

Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
**"Национальный исследовательский университет
Высшая школа экономики"**
Департамент прикладной математики, бакалавр

НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА
НА ТЕМУ:

**Методы анализа русскоязычных текстов для извлечения
ключевых слов**

Выполнили:

*Колодин Матвей Алексеевич
Батарева Софья Евгеньевна
Тупицкий Даниил Сергеевич*

Академический руководитель:

Сластников Сергей Александрович

Москва 2022

Содержание

I Извлечение ключевых слов (автор: Матвей Колодин) 4

1 Введение	4
1.1 Что такое NLP?	4
1.2 Актуальность направления	4
1.3 Проблемы NLP	4
1.4 Выводы	5
2 Реализация	6
2.1 Реализация алгоритмов NLP	6
2.2 Предобработка данных (Data Preprocessing)	6
2.2.1 Очистка данных	6
2.2.2 Перевод символов в нижний регистр	7
2.2.3 Токенизация	8
2.2.4 Лемматизация	8
2.2.5 Удаление стоп-слов	9
2.2.6 N-граммы	9
2.3 Векторизация текстовых данных	10
2.3.1 «Мешок слов» (Bag of words (BOW))	11
2.3.2 Обратная частота документа (TF-IDF)	13
2.4 Постановка задачи	15
2.5 Решение задачи	15
2.6 Применение результатов извлечения ключевых слов	17
3 Заключение	18
3.1 Анализ полученных данных	18
3.2 Извлечение ключевых слов	18
3.3 Извлечение самых популярных слов (ROF_MPW)	20
3.4 Приложение	20

II Выявление превалирующих качественных характеристик ключевых слов (автор: Батарева Софья) 21

4 Выявление превалирующих качественных характеристик ключевых слов	21
4.1 Введение	21
4.2 Надо установить библиотеку 'Rymorphy2' с помощью pip:	22
4.3 Затем импортируем в программу модули библиотек:	22
4.4 Подключаем к программе два текстовых файла:	23
4.5 Токенизируем текст:	23

4.6	Теперь надо лемматизировать получившиеся слова (привести их к начальной форме):	24
4.7	Токенизируем и лемматизируем текст из файла:	25
4.8	Создаем словарь начальных форм слов:	25
4.9	Переносим полученный словарь в файл ‘Synonym.txt’	26
4.10	Результат работы программы:	27
4.10.1	Тест 1:	27
4.10.2	Тест 2:	28
4.11	Текст программы:	29
4.12	Итог:	30

III Извлечение именованных сущностей (NER) (автор: Тупицкий Даниил) 31

5	NER	31
5.1	Что такое NER	31
5.2	Задача	31
5.3	Актуальность	31
5.4	Сложности решения	32
5.5	Метрика	33
6	Инструменты для решения NER для русского языка	33
6.1	Natasha	33
6.2	Stanza	34
6.3	Polyglot	35
6.4	PullEnti	36
6.5	Вывод	40

Список используемой литературы 41

Часть I

Извлечение ключевых слов (автор: Матвей Колодин)

1 Введение

1.1 Что такое NLP?

Обработка естественного языка (NLP) — это область лингвистики, информатики и искусственного интеллекта, связанная с взаимодействием между компьютерами и человеческим языком, в частности с тем, как программировать компьютеры для обработки и анализа больших объемов данных естественного языка.

В data science обработка естественного языка является важным компонентом, по той причине, что спектр ее применения в различных отраслях / секторах в настоящее время крайне широкий — например, когда мы делаем запрос в голосовом помощнике Google или Siri, когда пользуемся переводчиком или поисковой системой или занимаемся копирайтингом, а также иными видами обработки текста.

NLP - это подход, который позволяет машинам интерпретировать и, в какой-то степени, понимать смысл естественного языка.

1.2 Актуальность направления

В настоящее время социальные сети — это большой источник данных для естественных языков, будь то любые обзоры с любых онлайн-сайтов (Amazon, Google и т.д.) Или просто сообщения из Twitter, Facebook, Instagram, VK или электронной почты.

