Лабораторная работа №5

Ишков Денис Олегович, ИУ5-24М, 2021г.

Тема: Предобработка текста

Цель: Изучение методов предобработки текстов..

Требования к отчету: Отчет по лабораторной работе должен содержать:

- титульный лист;
- описание задания;
- текст программы;
- экранные формы с примерами выполнения программы. Задание:

Для произвольного предложения или текста решите следующие задачи:

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

Датасет ¶

Бинарная классификация текста

https://www.kaggle.com/blackmoon/russian-language-toxic-comments (https://www.kaggle.com/blackmoon/russian-language-toxic-comments)

In [1]:

```
# загрузка датасета
!pip install wldhx.yadisk-direct
!curl -L $(yadisk-direct https://disk.yandex.ru/d/wedARfrtMn-Y-Q) -o labeled.csv
```

```
Collecting wldhx.yadisk-direct
  Downloading wldhx.yadisk_direct-0.0.6-py3-none-any.whl (4.5 kB)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-p
ackages (from wldhx.yadisk-direct) (2.25.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python
3.7/site-packages (from requests->wldhx.yadisk-direct) (2020.12.5)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/lib/python
3.7/site-packages (from requests->wldhx.yadisk-direct) (4.0.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/pyt
hon3.7/site-packages (from requests->wldhx.yadisk-direct) (1.26.4)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/si
te-packages (from requests->wldhx.yadisk-direct) (2.10)
Installing collected packages: wldhx.yadisk-direct
Successfully installed wldhx.yadisk-direct-0.0.6
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
rrent
                                 Dload  Upload   Total   Spent    Left  Sp
eed
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--
0
100 4560k  100 4560k    0     0  1161k      0  0:00:03  0:00:03 --:--:-- 1
720k
```

Импорт нужных библиотек

In [3]:

```
import nltk
import spacy
import numpy as np
nltk.download('punkt')
from nltk import tokenize
import re
import pandas as pd
from sklearn.model_selection import train_test_split
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import ComplementNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import precision_score, recall_score, precision_recall_curve, clas
sification_report
from matplotlib import pyplot as plt
from sklearn.metrics import plot_precision_recall_curve
import numpy as np
from sklearn.model_selection import GridSearchCV
```

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

Анализ и обработка выбросов в данных

In [4]:

```
df = pd.read_csv("labeled.csv", sep=",")

df.describe()
```

Out[4]:

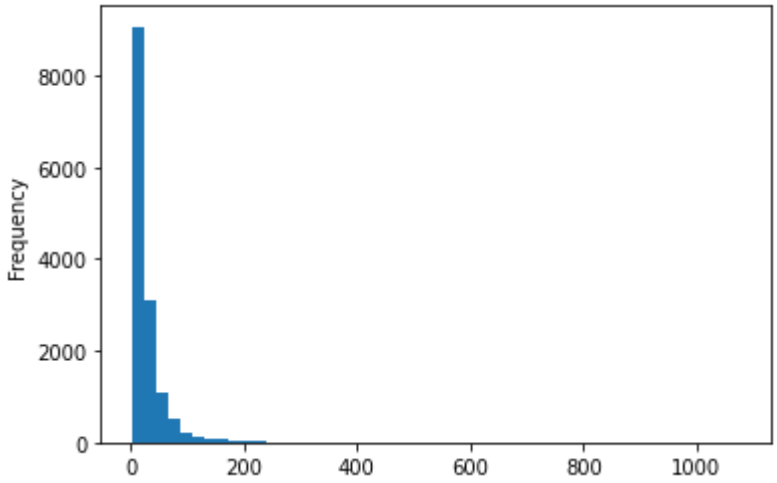
	toxic
count	14412.000000
mean	0.334860
std	0.471958
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

