

```
text = '''Повседневная практика показывает, что начало повседневной работы по формированию пс
text2 = 'Разнообразный и богатый опыт реализация намеченного плана развития требует от нас с
```

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

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

## ▼ Задача токенизации

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

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

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

```
['Повседневная',
 'практика',
 'показывает',
 ',',
 'что',
 'начало',
 'повседневной',
```

Сохранено



```
формирования',
 'позиции',
```

```
'в',
'значительной',
'степени',
'обуславливает',
'создание',
'модели',
'развития',
'?',
'Таким',
'образом',
',',
'курс',
'на',
'социально',
'-',
'ориентированный',
'национальный',
'проект',
'обеспечивает',
'актуальность',
'всесторонне',
'сбалансированных',
'нововведений',
'.',
'Дорогие',
'друзья',
',',
'реализация',
'намеченного',
'плана',
'развития',
'влечет',
'за',
'собой',
'процесс',
'внедрения',
'и',
'модернизации',
'существующих',
'финансовых',
'и',
'административных',
'условий',
'.']
```

```
nltk_tk_sents = nltk.tokenize.sent_tokenize(text2)
print(len(nltk_tk_sents))
nltk_tk_sents
```

1

1. Развитием и богатый опыт реализации намеченного плана развития требует от нас сис

Сохранено



## ▼ Частеречная разметка

```
!pip install natasha
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting natasha
  Downloading natasha-1.4.0-py3-none-any.whl (34.4 MB)
    |████████████████████████████████████████| 34.4 MB 151 kB/s
Collecting slovnet>=0.3.0
  Downloading slovnet-0.5.0-py3-none-any.whl (49 kB)
    |██████████████████████████████████████| 49 kB 5.5 MB/s
Collecting yargy>=0.14.0
  Downloading yargy-0.15.0-py3-none-any.whl (41 kB)
    |██████████████████████████████████████| 41 kB 104 kB/s
Collecting pymorphy2
  Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
    |██████████████████████████████████████| 55 kB 2.9 MB/s
Collecting razdel>=0.5.0
  Downloading razdel-0.5.0-py3-none-any.whl (21 kB)
Collecting ipymarkup>=0.8.0
  Downloading ipymarkup-0.9.0-py3-none-any.whl (14 kB)
Collecting navec>=0.9.0
  Downloading navec-0.10.0-py3-none-any.whl (23 kB)
Collecting 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/d
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from na
Requirement already satisfied: docopt>=0.6 in /usr/local/lib/python3.7/dist-packages (f
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 24.0 MB/s
Collecting dawg-python>=0.7.1
  Downloading DAWG_Python-0.7.2-py2.py3-none-any.whl (11 kB)
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
  Stored in directory: /root/.cache/pip/wheels/16/85/bd/1001cbb46dcfb71c2001cd7401c6fb2
Successfully built intervaltree
Installing collected packages: pymorphy2-dicts-ru, dawg-python, razdel, pymorphy2, nave
  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
```

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

Сохранено



```
from slovnet import Morph
```

```
# Файл необходимо скачать по ссылке https://github.com/natasha/navec#downloads
```

```
navec = Navec.load('navec_news_v1_1B_250K_300d_100q.tar')
```

```
# Файл необходимо скачать по ссылке https://github.com/natasha/slovnet#downloads
```

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

```
morph_res = n_morph.navec(navec)
```

```
def print_pos(markup):
```

```
    for token in markup.tokens:
```

```
        print('{} - {}'.format(token.text, token.tag))
```

```
def n_sentenize(text):
```

```
    n_sen_chunk = []
```

```
    for sent in sentenize(text):
```

```
        tokens = [_.text for _ in tokenize(sent.text)]
```

```
        n_sen_chunk.append(tokens)
```

```
    return n_sen_chunk
```

```
n_sen_chunk = n_sentenize(text)
```

```
n_sen_chunk
```

```
[['Повседневная',  
  'практика',  
  'показывает',  
  ',',  
  'что',  
  'начало',  
  'повседневной',  
  'работы',  
  'по',  
  'формированию',  
  'позиции',  
  'в',  
  'значительной',  
  'степени',  
  'обуславливает',  
  'создание',  
  'модели',  
  'развития',  
  '?'],  
 ['Таким',  
  'образом',  
  ',',  
  .
```

