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

Лабораторная работа №5
«Создание рекомендательной модели»

ИСПОЛНИТЕЛЬ:

Елизаров Олег Олегович
Группа ИУ5-21М

_____

"__"_____2022 г.

Москва 2022

Цель лабораторной работы: изучение методов предобработки и классификации текстовых данных.

Требования к отчету:

Отчет по лабораторной работе должен содержать:

1. титульный лист;
2. описание задания;
3. текст программы;
4. экранные формы с примерами выполнения программы.

Задание:

1. Для произвольного предложения или текста решите следующие задачи:

- Токенизация.
- Частеречная разметка.
- Лемматизация.
- Выделение (распознавание) именованных сущностей.
- Разбор предложения.

1. 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".

In [3]:

```
text = '''Мне кажется, если нужно что-то забыть, то забудешь рано или поздно. И не стоит на
text2 = 'Это будет равноценный обмен. Я отдам тебе половину своей жизни, а ты мне половину
```

In [4]:

```
!pip install natasha
```

```
Requirement already satisfied: natasha in d:\ml\lib\site-packages (1.4.0)
Requirement already satisfied: navec>=0.9.0 in d:\ml\lib\site-packages (from
natasha) (0.10.0)
Requirement already satisfied: razdel>=0.5.0 in d:\ml\lib\site-packages (from
natasha) (0.5.0)
Requirement already satisfied: yargy>=0.14.0 in d:\ml\lib\site-packages (from
natasha) (0.15.0)
Requirement already satisfied: ipymarkup>=0.8.0 in d:\ml\lib\site-packages (f
rom natasha) (0.9.0)
Requirement already satisfied: slovnet>=0.3.0 in d:\ml\lib\site-packages (fro
m natasha) (0.5.0)
Requirement already satisfied: pymorphy2 in d:\ml\lib\site-packages (from nat
asha) (0.9.1)
Requirement already satisfied: intervaltree>=3 in d:\ml\lib\site-packages (fr
om ipymarkup>=0.8.0->natasha) (3.1.0)
Requirement already satisfied: sortedcontainers<3.0,>=2.0 in d:\ml\lib\site-p
ackages (from intervaltree>=3->ipymarkup>=0.8.0->natasha) (2.4.0)
Requirement already satisfied: numpy in d:\ml\lib\site-packages (from navec>=
0.9.0->natasha) (1.20.3)
Requirement already satisfied: docopt>=0.6 in d:\ml\lib\site-packages (from p
ymorphy2->natasha) (0.6.2)
Requirement already satisfied: dawg-python>=0.7.1 in d:\ml\lib\site-packages
(from pymorphy2->natasha) (0.7.2)
Requirement already satisfied: pymorphy2-dicts-ru<3.0,>=2.4 in d:\ml\lib\site
-packages (from pymorphy2->natasha) (2.4.417127.4579844)
```

#Задача токенизации

In [5]:

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

In [6]:

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

Out[6]:

```
[Substring(0, 3, 'Мне'),
 Substring(4, 11, 'кажется'),
 Substring(11, 12, ','),
 Substring(13, 17, 'если'),
 Substring(18, 23, 'нужно'),
 Substring(24, 30, 'что-то'),
 Substring(31, 37, 'забыть'),
 Substring(37, 38, ','),
 Substring(39, 41, 'то'),
 Substring(42, 50, 'забудешь'),
 Substring(51, 55, 'рано'),
 Substring(56, 59, 'или'),
 Substring(60, 66, 'поздно'),
 Substring(66, 67, '.'),
 Substring(68, 69, 'И'),
 Substring(70, 72, 'не'),
 Substring(73, 78, 'стоит'),
 Substring(79, 81, 'на'),
 Substring(82, 86, 'этом'),
 Substring(87, 100, 'зацикливаться'),
 Substring(100, 101, '.'),
 Substring(102, 108, 'Знаешь'),
 Substring(108, 109, ','),
 Substring(110, 113, 'как'),
 Substring(114, 120, 'бывает'),
 Substring(120, 121, '?'),
 Substring(122, 125, 'Чем'),
 Substring(126, 132, 'больше'),
 Substring(133, 139, 'хочешь'),
 Substring(140, 149, 'выбросить'),
 Substring(150, 156, 'что-то'),
 Substring(157, 159, 'из'),
 Substring(160, 166, 'головы'),
 Substring(166, 167, ','),
 Substring(168, 171, 'тем'),
 Substring(172, 178, 'дольше'),
 Substring(179, 182, 'оно'),
 Substring(183, 184, 'в'),
 Substring(185, 191, 'памяти'),
 Substring(192, 200, 'остаётся'),
 Substring(200, 201, '.')]
```