In [5]:

```
df.comment.str.split(' ').apply(len).plot(kind='hist', bins=50)
```

Out[5]:

```
<AxesSubplot:ylabel='Frequency'>
```



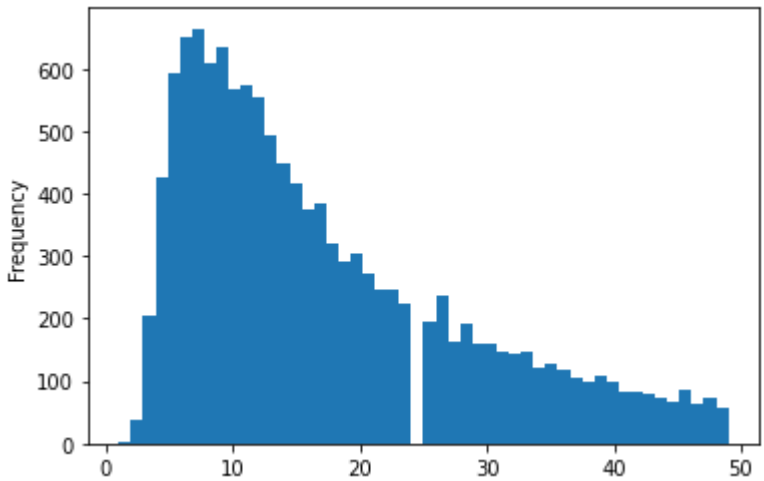
Как видно из гистограммы, количество слов сообщений в данных распределено по экспоненциальному закону. Уберём "хвост"

In [6]:

```
df = df[df.comment.str.split(' ').apply(len) < 50].copy()
df.comment.str.split(' ').apply(len).plot(kind='hist', bins=50)
```

Out[6]:

```
<AxesSubplot:ylabel='Frequency'>
```



In [7]:

```
df["toxic"] = df["toxic"].apply(int)
df["toxic"].value_counts()
```

Out[7]:

```
0    8102
1    4419
Name: toxic, dtype: int64
```

1. Токенизация

In [9]:

```
print('Токенизаторы NLTK')
for i in dir(tokenize)[:16]:
    print(i)
```

```
Токенизаторы NLTK
BlanklineTokenizer
LineTokenizer
MWETokenizer
PunktSentenceTokenizer
RegexpTokenizer
ReppTokenizer
SExprTokenizer
SpaceTokenizer
StanfordSegmenter
StanfordTokenizer
TabTokenizer
TextTilingTokenizer
ToktokTokenizer
TreebankWordTokenizer
TweetTokenizer
WhitespaceTokenizer
```

In [67]:

```
comment = df.comment.values[222]
print(comment)
cleaned_comment = re.sub('[^а-яА-Яa-zA-Z0-9 \n\.]', '', comment)
print(cleaned_comment)
```

```
Бросила. Во время каждого занятия получала небольшие травмы, в последний р
аз при выполнении Сурьи Намаскар растянула связки на руке, боль была така
я, что в первые секунды на перелом подумала. Теперь дома занимаюсь без нап
ряга со стороны тренера.

Бросила. Во время каждого занятия получала небольшие травмы в последний ра
з при выполнении Сурьи Намаскар растянула связки на руке боль была такая ч
то в первые секунды на перелом подумала. Теперь дома занимаюсь без напряга
со стороны тренера.
```

In [22]:

```
tokenizer_wp = nltk.WhitespaceTokenizer()
tokens = tokenizer_wp.tokenize(cleaned_comment)
tokens
```

Out[22]:

```
['Бросила.',
 'Во',
 'время',
 'каждого',
 'занятия',
 'получала',
 'небольшие',
 'травмы',
 'в',
 'последний',
 'раз',
 'при',
 'выполнении',
 'Сурьи',
 'Намаскар',
 'растянула',
 'связки',
 'на',
 'руке',
 'боль',
 'была',
 'такая',
 'что',
 'в',
 'первые',
 'секунды',
 'на',
 'перелом',
 'подумала.',
 'Теперь',
 'дома',
 'занимаюсь',
 'без',
 'напряга',
 'со',
 'стороны',
 'тренера.']
```

In [23]:

```
tokenizer_wp = nltk.WordPunctTokenizer()
tokens = tokenizer_wp.tokenize(cleaned_comment)
tokens
```

