'горгоны',
'найдут',
'способ',
'.'],
['Он', 'в', 'этом', 'не', 'сомневался', '.']]
```

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

```
n_sen_chunk_2
```

```
'но',
'и',
'за',
'все',
'время',
'существования',
'человечества',
'.'],
['Не', 'так', 'уж', 'важно', ',', 'кто', 'были', 'эти', 'пришельцы', '.'],
['Неважно',
',',
'откуда',
'они',
'прибыли',
',',
'зачем',
'прибыли',
',',
'почему',
'так',
'недолго',
'пробыли',
'и',
'квла'.
```



```

'уны',
'девались',
'потом',
'.'],
['Важно',
'то',
',',
'что',
'теперь',
'человечество',
'твердо',
'знает',
':',
'оно',
'не',
'одинок',
'во',
'Вселенной',
'.'],
['Бюсь',
',',
'что',
'Институту',
'Внеземных',
'Культур',
'уже',
'никогда',
'больше',
'не',
'повезет',
'сделать',
'более',
'фундаментальное',
'открытие',
'.']]

```

▼ 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_))

подкатил - NUM - ROOT
светло - ADJ - compound
- - PUNCT - punct

```

- - PUNCT - punct
 серый - NOUN - compound
 автомобиль - NOUN - dobj
 . - PUNCT - punct
 Распахнулась - PROPN - compound
 дверца - NOUN - ROOT
 , - PUNCT - punct
 из - PROPN - compound
 машины - PROPN - compound
 выскочил - PROPN - compound
 человек - PROPN - compound
 с - PROPN - compound
 трубкой - PROPN - appos
 в - PROPN - compound
 зубах - NOUN - conj
 . - PUNCT - punct
 Увидев - PROPN - compound
 приветливые - PROPN - compound
 лица - PROPN - nsubj
 , - PUNCT - punct
 букеты - PROPN - compound
 цветов - PROPN - appos
 , - PUNCT - punct
 он - PROPN - ROOT
 смущённо - PROPN - compound
 улыбнулся - PROPN - pobj
 . - PUNCT - punct
 Это - PROPN - compound
 был - PROPN - nsubj
 профессор - NOUN - ROOT
 Громов - PROPN - appos
 . - PUNCT - punct
 Почётный - PROPN - compound
 гость - NOUN - compound
 конгресса - PROPN - compound
 кибернетиков - PROPN - compound
 приехал - PROPN - compound
 из - PROPN - compound
 Синегорска - PROPN - ROOT
 , - PUNCT - punct
 сибирского - PROPN - compound
 научного - PROPN - compound
 городка - PROPN - conj
 , - PUNCT - punct
 и - PROPN - conj
 , - PUNCT - punct
 как - PROPN - compound
 всегда - PROPN - appos
 , - PUNCT - punct
 решил - PROPN - nsubj
 остановиться - PROPN - ROOT
 в - PROPN - det
 « - NOUN - compound
 Дубках - PROPN - dobj
 » - PUNCT - punct
 . - PUNCT - punct

Natasha

```

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)

```

Downloaded from <http://ajph.org/> on November 10, 2015

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

```
[print_pos(x) for x in n_text_markup]
```