Сохранено



```
'национальный',  
'проект',
```

```

'обеспечивает',
'актуальность',
'всесторонне',
'сбалансированных',
'нововведений',
'.'],
['Дорогие',
'друзья',
',',
'реализация',
'намеченного',
'плана',
'развития',
'влечет',
'за',
'собой',
'процесс',
'внедрения',
'и',
'модернизации',
'существующих',
'финансовых',
'и',
'административных',
'условий',
'.']]

```

```

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

```

```

Повседневная - ADJ|Case=Nom|Degree=Pos|Gender=Fem|Number=Sing
практика - NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing
показывает - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voic
, - PUNCT
что - CONJ
начало - NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
повседневной - ADJ|Case=Gen|Degree=Pos|Gender=Fem|Number=Sing
работы - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
по - ADP
формированию - NOUN|Animacy=Inan|Case=Dat|Gender=Neut|Number=Sing
позиции - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
в - ADP
значительной - ADJ|Case=Loc|Degree=Pos|Gender=Fem|Number=Sing
степени - NOUN|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
обуславливает - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|V
создание - NOUN|Animacy=Inan|Case=Acc|Gender=Neut|Number=Sing
модели - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
развития - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
? - PUNCT
Таким - DET|Case=Ins|Gender=Masc|Number=Sing
образом - NOUN|Animacy=Inan|Case=Ins|Gender=Masc|Number=Sing
на - ADP
социально-ориентированный - ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Gender=Masc|Number=Sin

```

Сохранено



Nom|Gender=Masc|Number=Sing

национальный - ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Gender=Masc|Number=Sing  
 проект - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing  
 обеспечивает - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act  
 актуальность - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Sing  
 всесторонне - ADV|Degree=Pos  
 сбалансированных - VERB|Aspect=Perf|Case=Gen|Number=Plur|Tense=Past|VerbForm=Part|Voice=Act  
 нововведений - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Plur  
 . - PUNCT  
 Дорогие - ADJ|Case=Nom|Degree=Pos|Number=Plur  
 друзья - NOUN|Animacy=Anim|Case=Nom|Gender=Masc|Number=Plur  
 , - PUNCT  
 реализация - NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing  
 намеченного - VERB|Aspect=Perf|Case=Gen|Gender=Masc|Number=Sing|Tense=Past|VerbForm=Part|Voice=Act  
 плана - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing  
 развития - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing  
 влечет - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act  
 за - ADP  
 собой - PRON|Case=Ins  
 процесс - NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing  
 внедрения - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing  
 и - CCONJ  
 модернизации - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing  
 существующих - VERB|Aspect=Imp|Case=Gen|Number=Plur|Tense=Pres|VerbForm=Part|Voice=Act  
 финансовых - ADJ|Case=Gen|Degree=Pos|Number=Plur  
 и - CCONJ  
 административных - ADJ|Case=Gen|Degree=Pos|Number=Plur  
 условий - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Plur  
 . - PUNCT  
 [None, None, None]



```
n_sen_chunk_2 = n_sentence_tokenize(text2)
```

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

Разнообразный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing  
 и - CCONJ  
 богатый - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing  
 опыт - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing  
 реализация - NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing  
 намеченного - VERB|Aspect=Perf|Case=Gen|Gender=Masc|Number=Sing|Tense=Past|VerbForm=Part|Voice=Act  
 плана - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing  
 развития - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing  
 требует - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act  
 от - ADP  
 нас - PRON|Case=Gen|Number=Plur|Person=1  
 системного - ADJ|Case=Gen|Degree=Pos|Gender=Masc|Number=Sing  
 анализа - NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing  
 дальнейших - ADJ|Case=Gen|Degree=Pos|Number=Plur  
 в - CCONJ  
 на - ADP  
 системы - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing  
 массового - ADJ|Case=Gen|Degree=Pos|Gender=Neut|Number=Sing