Out[23]:

```
['Бросила',
 '.',
 'Во',
 'время',
 'каждого',
 'занятия',
 'получала',
 'небольшие',
 'травмы',
 'в',
 'последний',
 'раз',
 'при',
 'выполнении',
 'Сурьи',
 'Намаскар',
 'растянула',
 'связки',
 'на',
 'руке',
 'боль',
 'была',
 'такая',
 'что',
 'в',
 'первые',
 'секунды',
 'на',
 'перелом',
 'подумала',
 '.',
 'Теперь',
 'дома',
 'занимаюсь',
 'без',
 'напряга',
 'со',
 'стороны',
 'тренера',
 '.']
```

Делаем вывод, что WordPunctTokenizer подходит лучше

2. Частеречная разметка (POStagging)

In [27]:

```
!pip install natasha
```

```
Collecting natasha
  Downloading natasha-1.4.0-py3-none-any.whl (34.4 MB)
     |████████████████████████████████| 34.4 MB 25.3 MB/s eta 0:00:01
Requirement already satisfied: ipymarkup>=0.8.0 in /opt/conda/lib/python3.
7/site-packages (from natasha) (0.9.0)
Collecting slovnet>=0.3.0
  Downloading slovnet-0.5.0-py3-none-any.whl (49 kB)
     |████████████████████████████████| 49 kB 3.5 MB/s  eta 0:00:01
Collecting pymorphy2
  Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
     |████████████████████████████████| 55 kB 2.1 MB/s  eta 0:00:01
Collecting razdel>=0.5.0
  Downloading razdel-0.5.0-py3-none-any.whl (21 kB)
Collecting navec>=0.9.0
  Downloading navec-0.10.0-py3-none-any.whl (23 kB)
Collecting yargy>=0.14.0
  Downloading yargy-0.15.0-py3-none-any.whl (41 kB)
     |████████████████████████████████| 41 kB 72 kB/s s eta 0:00:01
Requirement already satisfied: intervaltree>=3 in /opt/conda/lib/python3.
7/site-packages (from ipymarkup>=0.8.0->natasha) (3.1.0)
Requirement already satisfied: sortedcontainers<3.0,>=2.0 in /opt/conda/li
b/python3.7/site-packages (from intervaltree>=3->ipymarkup>=0.8.0->natash
a) (2.3.0)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-pack
ages (from navec>=0.9.0->natasha) (1.19.5)
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 42.0 MB/s eta 0:00:01
Collecting dawg-python>=0.7.1
  Downloading DAWG_Python-0.7.2-py2.py3-none-any.whl (11 kB)
Collecting docopt>=0.6
  Downloading docopt-0.6.2.tar.gz (25 kB)
Building wheels for collected packages: docopt
  Building wheel for docopt (setup.py) ... done
  Created wheel for docopt: filename=docopt-0.6.2-py2.py3-none-any.whl siz
e=13705 sha256=d85aee54dead03a93b816c9de6743098922292053c9314a9f03cc10fa46
08bae
  Stored in directory: /root/.cache/pip/wheels/72/b0/3f/1d95f96ff986c7dfff
e46ce2be4062f38ebd04b506c77c81b9
Successfully built docopt
Installing collected packages: pymorphy2-dicts-ru, docopt, dawg-python, ra
zdel, pymorphy2, navec, yargy, slovnet, natasha
Successfully installed dawg-python-0.7.2 docopt-0.6.2 natasha-1.4.0 navec-
0.10.0 pymorphy2-0.9.1 pymorphy2-dicts-ru-2.4.417127.4579844 razdel-0.5.0
slovnet-0.5.0 yargy-0.15.0
```