In [7]:

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

Out[7]:

```
['Мне',
 'кажется',
 ',',
 'если',
 'нужно',
 'что-то',
 'забыть',
 ',',
 'то',
 'забудешь',
 'рано',
 'или',
 'поздно',
 '.',
 'И',
 'не',
 'стоит',
 'на',
 'этом',
 'зацикливаться',
 '.',
 'Знаешь',
 ',',
 'как',
 'бывает',
 '?',
 'Чем',
 'больше',
 'хочешь',
 'выбросить',
 'что-то',
 'из',
 'головы',
 ',',
 'тем',
 'дольше',
 'оно',
 'в',
 'памяти',
 'остаётся',
 '.']
```

In [8]:

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

Out[8]:

```
[Substring(0,
           67,
           'Мне кажется, если нужно что-то забыть, то забудешь рано или поздн
о.'),
 Substring(68, 101, 'И не стоит на этом зацикливаться.'),
 Substring(102, 121, 'Знаешь, как бывает?'),
 Substring(122,
           201,
           'Чем больше хочешь выбросить что-то из головы, тем дольше оно в па
мяти остаётся.')]
```

In [9]:

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

Out[9]:

```
(['Мне кажется, если нужно что-то забыть, то забудешь рано или поздно.',
  'И не стоит на этом зацикливаться.',
  'Знаешь, как бывает?',
  'Чем больше хочешь выбросить что-то из головы, тем дольше оно в памяти оста
ётся.'],
 4)
```

In [10]:

```
# Этот вариант токенизации нужен для последующей обработки
def n_sentenize(text):
    n_sen_chunk = []
    for sent in sentenize(text):
        tokens = [_.text for _ in tokenize(sent.text)]
        n_sen_chunk.append(tokens)
    return n_sen_chunk
```

In [11]:

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

Out[11]:

```
[['Мне',
  'кажется',
  ',',
  'если',
  'нужно',
  'что-то',
  'забыть',
  ',',
  'то',
  'забудешь',
  'рано',
  'или',
  'поздно',
  '.'],
 ['И', 'не', 'стоит', 'на', 'этом', 'зацикливаться', '.'],
 ['Знаешь', ',', 'как', 'бывает', '?'],
 ['Чем',
  'больше',
  'хочешь',
  'выбросить',
  'что-то',
  'из',
  'головы',
  ',',
  'тем',
  'дольше',
  'оно',
  'в',
  'памяти',
  'остаётся',
  '.']]
```

In [12]:

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

Out[12]:

```
[['Это', 'будет', 'равноценный', 'обмен', '.'],
 ['Я',
  'отдам',
  'тебе',
  'половину',
  'своей',
  'жизни',
  ',',
  'а',
  'ты',
  'мне',
  'половину',
  'своей',
  '.']]
```

#Частеречная разметка

In [13]:

```
from navec import Navec
from slovnet import Morph
```

In [15]:

```
# Файл необходимо скачать по ссылке https://github.com/natasha/navec#downloads
navec = Navec.load('navec_news_v1_1B_250K_300d_100q.tar')
```

In [16]:

```
# Файл необходимо скачать по ссылке https://github.com/natasha/slovnet#downloads
n_morph = Morph.load('slovnet_morph_news_v1.tar', batch_size=4)
```

In [17]:

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

In [18]:

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

In [19]:

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

```
Мне - PRON|Case=Dat|Number=Sing|Person=1
кажется - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=F
in|Voice=Mid
, - PUNCT
если - SCONJ
нужно - ADJ|Degree=Pos|Gender=Neut|Number=Sing|Variant=Short
что-то - PRON|Case=Acc
забыть - VERB|Aspect=Perf|VerbForm=Inf|Voice=Act
, - PUNCT
то - SCONJ
забудешь - VERB|Aspect=Perf|Mood=Ind|Number=Sing|Person=2|Tense=Fut|VerbForm=
Fin|Voice=Act
рано - ADV|Degree=Pos
или - CCONJ
поздно - ADV|Degree=Pos
. - PUNCT
И - CCONJ
не - PART|Polarity=Neg
стоит - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin
|Voice=Act
на - ADP
этом - DET|Case=Loc|Gender=Masc|Number=Sing
зацикливаться - VERB|Aspect=Imp|VerbForm=Inf|Voice=Mid
. - PUNCT
Знаешь - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=2|Tense=Pres|VerbForm=Fi
n|Voice=Act
, - PUNCT
как - ADV|Degree=Pos
бывает - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fi
n|Voice=Act
? - PUNCT
Чем - SCONJ
больше - ADV|Degree=Cmp
хочешь - VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=2|Tense=Pres|VerbForm=Fi
n|Voice=Act
выбросить - VERB|Aspect=Perf|VerbForm=Inf|Voice=Act
что-то - PRON|Case=Acc
из - ADP
головы - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
, - PUNCT
тем - SCONJ
дольше - ADV|Degree=Cmp
оно - PRON|Case=Nom|Gender=Neut|Number=Sing|Person=3
в - ADP
памяти - NOUN|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
остаётся - ADJ|Degree=Pos|Gender=Neut|Number=Sing|Variant=Short
. - PUNCT
```

Out[19]:

```
[None, None, None, None]
```

In [20]:

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

```
Это - PRON|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
будет - AUX|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|
Voice=Act
равноценный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
обмен - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
. - PUNCT
Я - PRON|Case=Nom|Number=Sing|Person=1
отдам - VERB|Aspect=Perf|Mood=Ind|Number=Sing|Person=1|Tense=Fut|VerbForm=Fin
|Voice=Act
тебе - PRON|Case=Dat|Number=Sing|Person=2
половину - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Sing
своей - DET|Case=Gen|Gender=Fem|Number=Sing
жизни - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
, - PUNCT
а - CCONJ
ты - PRON|Case=Nom|Number=Sing|Person=2
мне - PRON|Case=Dat|Number=Sing|Person=1
половину - NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Sing
своей - DET|Case=Gen|Gender=Fem|Number=Sing
. - PUNCT
```

Out[20]:

```
[None, None]
```

#Лемматизация

In [21]:

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

In [22]:

```
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
```

In [23]:

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

Out[23]:

```
{'Мне': 'я',
 'кажется': 'казаться',
 ',': ',',
 'если': 'если',
 'нужно': 'нужный',
 'что-то': 'что-то',
 'забыть': 'забыть',
 'то': 'то',
 'забудешь': 'забыть',
 'рано': 'рано',
 'или': 'или',
 'поздно': 'поздно',
 '.': '.',
 'И': 'и',
 'не': 'не',
 'стоит': 'стоить',
 'на': 'на',
 'этом': 'этот',
 'зацикливаться': 'зацикливаться',
 'Знаешь': 'знать',
 'как': 'как',
 'бывает': 'бывать',
 '?': '?',
 'Чем': 'чем',
 'больше': 'большой',
 'хочешь': 'хотеть',
 'выбросить': 'выбросить',
 'из': 'из',
 'головы': 'голова',
 'тем': 'тем',
 'дольше': 'долгий',
 'оно': 'оно',
 'в': 'в',
 'памяти': 'память',
 'остаётся': 'оставаться'}
```

In [24]:

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

Out[24]:

```
{'Это': 'это',
 'будет': 'быть',
 'равноценный': 'равноценный',
 'обмен': 'обмен',
 '.': '.',
 'Я': 'я',
 'отдам': 'отдать',
 'тебе': 'ты',
 'половину': 'половина',
 'своей': 'свой',
 'жизни': 'жизнь',
 ',': ',',
 'а': 'а',
 'ты': 'ты',
 'мне': 'я'}
```

#Выделение (распознавание) именованных сущностей

In [25]:

```
from slovnet import NER
from ipymarkup import show_span_ascii_markup as show_markup
```

In [27]:

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

In [28]:

```
ner_res = ner.navec(navec)
```

In [32]:

```
text3 = text2 + 'Ну а я Олег.'

markup_ner = ner(text3)
markup_ner
```

Out[32]:

```
SpanMarkup(
    text='Это будет равноценный обмен. Я отдам тебе половину своей жизни, а т
ы мне половину своей.Ну а я Олег.',
    spans=[Span(
         start=95,
         stop=99,
         type='PER'
     )]
)
```

In [33]:

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

```
Это будет равноценный обмен. Я отдам тебе половину своей жизни, а ты
мне половину своей.Ну а я Олег.
                          PER─
```