Вариантами использования NLP можно назвать задачи классификации, обобщения текста, реализация интерактивных голосовых ответов (IVR), языковой перевод, чат-боты. Задачи могут отличаться в каждом секторе, но NLP — это та основа, которая необходима для каждой из задач, связанной с работой с языками.

1.3 Проблемы NLP

Естественные языки — это, прежде всего, свободная форма текста, что означает, что они крайне не структурированы по своей природе. А отсюда следуют многие проблемы при обработке компьютером данных.

Естественный язык изначально не отличается однозначностью толкования. Порой, даже для человека довольно трудно понять собеседника. В качестве примера возьмем следующую выдержку В. Кличко:

“А сегодня, в завтрашний день, не все могут смотреть. Вернее, смотреть могут не только лишь все, не каждый может это делать.”

Или, например, уже более бытовую фразу “Да нет”. Русскоговорящий человек с легкостью мог бы понять, что данная фраза - отказ, но для машины это не столь очевидно. Выразительные средства, использование синонимов, антонимов и т.д. для уточнения или усиления смысла еще более усложняют автоматический анализ текста.

1.4 Выводы

При работе с естественными языками возникают различные трудности, такие как - неоднозначность толкования слов, неструктурированность предложений, наличие различных выразительных средств. Исходя из наличия трудностей, возникает необходимость их решить.

Очистка и подготовка данных, перед извлечением ключевых слов, является важным начальным шагом для процесса NLP при разработке любой модели, как для качества выполнения алгоритма, так и для его производительности.

В моем исследовании будут рассмотрены основные шаги при работе с NLP, а также варианты применения данного направления в различных задачах.

Также важно отметить, что сам проект будет реализован на языке программирования **Python 3.9** при использовании ряда специализированных библиотек.

2 Реализация

2.1 Реализация алгоритмов NLP

Перед непосредственным решением задачи, нужно научиться базовым методам в NLP.

Прежде всего, я хочу привести список тех библиотек и функций, что мне понадобились при реализации проекта. К части библиотек я также прикладываю ссылку с их подробным описанием.

```
import re
import pandas as pd
import os
from pymorphy2 import MorphAnalyzer
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
from nltk.tokenize import sent_tokenize
from nltk.probability import FreqDist
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from nltk.corpus import stopwords
# nltk.download('punkt')
# nltk.download('stopwords')
# nltk.download('wordnet')
# nltk.download('omw-1.4')
# os.chdir("NLP")
```

Рис. 1: Список используемых библиотек и функций.

2.2 Предобработка данных (Data Preprocessing)

Перед непосредственной работой с текстом, как мы уже поняли, данные необходимо предобработать – то есть очистить их и подготовить его к построению модели.

В моей реализации за это будет отвечать функция *data_preprocessing*, которая на вход принимает текст и некоторое число k (оно будет отвечать за разбиение текста на n -grams. Далее я подробнее расскажу про это понятие).

В данной функции реализована задача предобработки текста. Давайте рассмотрим ближе, как она работает, и разберем ее реализацию.

2.2.1 Очистка данных

Начнем с очистки данных. Нам необходимо исключить из данных “символьный мусор” вроде смайликов, скобочек, кавычек, черточек и тому подобных символов.

```
def data_preprocessing(data, k):

    # Очищаем данные.
    data = re.sub('[^A-Za-zА-яа-я0-9]+', ' ', data)

    # Переводим слова в нижний регистр.
    data = data.lower()

    # Токенизируем корпус.
    data = word_tokenize(data, language='russian')

    # Производим лемматизацию.
    data = [morph.normal_forms(word)[0] for word in data]

    # Удаляем стоп-слова для случая с юниграммами. Иначе оставляем.
    if k == 1:
        data = [word for word in data if word not in stop_words]

    # Работаем с n-граммами.
    if k > 1:
        data = list(ngrams(data, k))

    # Возвращаем результат пользователю.
    return data
```

Рис. 2: Реализация алгоритма предобработки данных.