In [33]:

```
from natasha import (
    Segmenter,
    MorphVocab,

    NewsEmbedding,
    NewsMorphTagger,
    NewsSyntaxParser,
    NewsNERTagger,

    PER,
    NamesExtractor,

    Doc
)
segmenter = Segmenter()
emb = NewsEmbedding()
morph_tagger = NewsMorphTagger(emb)
syntax_parser = NewsSyntaxParser(emb)

doc = Doc(cleaned_text)
doc.segment(segmenter)
print(doc.tokens)
doc.tag_morph(morph_tagger)
doc.sents[1].morph.print()
```

```
[DocToken(stop=7, text='Бросила'), DocToken(start=7, stop=8, text='.'), Do
cToken(start=9, stop=11, text='Во'), DocToken(start=12, stop=17, text='вре
мя'), DocToken(start=18, stop=25, text='каждого'), DocToken(start=26, stop
=33, text='занятия'), DocToken(start=34, stop=42, text='получала'), DocTok
en(start=43, stop=52, text='небольшие'), DocToken(start=53, stop=59, text
='травмы'), DocToken(start=60, stop=61, text='в'), DocToken(start=62, stop
=71, text='последний'), DocToken(start=72, stop=75, text='раз'), DocToken
(start=76, stop=79, text='при'), DocToken(start=80, stop=90, text='выполне
нии'), DocToken(start=91, stop=96, text='Сурьи'), DocToken(start=97, stop=
105, text='Намаскар'), DocToken(start=106, stop=115, text='растянула'), Do
cToken(start=116, stop=122, text='связки'), DocToken(start=123, stop=125,
text='на'), DocToken(start=126, stop=130, text='руке'), DocToken(start=13
1, stop=135, text='боль'), DocToken(start=136, stop=140, text='была'), Doc
Token(start=141, stop=146, text='такая'), DocToken(start=147, stop=150, te
xt='что'), DocToken(start=151, stop=152, text='в'), DocToken(start=153, st
op=159, text='первые'), DocToken(start=160, stop=167, text='секунды'), Doc
Token(start=168, stop=170, text='на'), DocToken(start=171, stop=178, text
='перелом'), DocToken(start=179, stop=187, text='подумала'), DocToken(star
t=187, stop=188, text='.'), DocToken(start=189, stop=195, text='Теперь'),
DocToken(start=196, stop=200, text='дома'), DocToken(start=201, stop=210,
text='занимаюсь'), DocToken(start=211, stop=214, text='без'), DocToken(sta
rt=215, stop=222, text='напряга'), DocToken(start=223, stop=225, text='с
о'), DocToken(start=226, stop=233, text='стороны'), DocToken(start=234, st
op=241, text='тренера'), DocToken(start=241, stop=242, text='.')]
                  Во ADP
              время NOUN|Animacy=Inan|Case=Acc|Gender=Neut|Number=Sing
             каждого DET|Case=Gen|Gender=Neut|Number=Sing
             занятия NOUN|Animacy=Inan|Case=Gen|Gender=Neut|Number=Sing
            получала VERB|Aspect=Imp|Gender=Fem|Mood=Ind|Number=Sing|Tense
=Past|VerbForm=Fin|Voice=Act
           небольшие ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Number=Plur
              травмы NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Plur
                   в ADP
           последний ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Gender=Masc|Numb
er=Sing
                 раз NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing
                 при ADP
          выполнении NOUN|Animacy=Inan|Case=Loc|Gender=Neut|Number=Sing
               Сурьи PROPN|Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing
            Намаскар PROPN|Animacy=Anim|Case=Gen|Gender=Masc|Number=Sing
            растянула ADJ|Case=Gen|Degree=Pos|Gender=Fem|Number=Sing
              связки NOUN|Animacy=Inan|Case=Gen|Gender=Fem|Number=Sing
                  на ADP
               руке NOUN|Animacy=Inan|Case=Loc|Gender=Fem|Number=Sing
               боль NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Sing
               была AUX|Aspect=Imp|Gender=Fem|Mood=Ind|Number=Sing|Tense=
Past|VerbForm=Fin|Voice=Act
              такая DET|Case=Nom|Gender=Fem|Number=Sing
                что SCONJ
                  в ADP
            первые ADJ|Animacy=Inan|Case=Acc|Degree=Pos|Number=Plur
           секунды NOUN|Animacy=Inan|Case=Acc|Gender=Fem|Number=Plur
                  на ADP
            перелом NOUN|Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing
           подумала VERB|Aspect=Perf|Gender=Fem|Mood=Ind|Number=Sing|Tens
e=Past|VerbForm=Fin|Voice=Act
                  . PUNCT
```