[illegible]

```

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

,
важно - ADJ|Degree=Pos|Gender=Neut|Number=Sing|Variant=Short
, - PUNCT
кто - PRON|Case=Nom
были - AUX|Aspect=Imp|Mood=Ind|Number=Plur|Tense=Past|VerbForm=Fin|Voice=Act
эти - DET|Case=Nom|Number=Plur
пришельцы - NOUN|Animacy=Anim|Case=Nom|Gender=Masc|Number=Plur
. - PUNCT
Неважно - ADV|Degree=Pos
, - PUNCT
откуда - ADV|Degree=Pos
они - PRON|Case=Nom|Number=Plur|Person=3
прибыли - VERB|Aspect=Perf|Mood=Ind|Number=Plur|Tense=Past|VerbForm=Fin|Voice=Act
, - PUNCT
зачем - ADV|Degree=Pos
прибыли - VERB|Aspect=Perf|Mood=Ind|Number=Plur|Tense=Past|VerbForm=Fin|Voice=Act
, - PUNCT
почему - ADV|Degree=Pos
так - ADV|Degree=Pos
недолго - ADV|Degree=Pos
пробыли - VERB|Aspect=Perf|Mood=Ind|Number=Plur|Tense=Past|VerbForm=Fin|Voice=Act
и - CONJ
куда - ADV|Degree=Pos
девались - VERB|Aspect=Perf|Mood=Ind|Number=Plur|Person=3|Tense=Fut|VerbForm=Fin|Voice=Act
потом - ADV|Degree=Pos
. - PUNCT
Важно - ADJ|Degree=Pos|Gender=Neut|Number=Sing|Variant=Short
то - PRON|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
, - PUNCT
что - CONJ
теперь - ADV|Degree=Pos
человечество - NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
твердо - ADV|Degree=Pos
знает - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
: - PUNCT
оно - PRON|Case=Nom|Gender=Neut|Number=Sing|Person=3
не - PART|Polarity=Neg
одиноко - ADV|Degree=Pos
во - ADP
Вселенной - PROPON|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
. - PUNCT
Боясь - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=1|Tense=Pres|VerbForm=Fin|Voice=Mid
, - PUNCT

что - CONJ
Институту - PROPON|Animacy=Inan|Case=Dat|Gender=Masc|Number=Sing
Внеземных - ADJ|Case=Gen|Degree=Pos|Number=Plur
Культур - PROPON|Animacy=Anim|Case=Gen|Gender=Masc|Number=Plur
уже - ADV|Degree=Pos
никогда - ADV|Degree=Pos
больше - ADV|Degree=Cmp
не - PART|Polarity=Neg
повезет - VERB|Aspect=Perf|Mood=Ind|Number=Sing|Person=3|Tense=Fut|VerbForm=Fin|Voice=Act
сделать - VERB|Aspect=Perf|VerbForm=Inf|Voice=Act
более - ADV|Degree=Cmp
фундаментальное - ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Gender=Neut|Number=Sing
открытие - NOUN|Animacy=Inan|Case=Acc|Gender=Neut|Number=Sing
. - PUNCT
[None, None, None, None, None]

```

▼ Лемматизация

Spacy

for token in spacy test:

```
print(token, token.lemma, token.lemma_)
```

```
подкатил 15889380949265465631 подкатил
светло 10403397017051258397 светло
- 9153284864653046197 -
серый 293004198024914368 серый
автомобиль 15690161355021272870 автомобиль
. 12646065887601541794 .
Распахнулась 15035480498679999475 Распахнулась
дверца 8347868081549859214 дверца
, 2593208677638477497 ,
из 12183146372738139588 из
машины 3574605372934986972 машины
выскочил 5965066048200716231 выскочил
человек 7568775649844232870 человек
с 5863529159893111856 с
трубкой 1271733369831796785 трубкой
в 15939375860797385675 в
зубах 14663926127362700491 зубах
. 12646065887601541794 .
Увидев 6663573993513292520 Увидев
приветливые 10667803453304488442 приветливые
лица 16981797281768476569 лица
, 2593208677638477497 ,
букеты 16972093218046880272 букеты
цветов 14705294268059991987 цветов
, 2593208677638477497 ,
он 7004339974413567607 он
смущённо 2889550843480277746 смущённо
улыбнулся 8846738030547974580 улыбнулся
. 12646065887601541794 .
Это 6166395895414128982 Это
был 647519599663180051 был
профессор 18094890929148750994 профессор
Громов 2850279829997148809 Громов
. 12646065887601541794 .
Почётный 1467091420853914851 Почётный
гость 2806713705785072619 гость
конгресса 7106661838159002761 конгресса
кибернетиков 13534226636079181325 кибернетиков
приехал 7222518665697276410 приехал
из 12183146372738139588 из
Синегорска 5451488223726882187 Синегорска
, 2593208677638477497 ,
сибирского 13276346775445981270 сибирского
научного 8738816006371375365 научного
городка 9201303923069401183 городка
, 2593208677638477497 ,
и 15015917632809974589 и
, 2593208677638477497 ,
как 13039644133688645009 как
всегда 10633257961924346802 всегда
, 2593208677638477497 ,
решил 18242611503851558175 решил
остановиться 5707673703628731919 остановиться
в 15939375860797385675 в
« 1373379536816847062 «
Дубках 6201723511842712495 Дубках
» 5342463183827332990 »
. 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}