#Разбор предложения

In [34]:

```
from natasha import NewsSyntaxParser
```

In [35]:

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

In [36]:

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

```
             ┌► Мне       iobj
┌─┌───┌───── └─ кажется
│ │   │   ┌──► ,          punct
│ │   │   │ ┌► если       mark
│ │ ┌─└►└─ └─ нужно      advcl
│ │ │ │     ┌► что-то     obj
│ │ │ └──►└─ забыть     csubj
│ │ └─────► ,          punct
│ │         ┌► то         mark
│ └─────►┌─ └─ забудешь   ccomp
│      ┌─└──► рано       advmod
│      │    ┌► или        cc
│      └──►└─ поздно     conj
└────────► .          punct
```

In [37]:

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

```
    ┌──► И              cc
    │ ┌► не             advmod
┌─┌─└─└─ стоит
│ │   ┌► на             case
│ │ ┌►└─ этом           obl
│ └►└─── зацикливаться csubj
└──────► .              punct
```

In [38]:

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

```
     Знаешь
┌──► ,      punct
│  ┌► как    advmod
└──└─ бывает
     ?
```

In [39]:

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

```
┌─────► Это         nsubj
│  ┌──► будет       cop
│  │  ┌► равноценный amod
└──└──└─ обмен
   └──► .           punct
```

In [40]:

```
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")
```

#Векторизация текста на основе модели "мешка слов"

In [42]:

```
categories = ["rec.sport.hockey", "rec.sport.baseball", "sci.crypt","sci.space"]
newsgroups = fetch_20newsgroups(subset='train', categories=categories)
data = newsgroups['data']
```

In [ ]:

```
```

In [49]:

```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

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

In [50]:

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

```
Количество сформированных признаков - 36053
```

In [52]:

```
for i in list(corpusVocab)[1:15]:
    print('{}={}'.format(i, corpusVocab[i]))
```

```
eastgate=13606
world=35502
std=31184
com=10437
mark=21937
bernstein=7838
subject=31563
re=27488
jewish=19518
baseball=7514
players=26024
organization=24729
the=32523
public=26947
```

#Использование класса CountVectorizer

In [53]:

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

Out[53]:

```
<2385x36053 sparse matrix of type '<class 'numpy.int64'>'
        with 390795 stored elements in Compressed Sparse Row format>
```

In [54]:

```
test_features.todense()
```

Out[54]:

```
matrix([[0, 0, 0, ..., 0, 0, 0],
        [2, 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]], dtype=int64)
```

In [55]:

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

Out[55]:

```
36053
```

In [56]:

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

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1,
1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 4, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 3]
```

In [57]:

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

Out[57]:

```
['00',
 '000',
 '0000',
 '00000',
 '000000',
 '00000000',
 '00000000b',
 '00000001',
 '00000001b',
 '00000010']
```

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

In [58]:

```
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'], sc
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')
```

In [65]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [66]:

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

```
D:\ml\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWar
ning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
D:\ml\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWar
ning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
D:\ml\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWar
ning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(

Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '000
00': 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 = 0.9639412997903564
===========================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '000
00': 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 = 0.9651991614255765
============================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '000
00': 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 - DecisionTreeClassifier()
Accuracy = 0.8381551362683438
============================
```

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

In [67]:

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

In [68]:

```
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)
```

In [69]:

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

```
Метка    Accuracy
0        0.9401993355481728
1        0.9248366013071896
2        0.9659863945578231
3        0.9623287671232876
```

#Работа с векторными представлениями слов с использованием word2vec

In [72]:

```
!pip install gensim

import gensim
from gensim.models import word2vec
```

```
Collecting gensim
  Downloading gensim-4.2.0-cp39-cp39-win_amd64.whl (23.9 MB)
Requirement already satisfied: scipy>=0.18.1 in d:\ml\lib\site-packages (from
gensim) (1.7.1)
Collecting smart-open>=1.8.1
  Downloading smart_open-6.0.0-py3-none-any.whl (58 kB)
Requirement already satisfied: numpy>=1.17.0 in d:\ml\lib\site-packages (from
gensim) (1.20.3)
Collecting Cython==0.29.28
  Downloading Cython-0.29.28-py2.py3-none-any.whl (983 kB)
Installing collected packages: smart-open, Cython, gensim
  Attempting uninstall: Cython
    Found existing installation: Cython 0.29.24
    Uninstalling Cython-0.29.24:
      Successfully uninstalled Cython-0.29.24
Successfully installed Cython-0.29.28 gensim-4.2.0 smart-open-6.0.0
```