3. Лемматизация (Lemmatization)

In [35]:

```
morph_vocab = MorphVocab()
for token in doc.tokens:
    token.lemmatize(morph_vocab)
{_.text: _.lemma for _ in doc.tokens}
```

Out[35]:

```
{'Бросила': 'бросить',
 '.': '.',
 'Во': 'в',
 'время': 'время',
 'каждого': 'каждый',
 'занятия': 'занятие',
 'получала': 'получать',
 'небольшие': 'небольшой',
 'травмы': 'травма',
 'в': 'в',
 'последний': 'последний',
 'раз': 'раз',
 'при': 'при',
 'выполнении': 'выполнение',
 'Сурьи': 'сурья',
 'Намаскар': 'намаскар',
 'растянула': 'растянуть',
 'связки': 'связка',
 'на': 'на',
 'руке': 'рука',
 'боль': 'боль',
 'была': 'быть',
 'такая': 'такой',
 'что': 'что',
 'первые': 'первый',
 'секунды': 'секунда',
 'перелом': 'перелом',
 'подумала': 'подумать',
 'Теперь': 'теперь',
 'дома': 'дома',
 'занимаюсь': 'заниматься',
 'без': 'без',
 'напряга': 'напряга',
 'со': 'с',
 'стороны': 'сторона',
 'тренера': 'тренер'}
```

Выделение (распознавание) именованных сущностей (NER)

In [36]:

```
ner_tagger = NewsNERTagger(emb)
doc.tag_ner(ner_tagger)
doc.tag_ner(ner_tagger)
doc.ner.print()
```

```
Бросила. Во время каждого занятия получала небольшие травмы в
последний раз при выполнении Сурьи Намаскар растянула связки на руке
                              PER——————————
боль была такая что в первые секунды на перелом подумала. Теперь дома
занимаюсь без напряга со стороны тренера.
```

In [74]:

```
text = '''Курская битва (5 июля — 23 августа 1943 года; также известна как Битва на Кур
ской дуге) — совокупность стратегических оборонительной (5—23 июля) и наступательных (1
2 июля — 23 августа) операций Красной армии в Великой Отечественной войне с целью сорва
ть крупное наступление сил вермахта и разгромить его стратегическую группировку[9]. По
 своим масштабам, задействованным силам и средствам, напряжённости, результатам и военн
о-политическим последствиям является одним из ключевых сражений Второй мировой войны и
 Великой Отечественной войны. В историографии считается самым крупным (величайшим) танк
овым сражением в истории[10][11][12][13][14][Прим. 1]. В нём участвовали около 2 млн че
ловек, 6 тысяч танков, 4 тысячи самолётов; сражение проложило «путь к великим советским
наступательным действиям 1944-45 годов».
Сражение является важнейшей частью стратегического плана летне-осенней кампании 1943 го
да, согласно советской и российской историографии, включает в себя: Курскую стратегичес
кую оборонительную операцию (5—23 июля), Орловскую (12 июля — 18 августа) и Белгородско
-Харьковскую (3—23 августа) стратегические наступательные операции. Битва продолжалась
 50 дней. Немецкая сторона наступательную часть сражения называла операция «Цитадель».
В результате наступления по плану «Кутузов» потерпела поражение орловская группировка н
емецких войск, а занимаемый ею орловский стратегический плацдарм был ликвидирован. По и
тогам операции «Румянцев» потерпела поражение белгородско-харьковская группировка немце
в, и соответствующий плацдарм также был ликвидирован[15]. Коренной перелом в ходе Велик
ой Отечественной войны, начатый под Сталинградом, был завершён в Курской битве и сражен
ии за Днепр, а в последовавшей Тегеранской конференции по инициативе Ф. Рузвельта уже о
бсуждался составленный им лично «2 месяца тому назад план расчленения Германии на пять
 государств».
После завершения битвы стратегическая инициатива окончательно перешла на сторону Красно
й армии, которая продолжала освобождать страну от немецких захватчиков и до окончания в
ойны проводила в основном наступательные операции. Вермахт в ходе отступления с террито
рии СССР проводил тактику «выжженной земли»[16].
23 августа — день разгрома советскими войсками немецких войск в Курской битве — являетс
я одним из дней воинской славы России. Белгород, Курск и Орёл стали первыми городами Ро
ссии, которым присвоено почётное звание «Город воинской славы»[17].'''
doc2 = Doc(text)
doc2.segment(segmenter)
doc2.tag_morph(morph_tagger)
#doc2.sents[1].morph.print()
for token in doc.tokens:
    token.lemmatize(morph_vocab)
doc2.tag_ner(ner_tagger)
doc2.ner.print()
```

