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

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

Выполнил:
Студент группы ИУ5-21М Ся Бэйбэй

_____

Москва — 2022г.

1. Цель лабораторной работы

Изучение методов предобработки и классификации текстов.

2. 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.

3. Ход выполнения работы

3.1 Для произвольного предложения или текста

Токенизация
NLTK:

```
In [1]: text1 = 'Шэньчжэнь — крупный китайский город, расположенный на юге страны, возл
        text2 = 'Китай или Чжун Го, как его называют сами китайцы, является одной из самы
        text3 = 'Топография Китая очень разнообразна, на его территории имеются высоки

In [2]: #Natural Language Toolkit, 自然语言处理工具包
        import nltk
        nltk.download('punkt')

        [nltk_data] Downloading package punkt to
        [nltk_data]     C:\Users\92883\AppData\Roaming\nltk_data...
        [nltk_data]   Package punkt is already up-to-date!

Out[2]: True

In [3]: #tokenize网络符将一个字符串标记化;切分词;
        from nltk import tokenize
        dir(tokenize)[:18]

Out[3]: ['BlanklineTokenizer',
         'LegalitySyllableTokenizer',
         'LineTokenizer',
         'MWETokenizer',
         'NLTKWordTokenizer',
         'PunktSentenceTokenizer',
         'RegexpTokenizer',
         'ReppTokenizer',
         'SExprTokenizer',
         'SpaceTokenizer',
         'StanfordSegmenter',
         'SyllableTokenizer',
         'TabTokenizer',
         'TextTilingTokenizer',
         'ToktokTokenizer',
         'TreebankWordTokenizer',
         'TweetTokenizer',
         'WhitespaceTokenizer']

In [4]: nltk_tk_1 = nltk.WordPunctTokenizer()
        nltk_tk_1.tokenize(text1)
```

```
In [4]:
nltk_tk_1 = nltk.WordPunctTokenizer()
nltk_tk_1.tokenize(text1)
```

```
Out[4]: ['Шэньчжэнь',
 '—',
 'крупный',
 'китайский',
 'город',
 ',',
 'расположенный',
 'на',
 'юге',
 'страны',
 ',',
 'возле',
 'границы',
 'с',
 'Гонконгом',
 '.',
 'Он',
 'построен',
 'в',
 'устье',
 'Жемчужной',
 'реки',
 ',',
 'на',
 'побережье',
 'Южно',
 '-',
 'Китайского',
 'моря',
 '.',
 'В',
 'мире',
 'Шэньчжэнь',
 'воспринимают',
 'как',
 'экономическое']
```

```
In [5]:
# Токенизация по предложениям
nltk_tk_sents = nltk.tokenize.sent_tokenize(text1)
print(len(nltk_tk_sents))
nltk_tk_sents
```

```
4
```

```
Out[5]: ['Шэньчжэнь — крупный китайский город, расположенный на юге страны, возле границы с Гонконгом.',
 'Он построен в устье Жемчужной реки, на побережье Южно-Китайского моря.',
 'В мире Шэньчжэнь воспринимают как экономическое чудо.',
 'Его называют «городом парков и небоскребов», а также китайской «Силиконовой долиной».']
```

Spacy:

```
In [7]:
from spacy.lang.ru import Russian
import spacy
nlp = spacy.load('ru_core_news_sm')
spacy_text1 = nlp(text1)
spacy_text1
```

```
Out[7]: Шэньчжэнь — крупный китайский город, расположенный на юге страны, возле границы с Гонконгом. Он построен в устье Жемчужной реки, на побережье Южно-Китайского моря. В мире Шэньчжэнь воспринимают как экономическое чудо. Его называют «городом парков и небоскребов», а также китайской «Силиконовой долиной».
```

```
In [8]:
for t in spacy_text1:
    print(t)
```

```
Шэньчжэнь
—
крупный
китайский
город
,
расположенный
на
юге
страны
,
возле
границы
с
Гонконгом
.
Он
построен
в
устье
Жемчужной
реки
,
на
побережье
Ю
```

```
In [9]:
spacy_text2 = nlp(text2)
spacy_text2
```

```
Out[9]: Китай или Чжун Го, как его называют сами китайцы, является одной из самых удивительных и загадочных стран мира
```

```
In [10]:
spacy_text3 = nlp(text3)
spacy_text3
```

```
Out[10]: Топография Китая очень разнообразна, на его территории имеются высокие горы, плато, впадины, пустыни и обширные равнины.
```

Natasha:

In [11]:
```
#pip install razdel
from razdel   import tokenize, sentenize
```

In [12]:
```
n_tok_text1 = list(tokenize(text1))
n_tok_text1
```

Out[12]:
```
[Substring(0, 9, 'Шэньчжэнь'),
 Substring(10, 11, '—'),
 Substring(12, 19, 'крупный'),
 Substring(20, 29, 'китайский'),
 Substring(30, 35, 'город'),
 Substring(35, 36, ','),
 Substring(37, 50, 'расположенный'),
 Substring(51, 53, 'на'),
 Substring(54, 57, 'юге'),
 Substring(58, 64, 'страны'),
 Substring(64, 65, ','),
 Substring(66, 71, 'возле'),
 Substring(72, 79, 'границы'),
 Substring(80, 81, 'с'),
 Substring(82, 91, 'Гонконгом'),
 Substring(91, 92, '.'),
 Substring(93, 95, 'Он'),
 Substring(96, 104, 'построен'),
 Substring(105, 106, 'в'),
 Substring(107, 112, 'устье'),
 Substring(113, 122, 'Жемчужной'),
 Substring(123, 127, 'реки'),
 Substring(127, 128, ','),
 Substring(129, 131, 'на'),
 Substring(132, 141, 'побережье'),
 Substring(142, 157, 'Южно-Китайского'),
 Substring(158, 162, 'моря'),
 Substring(162, 163, '.'),
 Substring(164, 165, 'В'),
 Substring(166, 170, 'мире'),
 Substring(171, 180, 'Шэньчжэнь'),
 Substring(181, 193, 'воспринимают'),
```

In [13]:
```
[_.text for _ in n_tok_text1]
```

Out[13]:
```
['Шэньчжэнь',
 '—',
 'крупный',
 'китайский',
 'город',
 ',',
 'расположенный',
 'на',
 'юге',
 'страны',
 ',',
 'возле',
 'границы',
 'с',
 'Гонконгом',
 '.',
 'Он',
 'построен',
 'в',
 'устье',
 'Жемчужной',
 'реки',
 ',',
 'на',
 'побережье',
 'Южно-Китайского',
 'моря',
 '.',
 'В',
 'мире',
 'Шэньчжэнь',
 'воспринимают',
 'как',
 'экономическое',
 'чудо',
 '.',
```

In [14]:
```
n_sen_text1 = list(sentenize(text1))
n_sen_text1
```

Out[14]:
```
[Substring(0,
           92,
           'Шэньчжэнь — крупный китайский город, расположенный на юге страны, возле границы с Гонконгом.'),
 Substring(93,
           163,
           'Он построен в устье Жемчужной реки, на побережье Южно-Китайского моря.'),
 Substring(164, 217, 'В мире Шэньчжэнь воспринимают как экономическое чудо.'),
 Substring(218,
           303,
           'Его называют «городом парков и небоскребов», а также китайской «Силиконовой долиной».')]
```

In [15]:
```
[_.text for _ in n_sen_text1], len([_.text for _ in n_sen_text1])
```

Out[15]:
```
(['Шэньчжэнь — крупный китайский город, расположенный на юге страны, возле границы с Гонконгом.',
  'Он построен в устье Жемчужной реки, на побережье Южно-Китайского моря.',
  'В мире Шэньчжэнь воспринимают как экономическое чудо.',
  'Его называют «городом парков и небоскребов», а также китайской «Силиконовой долиной».'],
 4)
```

In [16]:
```
# Этот вариант токенизации нужен для последующей обработки
## 进一步处理需要这个标记化选项
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 [17]:
```
n_sen_chunk_1 = n_sentenize(text1)
n_sen_chunk_1
```

Out[17]:
```
[['Шэньчжэнь',
  '—',
  'крупный',
  'китайский',
  'город',
  ',',
  'расположенный',
  'на',
  'юге',
  'страны',
  ',',
  'возле',
  'границы',
  'с',
  'Гонконгом',
  '.'],
 ['Он',
  'построен',
  'в',
  'устье',
  'Жемчужной',
  'реки',
  ',',
  'на',
  'побережье',
  'Южно-Китайского',
  'моря',
```

In [18]:
```
n_sen_chunk_2 = n_sentenize(text2)
n_sen_chunk_2
```

Out[18]:
```
[['Китай',
  'или',
  'Чжун',
  'Го',
  ',',
  'как',
  'его',
  'называют',
  'сами',
  'китайцы',
  ',',
  'является',
  'одной',
  'из',
  'самых',
  'удивительных',
  'и',
  'загадочных',
  'стран',
  'мира']]
```

In [19]:
```
n_sen_chunk_3 = n_sentenize(text3)
n_sen_chunk_3
```

Out[19]:
```
[['Топография',
  'Китая',
  'очень',
  'разнообразна',
  ',',
  'на',
  'его',
  'территории',
  'имеются',
  'высокие',
  'горы',
  ',',
  'плато',
  ',',
  '.',
```

Частеречная разметка

Spacy:

In [20]:
```
#部分标记  Частеречная разметка  Spacy
for token in spacy_text1:
    print('{} - {} - {}'.format(token.text, token.pos_, token.dep_))
```