Сохранено



участия - NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing  
.  
[None]

## ▼ Лемматизация

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

```
def n_lemmatize(text):  
    emb = NewsEmbedding()  
    morph_tagger = NewsMorphTagger(emb)  
    segmenter = Segmenter()  
    morph_vocab = MorphVocab()  
    doc = Doc(text)  
    doc.segment(segmenter)  
    doc.tag_morph(morph_tagger)  
    for token in doc.tokens:  
        token.lemmatize(morph_vocab)  
    return doc
```

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

```
{',': ',',  
'.': '.',  
'?': '?',  
'Дорогие': 'дорогой',  
'Повседневная': 'повседневный',  
'Таким': 'такой',  
'административных': 'административный',  
'актуальность': 'актуальность',  
'в': 'в',  
'влечет': 'влечь',  
'внедрения': 'внедрение',  
'всесторонне': 'всесторонне',  
'друзья': 'друг',  
'за': 'за',  
'значительной': 'значительный',  
'и': 'и',  
'курс': 'курс',  
'модели': 'модель',  
'модернизации': 'модернизация',  
'на': 'на',  
'намеченного': 'наметить',
```

Сохранено



```
'обеспечивает': 'обеспечивать',
```

```

'образом': 'образ',
'обуславливает': 'обуславливать',
'плана': 'план',
'по': 'по',
'повседневной': 'повседневный',
'позиции': 'позиция',
'показывает': 'показывать',
'практика': 'практика',
'проект': 'проект',
'процесс': 'процесс',
'работы': 'работа',
'развития': 'развитие',
'реализация': 'реализация',
'сбалансированных': 'сбалансировать',
'собой': 'себя',
'создание': 'создание',
'социально-ориентированный': 'социально-ориентированный',
'степени': 'степень',
'существующих': 'существовать',
'условий': 'условие',
'финансовых': 'финансовый',
'формированию': 'формирование',
'что': 'что'}

```

```

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

```

```

{'.': '.',
 'Разнообразный': 'разнообразный',
 'анализа': 'анализ',
 'богатый': 'богатый',
 'дальнейших': 'дальнейший',
 'и': 'и',
 'массового': 'массовый',
 'намеченного': 'наметить',
 'направлений': 'направление',
 'нас': 'мы',
 'опыт': 'опыт',
 'от': 'от',
 'плана': 'план',
 'развитая': 'развить',
 'развития': 'развитие',
 'реализация': 'реализация',
 'системного': 'системный',
 'системы': 'система',
 'требуется': 'требовать',
 'участия': 'участие'}

```

## ▼ Выделение (распознавание) именованных сущностей

Сохранено



```
from slovnet import NER
```



```

from ipymarkup import show_span_ascii_markup as show_markup

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

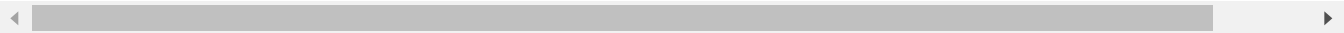
ner_res = ner.navec(navec)

text3 = 'Москва - столица России, по преданию ее основал князь Юрий Долгорукий в 1147 году.'

markup_ner = ner(text3)
markup_ner

SpanMarkup(
  text='Москва - столица России, по преданию ее основал князь Юрий Долгорукий в 1147
  spans=[Span(
    start=0,
    stop=6,
    type='LOC'
  ), Span(
    start=17,
    stop=23,
    type='LOC'
  ), Span(
    start=54,
    stop=69,
    type='PER'
  )]
)

```



```

show_markup(markup_ner.text, markup_ner.spans)

```

```

Москва - столица России, по преданию ее основал князь Юрий Долгорукий
LOC—— LOC—— PER—————
в 1147 году.