Курская битва (5 июля — 23 августа 1943 года; также известна как Битва на Курской дуге) — совокупность стратегических оборонительной (5—23 июля) и наступательных (12 июля — 23 августа) операций Красной армии в
ORG——————
Великой Отечественной войне с целью сорвать крупное наступление сил вермахта и разгромить его стратегическую группировку[9]. По своим масштабам, задействованным силам и средствам, напряжённости, результатам и военно-политическим последствиям является одним из ключевых сражений Второй мировой войны и Великой Отечественной войны. В историографии считается самым крупным (величайшим) танковым сражением в истории[10][11][12][13][14][Прим. 1]. В нём участвовали около 2 млн человек, 6 тысяч танков, 4 тысячи самолётов; сражение проложило «путь к великим советским наступательным действиям 1944-45 годов».

Сражение является важнейшей частью стратегического плана летне-осенней кампании 1943 года, согласно советской и российской историографии, включает в себя: Курскую стратегическую оборонительную операцию (5—23 июля), Орловскую (12 июля — 18 августа) и Белгородско-Харьковскую
LOC———— LOC————————————
(3—23 августа) стратегические наступательные операции. Битва продолжалась 50 дней. Немецкая сторона наступательную часть сражения называла операция «Цитадель».

В результате наступления по плану «Кутузов» потерпела поражение
ORG———
орловская группировка немецких войск, а занимаемый ею орловский стратегический плацдарм был ликвидирован. По итогам операции «Румянцев» потерпела поражение белгородско-харьковская группировка немцев, и соответствующий плацдарм также был ликвидирован[15].

Коренной перелом в ходе Великой Отечественной войны, начатый под Сталинградом, был завершён в Курской битве и сражении за Днепр, а в
LOC———————— LOC——
последовавшей Тегеранской конференции по инициативе Ф. Рузвельта уже
PER——————
обсуждался составленный им лично «2 месяца тому назад план расчленения Германии на пять государств».
LOC————

После завершения битвы стратегическая инициатива окончательно перешла на сторону Красной армии, которая продолжала освобождать страну от
ORG——————
немецких захватчиков и до окончания войны проводила в основном наступательные операции. Вермахт в ходе отступления с территории СССР
LOC—
проводил тактику «выжженной земли»[16].

23 августа — день разгрома советскими войсками немецких войск в Курской битве — является одним из дней воинской славы России.
LOC——

Белгород, Курск и Орёл стали первыми городами России, которым
LOC———— LOC—— LOC— LOC——
присвоено почётное звание «Город воинской славы»[17].