```
Шэньчжэнь - PROPN - nsubj
— - PUNCT - punct
крупный - ADJ - amod
китайский - ADJ - amod
город - NOUN - ROOT
, - PUNCT - punct
расположенный - ADJ - acl
на - ADP - case
юге - NOUN - obl
страны - NOUN - nmod
, - PUNCT - punct
возле - ADP - case
границы - NOUN - conj
с - ADP - case
Гонконгом - PROPN - nmod
. - PUNCT - punct
Он - PRON - nsubj:pass
построен - VERB - ROOT
в - ADP - case
устье - NOUN - obl
Жемчужной - ADJ - amod
реки - NOUN - nmod
, - PUNCT - punct
на - ADP - case
побережье - NOUN - conj
Южно - ADJ - amod
- - ADJ - amod
Китайского - ADJ - amod
моря - NOUN - nmod
. - PUNCT - punct
В - ADP - case
мире - NOUN - obl
Шэньчжэнь - PROPN - obj
воспринимают - VERB - ROOT
как - SCONJ - case
экономическое - ADJ - amod
чудо - NOUN - obl
```

Natasha:

In [21]:
```
#部分标记  Частеречная разметка  Natasha pip install
from navec import Navec
from slovnet import Morph
```

In [22]:
```
# Файл необходимо скачать по ссылке https://github.com/natasha/navec#downloads
#尝试将此模型用于新闻文本。 它比“hudlit”小两倍，但涵盖了新闻文章中相同的 98% 的单词。
navec = Navec.load('./navec_news_v1_1B_250K_300d_100q.tar')
```

In [23]:
```
# Файл необходимо скачать по ссылке https://github.com/natasha/slovnet#downloads
#针对新闻文章优化的俄语形态标记器
n_morph = Morph.load('./slovnet_morph_news_v1.tar', batch_size=4)
```

In [24]:
```
morph_res = n_morph.navec(navec)
```

In [25]:
```
def print_pos(markup):
    for token in markup.tokens:
        print('{} - {}'.format(token.text, token.tag))
```

In [26]:
```
n_text1_markup = list(_ for _ in n_morph.map(n_sen_chunk_1))
[print_pos(x) for x in n_text1_markup]
```

```
Шэньчжэнь - PROPN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
— - PUNCT
крупный - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
китайский - ADJ|Case=Nom|Degree=Pos|Gender=Masc|Number=Sing
город - NOUN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
, - PUNCT
расположенный - ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Gender=Masc|Number=Sing
на - ADP
юге - NOUN|Animacy=Inan|Case=Loc|Gender=Masc|Number=Sing
страны - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
, - PUNCT
возле - ADP
границы - NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
с - ADP
Гонконгом - PROPN|Animacy=Inan|Case=Ins|Gender=Masc|Number=Sing
. - PUNCT
Он - PRON|Case=Nom|Gender=Masc|Number=Sing|Person=3
построен - VERB|Aspect=Perf|Gender=Masc|Number=Sing|Tense=Past|Variant=Short|VerbForm=Part|Voice=Pass
в - ADP
```

In [27]:
```
n_text2_markup = list(n_morph.map(n_sen_chunk_2))
[print_pos(x) for x in n_text2_markup]
```

```
К и т а й  – PROPN|Animacy=Inan|Case=Nom|Gender=Masc|Number=Sing
и л и  – CCONJ
Ч ж у н  – PROPN|Animacy=Anim|Case=Nom|Gender=Masc|Number=Sing
Г о  – PROPN|Animacy=Anim|Case=Acc|Gender=Masc|Number=Sing
,  – PUNCT
к а к  – SCONJ
е г о  – PRON|Case=Acc|Gender=Masc|Number=Sing|Person=3
н а з ы в а ю т  – VERB|Aspect=Imp|Mood=Ind|Number=Plur|Person=3|Tense=Pres|VerbForm=Fin|Voice=Act
с а м и  – ADJ|Case=Nom|Degree=Pos|Number=Plur
к и т а й ц ы  – NOUN|Animacy=Anim|Case=Nom|Gender=Masc|Number=Plur
,  – PUNCT
я в л я е т с я  – VERB|Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin|Voice=Mid
о д н о й  – NUM|Case=Ins|Gender=Fem|Number=Sing
и з  – ADP
с а м ы х  – ADJ|Case=Gen|Degree=Pos|Number=Plur
у д и в и т е л ь н ы х  – ADJ|Case=Gen|Degree=Pos|Number=Plur
и  – CCONJ
з а г а д о ч н ы х  – ADJ|Case=Gen|Degree=Pos|Number=Plur
с т р а н  – NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Plur
м и р а  – NOUN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
```

Out[27]:
```
[None]
```

In [28]:
```
n_text3_markup = list(n_morph.map(n_sen_chunk_3))
[print_pos(x) for x in n_text3_markup]
```