Для решения данной задачи я буду пользоваться библиотекой **Re**. Сама библиотека посвящена обработке регулярных выражений. Для более глубокого изучения могу порекомендовать данную статью.

```
data = "✱ До/ Форума\ ✱осталось. <72> дня!"
print("Исходный текст - ", data)
data = re.sub('[^A-Za-zА-яа-я0-9]+', ' ', data)
print("Результат обработки - ", data)
```

Исходный текст - ✱ До/ Форума\ ✱осталось. <72> дня!
Результат обработки - До Форума осталось 72 дня

Рис. 3: Предобработка текста. Исключение лишних символов.

2.2.2 Перевод символов в нижний регистр

Следующим этапом предобработки будет перевод всех символов текста в нижний регистр.

Необходимость этого заключается в том, что слова в разных регистрах, вроде: Сыр, сЫр, сыР и т.д. необходимо считать, как одно. Поэтому перевод в нижний регистр символов важен нам.

```
data = data.lower()
print(data)

до форума осталось 72 дня
```

Рис. 4: Предобработка текста. Перевод символов в нижний регистр.

2.2.3 Токенизация

Токенизация – это процесс разбиения текстового документа на отдельные части, которые, в свою очередь, называются токенами.

Natural language toolkit (библиотека **NLTK**) — популярный открытый пакет библиотек, используемых для разного рода задач NLP. Ей я и буду пользоваться для решения этой подзадачи, а также в ходе решения исходной задачи мы обратимся к ней еще не один раз.

Подробнее про данную библиотеку можно прочитать тут.

```
data = word_tokenize(data, language='russian')
print(data)

['до', 'форума', 'осталось', '72', 'дня']
```

Рис. 5: Предобработка текста. Токенизация.

2.2.4 Лемматизация

Следующим и не менее важным этапом является приведение слов к их начальной форме – или *лемматизация* слова.

Есть довольно схожий процесс – стеммизация, но он более грубый. Поскольку стемматизатор обрубает слова до корня. Например, если есть слово “*emotions*”, то в результате стеммизации получим “*emot*”, а если бы работали с лемматизацией получили бы “*emotion*”.

Несмотря на грубость, у стеммизации есть свои преимущества – это скорость работы данного алгоритма, она значительно выше, чем у лемматизации. Но несмотря на это преимущество, лемматизация дает более лаконичный вид слов, поэтому я буду пользоваться именно ей.

При работе с русским языком возникают некоторые трудности. Связаны они с тем, что для английского языка есть множество реализаций лемматизатора,

а для русского языка можно выделить следующие 3 библиотеки – `rumorphy`, `Natasha` и `rulemma`. Я буду пользоваться `rumorphy2`.

Давайте посмотрим, как это работает!

```
morph = MorphAnalyzer()
data = [morph.normal_forms(word)[0] for word in data]
print(data)

['до', 'форум', 'остаться', '72', 'день']
```

Рис. 6: Предобработка текста. Лемматизация токенов.

2.2.5 Удаление стоп-слов

Стоп-слова — это часто используемые слова, которые не вносят никакой дополнительной информации в текст. Слова вроде "об" "во" "что" и т.д. не несут никакой ценности и только добавляют шум в данные.

В библиотеке **NLTK** есть встроенный список стоп-слов, который можно использовать, для последующего удаления их из текста. Однако это не универсальный список, и для каждой задачи он может отличаться.

Так, например, в данной задаче я использую собственный список стоп-слов. Вот его инициализация:

```
with open('stop_ru.txt', 'r', encoding='utf8') as stop_file:
    stop_words = [word.strip() for word in stop_file.readlines()]
```

Рис. 7: Инициализация списка стоп-слов.