In [84]:

```
names_extractor = NamesExtractor(morph_vocab)
for span in doc2.spans:
    span.normalize(morph_vocab)
{_.text: _.normal for _ in doc2.spans if _.text != _.normal}
```

Out[84]:

```
{'Красной армии': 'Красная армия',
 'Орловскую': 'Орловская',
 'Сталинградом': 'Сталинград',
 'Ф. Рузвельта': 'Ф. Рузвельт',
 'Германии': 'Германия',
 'России': 'Россия',
 'Орёл': 'Орел'}
```

5. Разбор предложения

In [86]:

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

```
Во          case
время
каждого     det
занятия     obl
получала
небольшие   amod
травмы      obl
в           case
последний   amod
раз         obl
при         case
выполнении  obl
Сурьи       nmod
Намаскар    flat:name
растянула   obl
связки      nsubj
на          case
руке        nmod
боль        nsubj
была        conj
такая       det
что         obj
в           case
первые      amod
секунды     obl
на          case
перелом     obl
подумала    conj
.           punct
```

Разделим данные на обучающую и тестовую выборки

In [ ]:

```
train_df, test_df = train_test_split(df, test_size=500, random_state=69, stratify=df.to
xic)
```

In [ ]:

```
test_df["toxic"].value_counts(), train_df["toxic"].value_counts()
```

Предобработка текста

In [ ]:

```
sentence_example = df.iloc[-1]["comment"]
tokens = word_tokenize(sentence_example, language="russian")
tokens_without_punctuation = [i for i in tokens if i not in string.punctuation]
russian_stop_words = stopwords.words("russian")
tokens_without_stop_words_and_punctuation = [i for i in tokens_without_punctuation if i
not in russian_stop_words]
snowball = SnowballStemmer(language="russian")
stemmed_tokens = [snowball.stem(i) for i in tokens_without_stop_words_and_punctuation]
```

In [ ]:

```
print(f"Исходный текст: {sentence_example}")
print("-----------------")
print(f"Токены: {tokens}")
print("-----------------")
print(f"Токены без пунктуации: {tokens_without_punctuation}")
print("-----------------")
print(f"Токены без пунктуации и стоп слов: {tokens_without_stop_words_and_punctuation}"
)
print("-----------------")
print(f"Токены после стемминга: {stemmed_tokens}")
print("-----------------")
```

In [ ]:

```
snowball = SnowballStemmer(language="russian")
russian_stop_words = stopwords.words("russian")

def tokenize_sentence(sentence: str, remove_stop_words: bool = True):
    tokens = word_tokenize(sentence, language="russian")
    tokens = [i for i in tokens if i not in string.punctuation]
    if remove_stop_words:
        tokens = [i for i in tokens if i not in russian_stop_words]
    tokens = [snowball.stem(i) for i in tokens]
    return tokens

tokenize_sentence(sentence_example)
```

Классификатор 1: Complement Naive Bayes

CountVectorizer

In [ ]:

```
grid_pipeline = Pipeline([
    ("vectorizer", CountVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model",
     GridSearchCV(
        ComplementNB(alpha=1.0, norm=False),
        param_grid={'alpha': [0.35+5e-3*i for i in range(20)]+\
                             [0.5+5e-3*i for i in range(60)]+\
                             [0.5, 0.75, 1., 10., 100., 1e3, 1e4, 1e5],
                    'norm': [True, False]},
        cv=5,
        verbose=0,
        scoring='roc_auc',
        )
    )
])

grid_pipeline.fit(train_df["comment"], train_df["toxic"])
print(grid_pipeline['model'].best_params_)
model_pipeline = Pipeline([
    ("vectorizer", CountVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model", ComplementNB(**grid_pipeline['model'].best_params_),)
]
)

model_pipeline.fit(train_df["comment"], train_df["toxic"])
prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"],
                                                              probas_pred=model_pipelin
e.predict_proba(test_df["comment"])[:, 1])
plot_precision_recall_curve(estimator=model_pipeline, X=test_df["comment"], y=test_df[
"toxic"])
print(classification_report(y_true=test_df["toxic"],
                            y_pred=model_pipeline.predict(test_df["comment"]),
                            digits=4))
```