```
Т о п о г р а ф и я  – NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Sing
К и т а я  – PROPN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
о ч е н ь  – ADV|Degree=Pos
р а з н о о б р а з н а  – ADJ|Degree=Pos|Gender=Fem|Number=Sing|Variant=Short
,  – PUNCT
н а  – ADP
е г о  – DET
т е р р и т о р и и  – NOUN|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
и м е ю т с я  – VERB|Aspect=Imp|Mood=Ind|Number=Plur|Person=3|Tense=Pres|VerbForm=Fin|Voice=Mid
в ы с о к и е  – ADJ|Case=Nom|Degree=Pos|Number=Plur
г о р ы  – NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Plur
,  – PUNCT
п л а т о  – NOUN|Animacy=Inan|Case=Nom|Gender=Neut|Number=Sing
,  – PUNCT
в п а д и н ы  – NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
,  – PUNCT
п у с т ы н и  – NOUN|Animacy=Inan|Case=Nom|Gender=Fem|Number=Plur
```

Лемматизация

Spacy

In [29]:
```
#Л е м м а т и з а ц и я  词形还原
for token in spacy_text1:
      print(token,  token.lemma,  token.lemma_)
```

```
Ш э н ь ч ж э н ь  7185727914262277272  ш э н ь ч ж э н ь
—  13657828488461764581  —
к р у п н ы й  875311894405788432O  к р у п н ы й
к и т а й с к и й  17143932885149659824  к и т а й с к и й
г о р о д  6063391427805833384  г о р о д
,  2593208677638477497  ,
р а с п о л о ж е н н ы й  11348170133137897564  р а с п о л о ж е н н ы й
н а  16191904166009283104  н а
ю г е  5472666064066444577  ю г
с т р а н ы  16360468073211545296  с т р а н а
,  2593208677638477497  ,
в о з л е  14512160503988266662  в о з л е
г р а н и ц ы  15595961654697241156  г р а н и ц а
с  5863529159893111856  с
Г о н к о н г о м  669429918178398540  г о н к о н г
.  12646065887601541794  .
О н  7004339974413567607  о н
п о с т р о е н  4369257707187989914  п о с т р о и т ь
в  15939375860797385675  в
у с т ь е  17149409378721191859  у с т ь е
Ж е м ч у ж н о й  1633469237953584080  ж е м ч у ж н ы й
р е к и  2877229459635558614  р е к а
,  2593208677638477497  ,
н а  16191904166009283104  н а
п о б е р е ж ь е  6486368253002029509  п о б е р е ж ь е
Ю ж н о  703679541340017613S  ю ж н ы й
–  9153284864653046197  –
К и т а й с к о г о  17143932885149659824  к и т а й с к и й
м о р я  9548742372896973111  м о р е
.  12646065887601541794  .
В  15939375860797385675  в
м и р е  3830696184003690382  м и р
Ш э н ь ч ж э н ь  7185727914262277272  ш э н ь ч ж э н ь
в о с п р и н и м а ю т  12354405791645288502  в о с п р и н и м а т ь
к а к  13039644133688645009  к а к
э к о н о м и ч е с к о е  3234076322849381152  э к о н о м и ч е с к и й
ц у л о  1158023793715634766O  ц у л о
```

Natasha

In [30]:
```
from natasha import Doc,  Segmenter,  NewsEmbedding,  NewsMorphTagger,  MorphVocab
```

In [31]:
```
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 [32]:
```
n_doc1 = n_lemmatize(text1)
{_.text: _.lemma for _ in n_doc1.tokens}
```

Out[32]:
```
{'Ш э н ь ч ж э н ь': 'ш э н ь ч ж э н ь',
 '—': '—',
 'к р у п н ы й': 'к р у п н ы й',
 'к и т а й с к и й': 'к и т а й с к и й',
 'г о р о д': 'г о р о д',
 ',': ',',
 'р а с п о л о ж е н н ы й': 'р а с п о л о ж и т ь',
 'н а': 'н а',
 'ю г е': 'ю г',
 'с т р а н ы': 'с т р а н а',
 'в о з л е': 'в о з л е',
 'г р а н и ц ы': 'г р а н и ц а',
 'с': 'с',
 'Г о н к о н г о м': 'г о н к о н г',
 '.': '.',
 'О н': 'о н',
 'п о с т р о е н': 'п о с т р о и т ь',
 'в': 'в',
 'у с т ь е': 'у с т ь е',
 'Ж е м ч у ж н о й': 'ж е м ч у ж н ы й',
 'р е к и': 'р е к а',
 'п о б е р е ж ь е': 'п о б е р е ж ь е',
 'Ю ж н о - К и т а й с к о г о': 'ю ж н о - к и т а й с к и й',
 'м о р я': 'м о р е',
 'В': 'в'
```

```
In [33]: n_doc2 = n_lemmatize(text2)
         {_.text: _.lemma for _ in n_doc2.tokens}
Out[33]: {'Китай': 'китай',
          'или': 'или',
          'Чжунго': 'чжунго',
          ',': ',',
          'как': 'как',
          'его': 'он',
          'называют': 'называть',
          'сами': 'сам',
          'китайцы': 'китаец',
          'является': 'являться',
          'одной': 'один',
          'из': 'из',
          'самых': 'самый',
          'удивительных': 'удивительный',
          'и': 'и',
          'загадочных': 'загадочный',
          'стран': 'страна',
          'мира': 'мир'}

In [34]: n_doc3 = n_lemmatize(text3)
         {_.text: _.lemma for _ in n_doc3.tokens}
Out[34]: {'Топография': 'топография',
          'Китая': 'китай',
          'очень': 'очень',
          'разнообразна': 'разнообразный',
          ',': ',',
          'на': 'на',
          'его': 'его',
          'территории': 'территория',
          'имеются': 'иметься',
          'высокие': 'высокий',
          'горы': 'гора',
          'плато': 'плато',
          'впадины': 'впадина',
          'пустыни': 'пустынь',
          'и': 'и',
          'обширные': 'обширный',
          'равнины': 'равнина'}
```