```

## ▼ Разбор предложения

```

from natasha import NewsSyntaxParser

emb = NewsEmbedding()
syntax_parser = NewsSyntaxParser(emb)

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

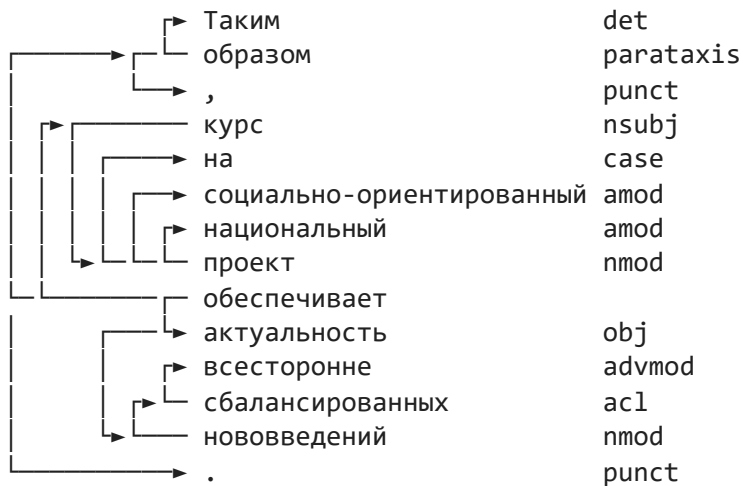
```

Сохранено





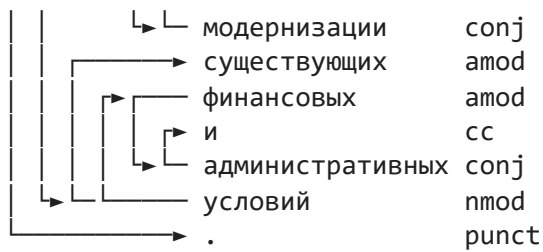
```
n_doc.parse_syntax(syntax_parser)
n_doc.sents[1].syntax.print()
```



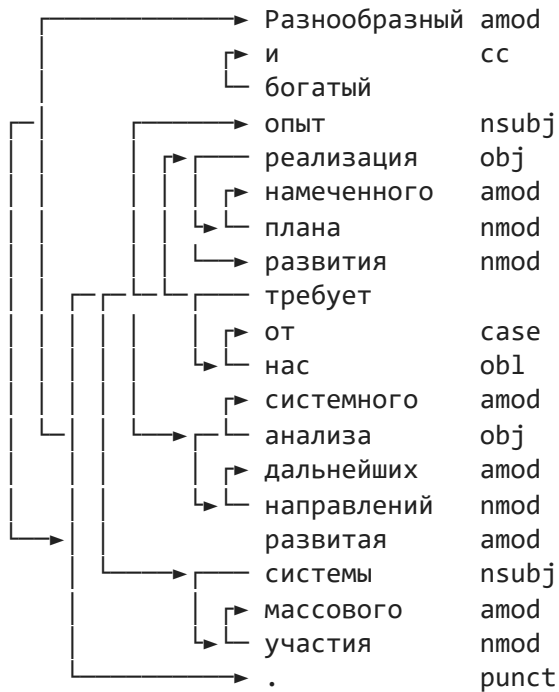
```
n_doc.parse_syntax(syntax_parser)
n_doc.sents[2].syntax.print()
```



Сохранено



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



```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error,
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
train_test_split
```

Сохранено



```
from sklearn.datasets import fetch_20newsgroups
```

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline  
sns.set(style="ticks")
```

## ▼ Векторизация текста на основе модели "мешка слов"

```
categories = ["rec.autos", "rec.sport.hockey", "sci.crypt", "sci.space"]  
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):
```

Сохранено



ого 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 - 37036

```

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

```

```

jake=19872
rambler=27980
eng=14514
sun=32456
com=10748
jason=19918
cockroft=10613
subject=32275
re=28120

```

## ▼ Использование класса CountVectorizer

```

test_features = vocabVect.transform(data)
test_features

```

```

<2382x37036 sparse matrix of type '<class 'numpy.int64'>'
  with 393370 stored elements in Compressed Sparse Row format>

```