TfidfVectorizer

In [ ]:

```
grid_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model",
     GridSearchCV(
        ComplementNB(alpha=1.0, norm=False),
        param_grid={'alpha': [0.3+5e-3*i for i in range(60)]+\
                             [0.5, 0.75, 1., 10., 100., 1e3, 1e4, 1e5],
                    'norm': [True, False]},
        cv=5,
        verbose=0,
        scoring='roc_auc',
        )
    )
])

grid_pipeline.fit(train_df["comment"], train_df["toxic"])
print(grid_pipeline['model'].best_params_)
model_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model", ComplementNB(**grid_pipeline['model'].best_params_),)
]
)

model_pipeline.fit(train_df["comment"], train_df["toxic"])
prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"],
                                                              probas_pred=model_pipelin
e.predict_proba(test_df["comment"])[:, 1])
plot_precision_recall_curve(estimator=model_pipeline, X=test_df["comment"], y=test_df[
"toxic"])
print(classification_report(y_true=test_df["toxic"],
                            y_pred=model_pipeline.predict(test_df["comment"]),
                            digits=4))
```

Классификатор 2: KNearestNeighbors

CountVectorizer

In [ ]:

```
grid_pipeline = Pipeline([
    ("vectorizer", CountVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model",
     GridSearchCV(
        KNeighborsClassifier(),
        param_grid={'n_neighbors': [i for i in range(48, 96, 2)],
                    'weights': ['uniform', 'distance'],
                    'metric': ['euclidean', 'cosine',]},
        cv=5,
        verbose=1,
        scoring='roc_auc', #'f1'
        )
    )
])

grid_pipeline.fit(train_df["comment"], train_df["toxic"])
print(grid_pipeline['model'].best_params_)
model_pipeline = Pipeline([
    ("vectorizer", CountVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model", KNeighborsClassifier(**grid_pipeline['model'].best_params_),)
]
)

model_pipeline.fit(train_df["comment"], train_df["toxic"])
prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"],
                                                              probas_pred=model_pipelin
e.predict_proba(test_df["comment"])[:, 1])
plot_precision_recall_curve(estimator=model_pipeline, X=test_df["comment"], y=test_df[
"toxic"])
print(classification_report(y_true=test_df["toxic"],
                            y_pred=model_pipeline.predict(test_df["comment"]),
                            digits=4))
```

Какие параметры лучшие?

In [ ]:

```
cols = ['param_metric', 'param_n_neighbors', 'param_weights', 'mean_test_score']
pd.DataFrame(grid_pipeline['model'].cv_results_).sort_values(by='rank_test_score').loc
[:, cols].head(10)
```

TFidfVectorizer

In [ ]:

```
grid_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model",
     GridSearchCV(
        KNeighborsClassifier(),
        param_grid={'n_neighbors': [i for i in range(31, 64, 2)]+[44, 46],
                    'weights': ['uniform', 'distance'],
                    'metric': ['euclidean', 'cosine',]},
        cv=5,
        verbose=1,
        scoring='roc_auc', #'f1'
        )
    )
])

grid_pipeline.fit(train_df["comment"], train_df["toxic"])
print(grid_pipeline['model'].best_params_)
model_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model", KNeighborsClassifier(**grid_pipeline['model'].best_params_),)
]
)

model_pipeline.fit(train_df["comment"], train_df["toxic"])
prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"],
                                                              probas_pred=model_pipelin
e.predict_proba(test_df["comment"])[:, 1])
plot_precision_recall_curve(estimator=model_pipeline, X=test_df["comment"], y=test_df[
"toxic"])
print(classification_report(y_true=test_df["toxic"],
                            y_pred=model_pipeline.predict(test_df["comment"]),
                            digits=4))
```

Какие параметры лучшие?

In [ ]:

```
cols = ['param_metric', 'param_n_neighbors', 'param_weights', 'mean_test_score']
pd.DataFrame(grid_pipeline['model'].cv_results_).sort_values(by='rank_test_score').loc
[:, cols].head(10)
```

Выводы

	ComplementNB	KNN Classifier
CountVectorizer	0.8701	0.7326
TfIdfVectorizer	0.8703	0.7848

Лучше всего по f1-мере показала себя связка TfIdfVectorizer + ComplementNB. Скорее всего, качество выше соседей, потому что модель специально заточена под несбалансированные выборки. В таком случае нужно было заранее выравнивать по классам данные для обучения соседей, чего не требовалось по заданию РК.