```

```

'прежде': 'прежде',
'прежде': 'прежде',
'прикончил': 'прикончить',
'прошло': 'пройти',
'пытались': 'пытаться',
'раз': 'раз',
'раздражать': 'раздражать',
'с': 'с',

'сбросил': 'сбросить',
'свалится': 'свалиться',
'своими': 'свой',
'сдохнуть': 'сдохнуть',
'себя': 'себя',
'сегодня': 'сегодня',
'сих': 'сей',
'следа': 'след',
'слюной': 'слюна',
'сомневался': 'сомневаться',
'состоянии': 'состояние',
'спал': 'спать',
'способ': 'способ',
'стянуть': 'стянуть',
'так': 'так',
'теперь': 'теперь',
'тетки': 'тетка',
'тех': 'тот',
'то': 'тот',
'тогда': 'тогда',
'того': 'тот',
'тоже': 'тоже',
'только': 'только',
'торгового': 'торговый',
'точно': 'точно',
'три': 'три',
'трудно': 'трудный',
'у': 'у',
'убивал': 'убивать',
'убить': 'убить',
'удавалось': 'удаваться',
'уж': 'уж',
'уже': 'уже',
'укусить': 'укусить',
'умереть': 'умереть',
'умирают': 'умирать',
'утром': 'утро',
'холма': 'холм',
'хоть': 'хоть',
'хотя': 'хотя',
'часа': 'час',
'чем': 'чем',
'черствый': 'черствый',
'что': 'что',
'шарами': 'шар',
'эти': 'этот',
'этом': 'это',
'ящик': 'ящик',
'_': '_',
'...': '...'

```

```

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

{',': ',',
 '.': '.',
 ':': ':',
 'Бюкс': 'бояться',
 'Важно': 'важный',
 'Внеземных': 'внеземной',
 'Вселенной': 'вселенная',
 'Институту': 'институт',
 'Культур': 'культура',
 'Не': 'не',
 'Неважно': 'неважно',
 'Посещения': 'посещение',
 'Сам': 'сам',
 'более': 'более',
 'больше': 'большой',
 'были': 'быть',
 'важно': 'важный',
 'важным': 'важный',
 'во': 'в',
 'время': 'время',
 'все': 'весь',
 'девались': 'деваться',
 'за': 'за',
 'зачем': 'зачем',
 'знает': 'знать',
 'и': 'и',
 'истекшие': 'истечь',
 'кто': 'кто',
 'куда': 'куда',
 'лет': 'год',
 'наиболее': 'наиболее',
 'не': 'не',
 'недолго': 'недолго',
 'никогда': 'никогда',
 'но': 'но',
 'одинок': 'одинок',
 'они': 'они',
 'оно': 'оно',
 'открытие': 'открытие',
 'открытием': 'открытие',
 'откуда': 'откуда',
 'повезет': 'повезти',
 'потом': 'потом',
 'почему': 'почему',
 'прибыли': 'прибыть',
 'пришельцы': 'пришелец',
 'пробыли': 'пробыть',
 'сделать': 'сделать',
 'существования': 'существование',
 'так': 'так',
 'твердо': 'твердо',
 'теперь': 'теперь',
 'то': 'тот',
 'только': 'только',
 'тринадцать': 'тринадцать',
 'уж': 'уж',
 'уже': 'уже',
 'факт': 'факт',

```

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

Spacy

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

Дубки ORG
выскочил человек PERSON
лица ORG
Это ORG
Громов ORG
Почётный гость PERSON
кибернетиков приехал PERSON
Синегорска PRODUCT
сибирского научного городка ORG
```

```
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)
```

Ранним майским утром к гостинице « Дубки **ORG** » подкатил светло-серый автомобиль. Распахнулась дверца, из машины выскочил человек **PERSON** с трубкой в зубах. Увидев приветливые лица **ORG** , букеты цветов, он смущённо улыбнулся. Это **ORG** был профессор Громов **ORG** . Почётный гость **PERSON** конгресса кибернетиков приехал **PERSON** из Синегорска **PRODUCT** , сибирского научного городка **ORG** , и, как всегда, решил остановиться в «Дубках».

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='Сам факт Посещения является наиболее важным открытием не только за истекшие тринадцать лет,
  spans=[Span(
    start=9,
```



```

        stop=18,
        type='LOC'
    ), Span(
        start=361,
        stop=388,
        type='ORG'
    )
]
)