Выделение (распознавание) именованных сущностей

Spacy

```
In [35]: #Выделение  选择  #Spacy
         for ent in spacy_text3.ents:
             print(ent.text, ent.label_)

Китая LOC
впадины LOC
```

```
In [36]: from spacy import displacy
         displacy.render(spacy_text3, style='ent', jupyter=True)
```

Топография Китая LOC очень разнообразна, на его территории имеются высокие горы, плато, впадины LOC , пустыни и обширные равнины.

```
In [37]: print(spacy.explain("LOC"))

Non-GPE locations, mountain ranges, bodies of water
```

```
In [38]: print(spacy.explain("PER"))

Named person or family.
```

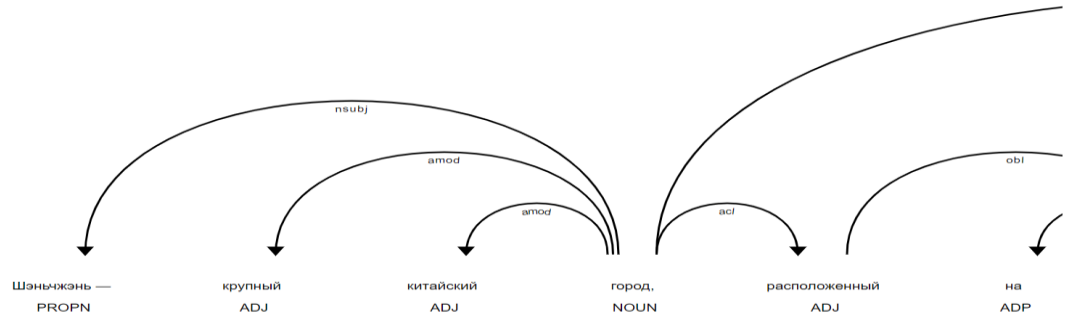
Natasha

```
In [40]: #俄语 NER、标准 PER、LOC、ORG 注释，受过新闻文章培训
         ner = NER.load('./slovnet_ner_news_v1.tar')

In [41]: ner_res = ner.navec(navec)

In [42]: markup_ner3 = ner(text3)

In [43]: markup_ner3
Out[43]: SpanMarkup(
             text='Топография Китая очень разнообразна, на его территории имеются высокие горы, плато, впадины, пустыни и обширные равнины.',
             spans=[Span(
                  start=0,
                  stop=10,
                  type='ORG'
              ),
              Span(
                  start=11,
                  stop=16,
                  type='LOC'
              )]
         )

In [44]: show_markup(markup_ner3.text, markup_ner3.spans)

Топография Китая очень разнообразна, на его территории имеются высокие
ORG———————— LOC——
горы, плато, впадины, пустыни и обширные равнины.
```