```
test_features.todense()
```

```

matrix([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]])

```

Сохранено



1()

37036

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

[1, 1, 1, 1, 2, 1, 1, 1, 1, 7, 1, 2, 1, 1, 1, 1, 1, 8, 2, 1, 1, 1, 10, 1, 1, 1, 1, 1
```

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

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:
  warnings.warn(msg, category=FutureWarning)
['00',
 '000',
 '0000',
 '00000',
 '000000',
 '0000000',
 '00000000',
 '00000000b',
 '00000001',
 '00000001b',
 '00000010']
```

## 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'], scor
            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)
```

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

extra\_warning\_msg= LOGISTIC\_SOLVER\_CONVERGENCE\_MSG,

Сохранено



packages/sklearn/linear\_model/\_logistic.py:818: Converge  
REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

<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](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: Converge  
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](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,  
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,  
      '000000': 4, '00000000': 5, '00000000b': 6,  
      '00000001': 7, '00000001b': 8, '00000010': 9,  
      '00000010b': 10, '00000011': 11, '00000011b': 12,  
      '00000100': 13, '00000100b': 14, '00000101': 15,  
      '00000101b': 16, '00000110': 17, '00000110b': 18,  
      '00000111': 19, '00000111b': 20, '00001000': 21,  
      '00001000b': 22, '00001001': 23, '00001001b': 24,  
      '00001010': 25, '00001010b': 26, '00001011': 27,  
      '00001011b': 28, '00001100': 29, ...})
```

Модель для классификации - LogisticRegression(C=3.0)

Accuracy = 0.971872376154492

=====

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,  
      '000000': 4, '00000000': 5, '00000000b': 6,  
      '00000001': 7, '00000001b': 8, '00000010': 9,  
      '00000010b': 10, '00000011': 11, '00000011b': 12,  
      '00000100': 13, '00000100b': 14, '00000101': 15,  
      '00000101b': 16, '00000110': 17, '00000110b': 18,  
      '00000111': 19, '00000111b': 20, '00001000': 21,  
      '00001000b': 22, '00001001': 23, '00001001b': 24,  
      '00001010': 25, '00001010b': 26, '00001011': 27,  
      '00001011b': 28, '00001100': 29, ...})
```

Модель для классификации - LinearSVC()

Accuracy = 0.9748110831234257

=====

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,  
      '000000': 4, '00000000': 5, '00000000b': 6,  
      '00000001': 7, '00000001b': 8, '00000010': 9,  
      '00000010b': 10, '00000011': 11, '00000011b': 12,  
      '00000100': 13, '00000100b': 14, '00000101': 15,  
      '00000101b': 16, '00000110': 17, '00000110b': 18,  
      '00000111': 19, '00000111b': 20, '00001000': 21,  
      '00001000b': 22, '00001001': 23, '00001001b': 24,  
      '00001010': 25, '00001010b': 26, '00001011': 27,  
      '00001011b': 28, '00001100': 29, ...})
```

Модель для классификации - KNeighborsClassifier()

Accuracy = 0.7065491183879092

Сохранено



## Разделим выборку на обучающую и тестовую и

```
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())
```

Метка	Accuracy
0	0.973421926910299
1	0.9765886287625418
2	0.9419795221843004
3	0.9832214765100671

## Работа с векторными представлениями слов с использованием word2vec

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

```
model_path = 'ruscorpora_mystem_cbow_300_2_2015.bin.gz'
```

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

```
words = ['жар_S', 'тепло_S', 'вода_S', 'лёд_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('{} не в модели'.format(word))
```

Сохранено

СЛОВО - жар\_S



```
5 ближайших соседей слова:
жара_S => 0.5225024819374084
зной_S => 0.5192716717720032
озноб_S => 0.508800745010376
пламень_S => 0.49908268451690674
холод_S => 0.49906834959983826
```

СЛОВО - тепло\_S