```



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

Сам факт Посещения является наиболее важным открытием не только за
 LOC—————
 истекшие тринадцать лет, но и за все время существования человечества.
 Не так уж важно, кто были эти пришельцы. Неважно, откуда они прибыли,
 зачем прибыли, почему так недолго пробыли и куда девались потом.
 Важно то, что теперь человечество твердо знает: оно не одиноко во
 Вселенной. Боюсь, что Институту Внеземных Культур уже никогда больше
 ORG—————
 не повезет сделать более фундаментальное открытие.

▼ 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

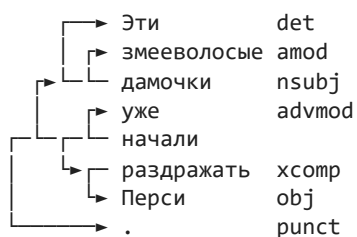
```
from natasha import NewsSyntaxParser
```

```
emb = NewsEmbedding()
```

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

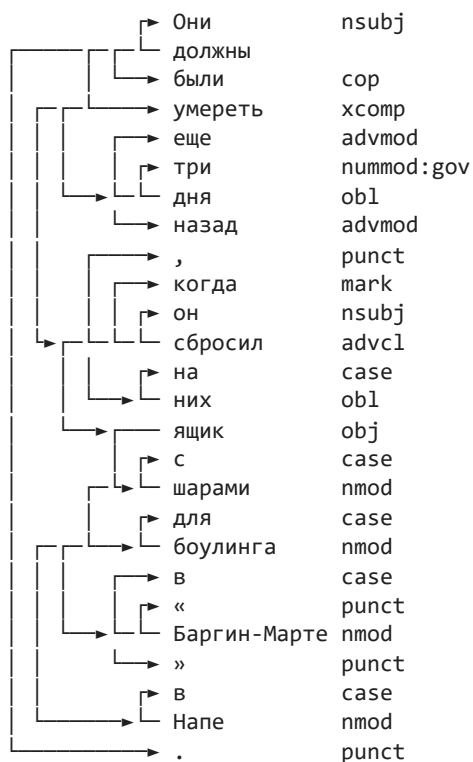
```
n_doc.parse_syntax(syntax_parser)
```

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



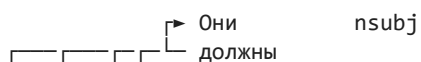
```
n_doc.parse_syntax(syntax_parser)
```

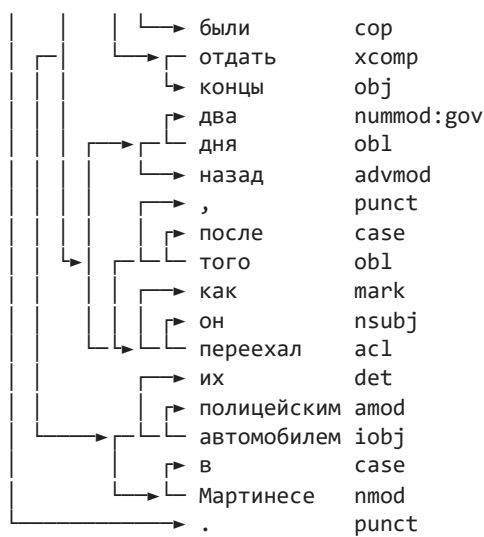
```
n_doc.sents[1].syntax.print()
```



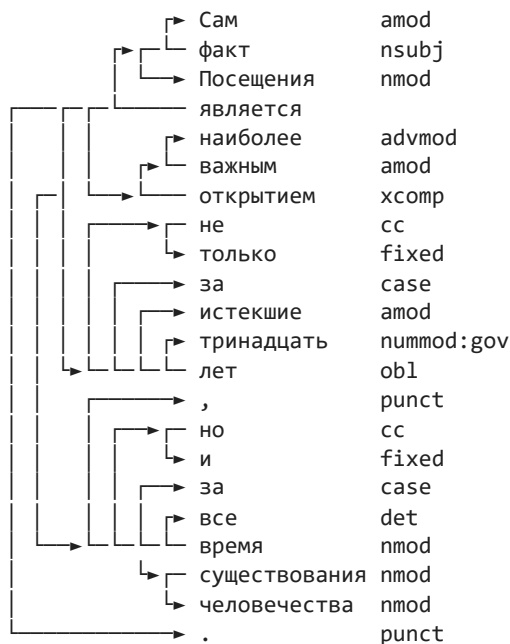
```
n_doc.parse_syntax(syntax_parser)
```

```
n_doc.sents[2].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, median_absolute_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 ассурасу для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Ассурасу 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 ассурасу 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 ассурасу 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])
```