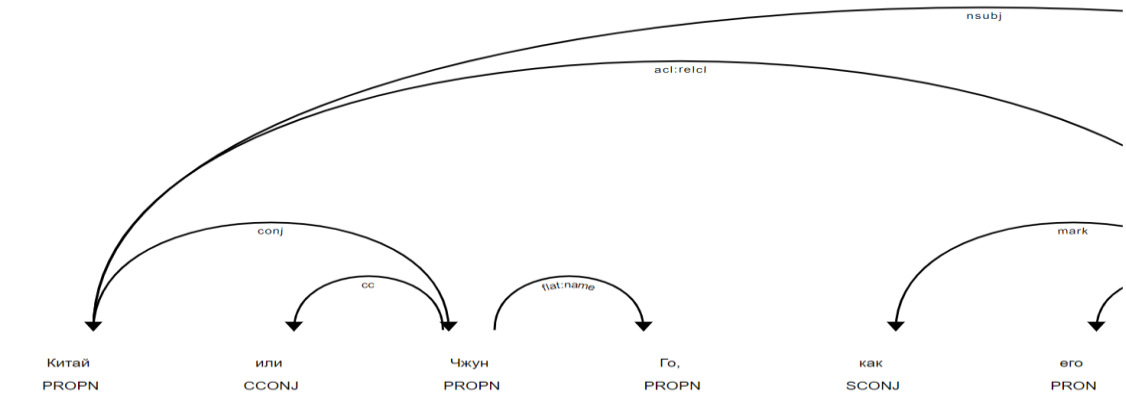
Разбор предложения

Spacy

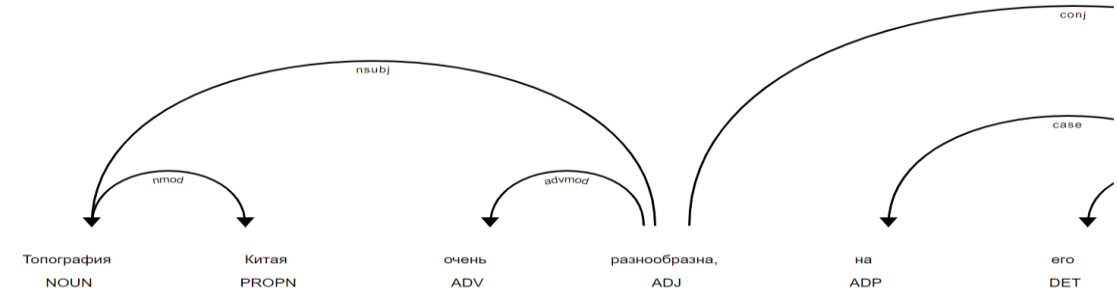
```
In [45]:  #Разбор предложения  提案分析
          from spacy import displacy
          displacy.render(spacy_text1, style='dep', jupyter=True)
```




```
In [46]:  displacy.render(spacy_text2, style='dep', jupyter=True)
```




```
In [47]:  print(spacy.explain("NOUN"))
          noun

In [48]:  print(spacy.explain("amod"))
          adjectival modifier

In [49]:  displacy.render(spacy_text3, style='dep', jupyter=True)
```



Natasha

In [50]:
```
#natasha
from natasha import NewsSyntaxParser
```

In [51]:
```
emb = NewsEmbedding()
syntax_parser = NewsSyntaxParser(emb)
```

In [52]:
```
n_doc1.parse_syntax(syntax_parser)
n_doc1.sents[0].syntax.print()
```

```
┌──────────► Шэньчжэнь      nsubj
│      —
│  ┌───────► крупный         amod
│  │  ┌────► китайский       amod
┌──┌──└──└──└─ город
│  │  │  ┌──► ,              punct
│  │  └──└──► расположенный amod
│  │     ┌──► на              case
│  └─────└──┌─ юге            obl
│  │        └► страны         nmod
│  │        ┌► ,              punct
│  │        │  ┌► возле        case
│  └────────┌──└──└─ границы    conj
│           │  ┌► с            case
│           └──└─ Гонконгом    nmod
└──────────────► .              punct
```

In [53]:
```
n_doc1.parse_syntax(syntax_parser)
n_doc1.sents[1].syntax.print()
```

```
┌──────────► Он             nsubj:pass
┌──────────┌──└─ построен
│     │  ┌► в               case
│  ┌──┌──└──└─ устье        obl
│  │  │  ┌► Жемчужной       amod
│  │  └──└─ реки            nmod
│  │  ┌──► ,               punct
│  │  │  ┌► на              case
│  └─┌──└──└─ побережье     conj
│    │  ┌► Южно-Китайского amod
│    └──└─ моря             nmod
└──────────► .              punct
```

In [54]:
```
n_doc2.parse_syntax(syntax_parser)
n_doc2.sents[0].syntax.print()
```

```
┌──────────────────► Китай          nsubj
│           ┌──► или               cc
│        ┌──└─ Чжун
│        └────► Го                 flat:name
│     ┌─────► ,                    punct
│     │  ┌──► как                  advmod
│     │  │  ┌► его                 obj
│  ┌──└──└──└─ называют            parataxis
│  │  │  │  ┌► сами                amod
│  │  │  └──└─ китайцы             nsubj
│  │  └────────► ,                 punct
└──└──────────┌──└─ является
   ┌─────────└► одной              xcomp
   │  ┌──────► из                  case
   │  │  ┌► самых                  amod
   │  │  ┌──└─ удивительных        amod
   │  │  │  ┌► и                   cc
   │  │  └──└─ загадочных          conj
   └─└──└─────┌─ стран              nmod
      └► мира                      nmod
```