```
5 ближайших соседей слова:
теплота_S => 0.6763811111450195
теплый_A => 0.6017489433288574
тепло_ADV => 0.5947409868240356
прохлада_S => 0.5492426156997681
согреть_V => 0.494480699300766
```

СЛОВО - вода\_S

```
5 ближайших соседей слова:
вод_S => 0.6678440570831299
водичка_S => 0.639014482498169
влага_S => 0.5951642990112305
водица_S => 0.5687029361724854
струя_S => 0.5454239249229431
Слово "лёд_S" не найдено в модели
```

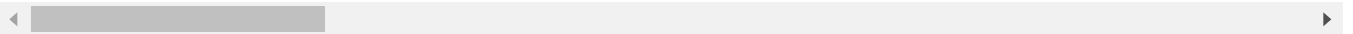
## ▼ Находим близость между словами и строим аналогии

```
print(model.similarity('жар_S', 'тепло_S'))
```

```
0.45076987
```

```
print(model.most_similar(positive=['жар_S', 'теплота_S'], negative=['тепло_S']))
```

```
[('горячность_S', 0.4428492784500122), ('страстность_S', 0.43567872047424316), ('ожесто
```



## Обучим word2vec на наборе данных

### ▼ "fetch\_20newsgroups"

```
import re
import pandas as pd
import numpy as np
from typing import Dict, Tuple
```

Сохранено



```
accuracy_score, balanced_accuracy_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.pipeline import Pipeline
from nltk import WordPunctTokenizer
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
```

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

```
categories = ["rec.autos", "rec.sport.hockey", "sci.crypt", "sci.space"]
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]
```

```
pick ,
'shadow',
'r',
'node',
'gts',
'org',
'see',
'real',
'see',
'transparent',
'see',
' erased'],
['rins',
'ryukoku',
'ac',
'jp',
'william',
'reiken',
'subject',
'nuclear',
'waste',
'organization'.
```

Сохранено



```
see ,
'japan',
'lines'.
```

```

'greedy',
'little',
'oil',
'companies',
'blame',
'oil',
'companies',
'supply',
'demand',
'created',
'everyone',
'else',
'planet',
'run',
'faults',
'ok',
'creation',
'oil',
'producing'.

```

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

```

CPU times: user 8.92 s, sys: 33.2 ms, total: 8.95 s
Wall time: 5.5 s

```

# Проверим, что модель обучилась

```
print(model_imdb.wv.most_similar(positive=['find'], topn=5))
```

```
[('reverse', 0.9620099067687988), ('could', 0.9579818844795227), ('easy', 0.95592808723
```

```
def sentiment_2(v, c):
```

```

    model = Pipeline(
        [("vectorizer", v),
         ("classifier", c)])

```

```
    model.fit(X_train, y_train)
```

Сохранено



```
    return model.predict_proba(X_test)[0,1]
```

## ▼ 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_dataflt = df[df['t']==c]
        # расчет ассурасу для заданной метки класса
        temp_acc = accuracy_score(
            temp_dataflt['p'], temp_dataflt['t'])
        res[c] = temp_acc
    return res
```

Сохранено



Словарь

```
def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))
```

# Обучающая и тестовая выборки

boundary = 1500

X\_train = corpus[:boundary]

X\_test = corpus[boundary:]

y\_train = newsgroups['target'][:boundary]

y\_test = newsgroups['target'][boundary:]

sentiment\_2(EmbeddingVectorizer(model\_imdb.wv), LogisticRegression(C=5.0))

/usr/local/lib/python3.7/dist-packages/sklearn/linear\_model/\_logistic.py:818: Convergen  
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](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

extra\_warning\_msg=\_LOGISTIC\_SOLVER\_CONVERGENCE\_MSG,

Метка	Accuracy
0	0.9279279279279279
1	0.9389671361502347
2	0.9417040358744395
3	0.9241071428571429

*Как видно из результатов проверки качества моделей, лучшее качество показала модель на основе CountVectorizer*

Сохранено



✓ 1 сек. выполнено в 17:59



Сохранено