In [75]:

```
model_path = 'web_upos_cbow_300_20_2017.bin.gz'
```

In [76]:

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

In [111]:

```
words = ['покемон_NOUN', 'город_NOUN', 'лениться_VERB', 'пингвин_NOUN']
model.index_to_key
```

Out[111]:

```
['весь_DET',
 'год_NOUN',
 'мочь_VERB',
 'сайт_NOUN',
 'время_NOUN',
 'человек_NOUN',
 'работа_NOUN',
 'новый_ADJ',
 'день_NOUN',
 'также_ADV',
 'товар_NOUN',
 'самый_DET',
 'становиться_VERB',
 'компания_NOUN',
 'первый_ADJ',
 'очень_ADV',
 'информация_NOUN',
 'получать_VERB',
```

In [112]:

```
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 - покемон_NOUN
5 ближайших соседей слова:
покахонтас_NOUN => 0.708166778087616
пожарка_NOUN => 0.573899507522583
пикатить_VERB => 0.5102546215057373
полль_NOUN => 0.49793606996536255
полтергула_NOUN => 0.48930680751800537

СЛОВО - город_NOUN
5 ближайших соседей слова:
столица_NOUN => 0.6723980903625488
район_NOUN => 0.5644219517707825
городок_NOUN => 0.5557301044464111
пригород_NOUN => 0.555604100227356
городской_ADJ => 0.5403953194618225

СЛОВО - лениться_VERB
5 ближайших соседей слова:
лень_NOUN => 0.6229262351989746
неохота_ADV => 0.5809724926948547
бездельничать_VERB => 0.5659275054931641
лень_ADV => 0.52410888671875
влом_NOUN => 0.4994359612464905

СЛОВО - пингвин_NOUN
5 ближайших соседей слова:
тюлень_NOUN => 0.6302710771560669
пингвинчик_NOUN => 0.5950493812561035
горилла_NOUN => 0.5801599025726318
пиранья_NOUN => 0.5800156593322754
лемур_NOUN => 0.5633261799812317
```

#Находим близость между словами и строим аналогии

In [113]:

```
print(model.similarity('человек_NOUN', 'машина_NOUN'))
```

```
0.010995562
```

In [ ]:

```

```

#Обучим word2vec на наборе данных "fetch_20newsgroups"

In [114]:

```
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
[nltk_data]     C:\Users\eliza\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Out[114]:

```
True
```

In [115]:

```
categories = ["rec.sport.hockey", "rec.sport.baseball", "sci.crypt","sci.space"]
newsgroups = fetch_20newsgroups(subset='train', categories=categories)
data = newsgroups['data']
```

In [116]:

```
# 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)
```

In [117]:

```
corpus[:5]
```

Out[117]:

```
[['eastgate',
  'world',
  'std',
  'com',
  'mark',
  'bernstein',
  'subject',
  'jewish',
  'baseball',
  'players',
  'organization',
  'world',
  'public',
  'access',
  'unix',
  'brookline',
  'lines',
  'al',
```

In [118]:

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

```
Wall time: 1.51 s
```

In [133]:

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

```
[('tell', 0.9522036910057068), ('anything', 0.9488185048103333), ('even', 0.9
48540210723877), ('might', 0.9472000598907471), ('way', 0.9470121264457703)]
```

In [134]:

```
def sentiment_2(v, c):
    model = Pipeline(
        [("vectorizer", v),
         ("classifier", c)])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print_accuracy_score_for_classes(y_test, y_pred)
```

#Проверка качества работы модели word2vec

In [135]:

```
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])
```

In [136]:

```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

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

In [137]:

```
# Обучающая и тестовая выборки
boundary = 1500
X_train = corpus[:boundary]
X_test = corpus[boundary:]
y_train = newsgroups['target'][:boundary]
y_test = newsgroups['target'][boundary:]
```

In [146]:

```
sentiment_2(EmbeddingVectorizer(model_imdb.wv), LinearSVC())
```

```
D:\ml\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Libline
ar failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Метка	Accuracy
0	0.8272727272727273
1	0.7982456140350878
2	0.9811320754716981
3	0.9733333333333334

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

In [ ]:

```

```