In [55]:
```
n_doc3.parse_syntax(syntax_parser)
n_doc3.sents[0].syntax.print()
```

```
┌─────► Топография      nsubj
│  Китая
│  ┌► очень             advmod
┌──└──└─ разнообразна
│  ┌─────► ,                    punct
│  │  ┌──► на                   case
│  │  │  ┌► его                 det
│  │  └──└─ территории          obl
│  └─────└─ имеются
│  │  ┌► высокие               amod
┌──┌──┌──┌──└──└─ горы          nsubj
│  │  │  │  ┌► ,                punct
│  │  │  └──└─ плато            conj
│  │  │  ┌► ,                   punct
│  │  └──└─ впадины            conj
│  │  ┌► ,                      punct
│  └──└─ пустыни                conj
│  │  ┌► и                     cc
```

3.2 Для произвольного набора данных, предназначенного для классификации текстов
Анализоценки:

1.Модель word2vec:

In [2]:
```
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\92883\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]: True

In [3]:
```
# Загрузка данных
imdb_df = pd.read_csv("./sen.txt", delimiter='__label__ ', header=None, names=['value', 'text'])
imdb_df.head()
```

```
D:\AppIntas11\Anaconda\lib\site-packages\pandas\util\_decorators.py:311: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoid this warning by specifying engine='python'.
  return func(*args, **kwargs)
```

Out[3]:

	value	text
0	2	Great CD: My lovely Pat has one of the GREAT v...
1	2	One of the best game music soundtracks - for a...
2	1	Batteries died within a year ...: I bought thi...
3	2	works fine, but Maha Energy is better: Check o...
4	2	Great for the non-audiophile: Reviewed quite a...

```
corpus = []
stop_words = stopwords.words('english')
tok = WordPunctTokenizer()
for line in imdb_df['text'].values:
    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 [5]:
```
corpus[:5]
```

```
 'great',
 'voices',
 'generation',
 'listened',
 'cd',
 'years',
 'still',
 'love',
 'good',
 'mood',
 'makes',
 'feel',
 'better',
 'bad',
 'mood',
 'evaporates',
 'like',
 'sugar',
 'rain',
 'cd',
```

In [6]:
```
# количество текстов в корпусе не изменилось и соответствует целевому призн
#语料库中的文本数量没有变化，与目标特征相对应
assert imdb_df.shape[0]==len(corpus)
```

In [7]:
```
%time model_imdb = word2vec.Word2Vec(corpus, workers=4, min_count=10, window=10, sample=1e-3)
```

```
Wall time: 673 ms
```

In [8]:
```
# Проверим, что модель обучилась
print(model_imdb.wv.most_similar(positive=['great'], topn=5))
```

```
[('also', 0.9997166395187378), ('high', 0.9996911883354187), ('get', 0.9996903538703918), ('never', 0.9996834993362427), ('way', 0.9996824860572815)]
```

In [9]:
```
#基于word2vec模型解决文本情感分析问题

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 [10]:
```
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 [11]:
```
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 [12]:
```
# Обучающая и тестовая выборки  训练和测试集
boundary = 700
X_train = corpus[:boundary]
X_test = corpus[boundary:]
y_train = imdb_df.value.values[:boundary]
y_test = imdb_df.value.values[boundary:]
```

Accuary:

In [13]:
```
sentiment(EmbeddingVectorizer(model_imdb.wv), LogisticRegression(C=5.0))
```

```
Метка       Accuracy
1       0.905829596412556
2       0.2729044834307992
```

2. CountVectorizer и TfidfVectorizer:

In [16]:
```
vocabVect = CountVectorizer()
vocabVect.fit(vocab_list)
corpusVocab = vocabVect.vocabulary_
print('Количество сформированных признаков - {}'.format(len(corpusVocab)))
```

```
Количество сформированных признаков - 12348
```

In [17]:
```
for i in list(corpusVocab)[1:10]:
    print('{}={}'.format(i, corpusVocab[i]))
```