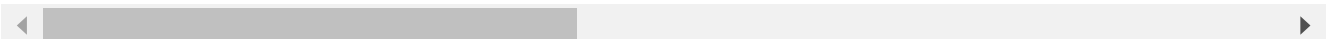
```
[1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2,
```



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

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_fe
warnings.warn(msg, category=FutureWarning)
```

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



▼


```

'000th': 16, '001': 17, '0010': 18, '001004': 19,
'0011': 20, '001211': 21, '0013': 22, '001642': 23,
'001813': 24, '002': 25, '002222': 26,
'002251w': 27, '0023': 28, '002937': 29, ...})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.6655358653541747
=====

```

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

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

```

def sentiment(v, c):
    model = Pipeline(
        [("vectorizer", v),
         ("classifier", c)])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print_accuracy_score_for_classes(y_test, y_pred)

```

```
sentiment(CountVectorizer(), LinearSVC())
```

Метка	Accuracy
0	0.9290322580645162
1	0.9675090252707581
2	0.9026845637583892
3	0.9245901639344263

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/simple/
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.6.0)
Requirement already satisfied: numpy>=1.11.3 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: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (from gensim) (1.1
Requirement already satisfied: scipy>=0.18.1 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]   Unzipping corpora/stopwords.zip.
True

model_path = '/content/ruscorpora_mystem_cbow_300_2_2015.bin.gz'

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

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

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

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

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

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

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

```

▼ Находим близость между словами и строим аналогии

```

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

0.7943247

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

```



```
[('сырость_S', 0.5040211081504822), ('стылость_S', 0.46336129307746887), ('голод_S', 0.460481643676757
```

▼ 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]
['sore',
'spot',
'spot',
'size',
'nickel',
'one',
'testicles',
'bottom',
'side',
'knots',
'lumps',

'little',
'sore',
'spot',
'says',
'reminds',
'bruise',
'feels',
'recollection',
'hitting',
'anything',
'like',
'would',
'cause']
```

```

    cause ,
    'bruise',
    'asssures',
    'remember',
    'something',
    'like',
    'clues',
    'might',
    'somewhat',
    'hypochondriac',
    'sp',
    'sure',
    'gonna',
    'die',
    'thanks',
    'opinions',
    'expressed',
    'necessarily',
    'university',
    'north',
    'carolina',
    'chapel',
    'hill',
    'campus',
    'office',
    'information',
    'technology',
    'experimental',
    'bulletin',
    'board',
    'service',
    'internet',
    'launchpad',
    'unc',
    'edu']]

```

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

```

CPU times: user 4.68 s, sys: 66 ms, total: 4.75 s
Wall time: 2.98 s

```

```

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

```

```
[('circuit', 0.9914703369140625), ('voltage', 0.9902374744415283), ('using', 0.9898467063903809), ('im
```

```

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 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 ассурасу для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Ассурасу 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 ассурасу 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 ассурасу 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: lbfg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

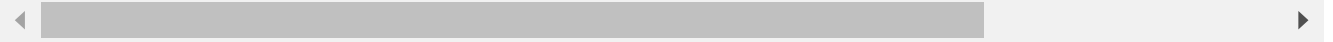
Increase the number of iterations (max_iter) or scale the data as shown in:

<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         0.8233618233618234  
1         0.9015384615384615  
2         0.736231884057971  
3         0.7214484679665738  
CPU times: user 798 ms, sys: 205 ms, total: 1 s  
Wall time: 754 ms
```



Результаты:

Модель CountVectorizer

Метка	Accuracy
0	0.9290322580645162
1	0.9675090252707581
2	0.9026845637583892
3	0.9245901639344263

Модель word2vec

Метка	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.

✓ 0s completed at 4:00 PM