После инициализации мы начинаем исключать из текста те слова, которые находятся в списке стоп-слов.

```
with open('stop_ru.txt', 'r', encoding='utf8') as stop_file:
    stop_words = [word.strip() for word in stop_file.readlines()]
data = [word for word in data if word not in stop_words]
print(data)

['форум', 'остаться', '72']
```

Рис. 8: Предобработка текста. Удаление стоп-слов.

2.2.6 N-граммы

N-граммы — это комбинации из нескольких слов, идущих подряд.

N-граммы, где $N=1$ называются юниграммами (unigrams). Подобным же образом получают свои названия биграмы ($N=2$), триграммы ($N=3$ и т.д. N-граммы могут использоваться, когда нам нужно сохранить какую-то последовательность

данных, например, какое слово чаще следует за заданным словом. Юниграммы не будут содержать никакой последовательности данных, так как каждое слово берется индивидуально.

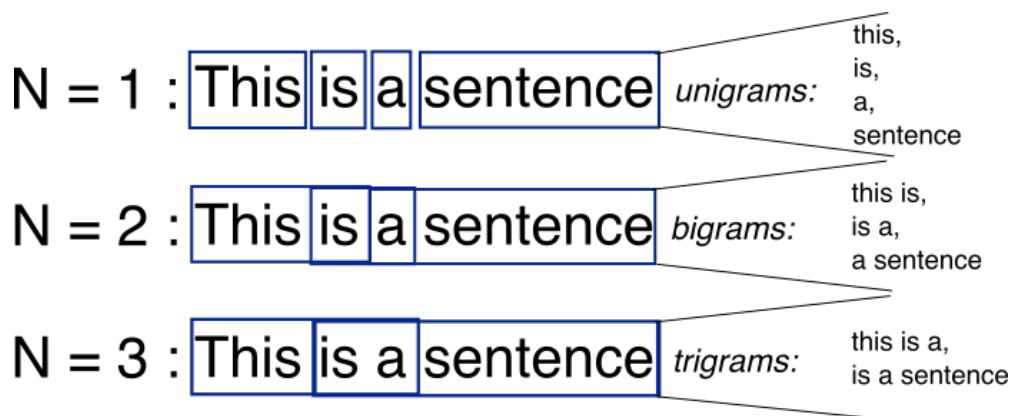


Рис. 9: N-граммы.

В поставленной задаче я буду использовать только юниграммы и биграммы, но также оставляю возможность воспользоваться и N-граммами более высоких порядков.

```
# Удаляем стоп-слова для случая с юниграммами. Иначе оставляем.
if k == 1:
    data = [word for word in data if word not in stop_words]

# Работаем с n-граммами.
if k > 1:
    data = list(ngrams(data, k))
```

Рис. 10: Условия разбиения на N-граммы в предобработке данных.

Возвращаясь к числу k , про которое мы говорили в самом начале – оно как раз-таки и будет отвечать за n-граммы.

Помимо всего, я реализовал `data_preprocessing` так, что если мы работаем с биграммами, триграммами... то стоп – слова удаляться не будут. Поскольку их роль при работе с такими разбиениями значительно выше.

Для реализации n-грам я снова воспользовался библиотекой **NLTK**:

```
from nltk.util import ngrams
```

Рис. 11: Импортирование библиотеки для работы с N-граммами.

На этом мы закончили с предварительной обработкой текста! Перейдем к следующему этапу.

2.3 Векторизация текстовых данных

Одним из главных подходов в NLP является *векторизация слова*. Процесс конвертации текста в числа и называется *векторизацией*.

```
print(list(ngrams(data,1)))
print(list(ngrams(data,2)))

[('форум',), ('остаться',), ('72',)]
[('форум', 'остаться'), ('остаться', '72')]
```

Рис. 12: Предобработка текста. Разбиение текста на юниграммы и биграмы.

Теперь после предварительной обработки текста нам необходимо его векторизовать.

Давайте рассмотрим некоторые решения в этой области.