```
cd=1819
my=7178
lovely=6453
pat=7875
has=4990
one=7552
of=7503
the=10919
voices=11806
```

In [18]:
```
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, imdb_df['text'], imdb_df['value'], scoring='accuracy', cv=3).mean()
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('===========================')
```

In [20]:
```
vectorizers_list = [CountVectorizer(vocabulary = corpusVocab), TfidfVectorizer(vocabulary = corpusVocab)]
classifiers_list = [LogisticRegression(C=3.0), LinearSVC(), KNeighborsClassifier()]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '00000': 2, '02': 3, '03': 4,
                            '04': 5, '05': 6, '09': 7, '0zero0': 8, '10': 9,
                            '100': 10, '1000': 11, '10000': 12, '10000000': 13,
                            '100bucks': 14, '100ft': 15, '100x': 16, '101': 17,
                            '1020': 18, '10off': 19, '10th': 20, '11': 21,
                            '1100': 22, '112': 23, '11th': 24, '12': 25,
                            '120lbs': 26, '1215': 27, '123': 28, '124': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.7763713080168776
===========================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '00000': 2, '02': 3, '03': 4,
                            '04': 5, '05': 6, '09': 7, '0zero0': 8, '10': 9,
                            '100': 10, '1000': 11, '10000': 12, '10000000': 13,
                            '100bucks': 14, '100ft': 15, '100x': 16, '101': 17,
                            '1020': 18, '10off': 19, '10th': 20, '11': 21,
                            '1100': 22, '112': 23, '11th': 24, '12': 25,
                            '120lbs': 26, '1215': 27, '123': 28, '124': 29, ...})
Модель для классификации - LinearSVC()
Accuracy = 0.7643158529234478
===========================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '00000': 2, '02': 3, '03': 4,
                            '04': 5, '05': 6, '09': 7, '0zero0': 8, '10': 9,
                            '100': 10, '1000': 11, '10000': 12, '10000000': 13,
                            '100bucks': 14, '100ft': 15, '100x': 16, '101': 17,
                            '1020': 18, '10off': 19, '10th': 20, '11': 21,
                            '1100': 22, '112': 23, '11th': 24, '12': 25,
                            '120lbs': 26, '1215': 27, '123': 28, '124': 29, ...})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.5762507534659433
===========================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '00000': 2, '02': 3, '03': 4,
                            '04': 5, '05': 6, '09': 7, '0zero0': 8, '10': 9,
                            '100': 10, '1000': 11, '10000': 12, '10000000': 13,
                            '100bucks': 14, '100ft': 15, '100x': 16, '101': 17,
                            '1020': 18, '10off': 19, '10th': 20, '11': 21,
                            '1100': 22, '112': 23, '11th': 24, '12': 25,
                            '120lbs': 26, '1215': 27, '123': 28, '124': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.7974683544303797
```

```
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '00000': 2, '02': 3, '03': 4,
                           '04': 5, '05': 6, '09': 7, '0zero0': 8, '10': 9,
                           '100': 10, '1000': 11, '10000': 12, '10000000': 13,
                           '100bucks': 14, '100ft': 15, '100x': 16, '101': 17,
                           '1020': 18, '10off': 19, '10th': 20, '11': 21,
                           '1100': 22, '112': 23, '11th': 24, '12': 25,
                           '120lbs': 26, '1215': 27, '123': 28, '124': 29, ...})
Модель для классификации - LinearSVC()
Accuracy = 0.7920433996383364
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '00000': 2, '02': 3, '03': 4,
                           '04': 5, '05': 6, '09': 7, '0zero0': 8, '10': 9,
                           '100': 10, '1000': 11, '10000': 12, '10000000': 13,
                           '100bucks': 14, '100ft': 15, '100x': 16, '101': 17,
                           '1020': 18, '10off': 19, '10th': 20, '11': 21,
                           '1100': 22, '112': 23, '11th': 24, '12': 25,
                           '120lbs': 26, '1215': 27, '123': 28, '124': 29, ...})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.6449668474984931
============================
```

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(imdb_df['text'], imdb_df['value'], test_size=0.5, random_state=1)
```

In [22]:
```
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 [23]:
```
sentiment(TfidfVectorizer(), LogisticRegression(C=5.0))
```

```
Метка       Accuracy
1       0.8034398034398035
2       0.8250591016548463
```

In [28]:
```
sentiment(TfidfVectorizer(ngram_range=(1,3)),  LogisticRegression(C=5.0))
```

```
Метка       Accuracy
1       0.7936117936117936
2       0.8037825059101655
```

In [25]:
```
sentiment(TfidfVectorizer(ngram_range=(2,3)), LogisticRegression(C=5.0))
```

```
Метка       Accuracy
1       0.6953316953316954
2       0.8392434988179669
```

In [26]:
```
sentiment(TfidfVectorizer(ngram_range=(1,4)), LogisticRegression(C=5.0))
```

```
Метка       Accuracy
1       0.7960687960687961
2       0.8014184397163121
```

In [27]:
```
sentiment(TfidfVectorizer(ngram_range=(2,6)), LogisticRegression(C=5.0))
```

```
Метка       Accuracy
1       0.6093366093366094
2       0.8865248226950354
```

Сравнивая точность, я думаю, модель(TfidfVectorizer и LogisticRegression) лучше при анализе моих данных

4. Список литературы

[1] Amazon Reviews for Sentiment Analysis // Kaggle. — 2022. — Access mode: https://www.kaggle.com/datasets/bittlingmayer/amazonreviews/code?resource=download&select=train.ft.txt.bz2 (online; accessed:26.04.2022).
[2] url-
https://nbviewer.org/github/ugapanyuk/ml_course_2022/blob/main/common/notebooks/text/preprocess.ipynb.
[3] url-
https://nbviewer.org/github/ugapanyuk/ml_course_2022/blob/main/common/notebooks/text/bag_of_words.ipynb .
[4] url-
https://nbviewer.org/github/ugapanyuk/ml_course_2022/blob/main/common/notebooks/text/embeddings.ipynb.