2.3.1 «Мешок слов» (Bag of words (BOW))

Мешок слов — это одна из наиболее простых методик векторизации текста. Можно отметить, что в логике BOW два предложения могут называться одинаковыми, если содержат один и тот же набор слов. Почему так происходит разберем далее.

Давайте рассмотрим два предложения:

S1: Without music life would be a mistake

S2: Radiohead are a great music band

Важно отметить, что задачах NLP, каждое текстовое предложение называется *документом*, а несколько таких документов называют *корпусом текстов*.

BOW создает словарь уникальных d слов в корпусе (собрание всех токенов в данных). Например, корпус на изображении выше состоит из всех слов предложений *S1* и *S2*.

Теперь мы можем создать таблицу, где столбцы соответствуют входящим в корпус уникальным d словам, а строки предложениям (документам). Мы будем устанавливать значение 1, если слово в предложении есть, и 0, если его там нет.

	without	music	life	would	be	a	mistake	Radiohead	are	great	band
S1	1	1	1	1	1	1	1	0	0	0	0
S2	0	1	0	0	0	1	0	1	1	1	1

Рис. 13: Матрица результата обработки строк методом BOW

Это позволит нам создать dxn матрицу, где d это общее число уникальных токенов в корпусе и n равно числу документов. В примере выше матрица будет иметь форму **11x2**.

При решении задачи я буду брать ключевые слова из документа целиком, а не из одного предложения.

Для этого я буду брать корпус как единое предложение, а после чего находить в нем токены с наибольшим значением BOW. Это несколько отличается от изначальной задумки BOW, где мы берем предложение, и находим в нем ключевые слова, но идея в моем подходе абсолютна также самая.

Для реализации BOW я воспользуюсь функцией *FreqDist* из **NLTK**. А уже для подсчета слов с наибольшим значением BOW буду пользоваться встроенной во *FreqDist* функцией *most_common*, которая как раз таки и выводит самые частые слова:

```
data_1 = "Был холодный ясный апрельский день, и часы пробили тринадцать."
data_2 = "Уинстон Смит торопливо шмыгнул за дверь в холодный и ясный апрельский день."
data_3 = data_1 + data_2
print(pd.Series(FreqDist(data_preprocessing(data_3, 1))))
FreqDist(data_preprocessing(data_3, 1)).most_common(3)
```

```
холодный      2
ясный         2
апрельский    2
часы          1
пробить       1
уинстон       1
смит          1
торопливо     1
шмыгнуть      1
дверь         1
dtype: int64
```

```
[('холодный', 2), ('ясный', 2), ('апрельский', 2)]
```

Рис. 14: Результат извлечения слов, с наибольшим значением BOW.

Теперь давайте перейдем к более замысловатому методу, а именно TF-IDF.

2.3.2 Обратная частота документа (TF-IDF)

Term Frequency - Inverse Document Frequency

Weight rare words higher than common words

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
 df_i = number of documents containing i
 N = total number of documents

	without	music	life	would	be	a	mistake	Radiohead	are	great	band
s1	0.3	0	0.3	0.3	0.3	0	0.3	0	0	0	0
s2	0	0	0	0	0	0	0	0.3	0.3	0.3	0.3

Рис. 15: Матрица результата обработки строк методом TF-IDF

Расшифровывается данное понятие как Term Frequency (TF)-Inverse Document Frequency (IDF).

Частота слова (Term Frequency):

Term Frequency высчитывает вероятность найти какое-то слово в документе. Например, мы хотим узнать, какова вероятность найти слово W_i в документе D_j .

Term Frequency (W_i, D_j) = Количество раз, которое W_i встречается в D_j / Общее число слов в D_j

Обратная частота документа (Inverse Document Frequency):

В логике IDF, если слово встречается во всех документах, оно не очень полезно. Так определяется, насколько уникально слово во всем корпусе.

$$IDF(W_i, D_c) = \log(N/n_i)$$

Здесь D_c = Все документы в корпусе,

N = Общее число документов,

n_i = документы, которые содержат слово (W_i).

Если W_i встречается в корпусе часто, значение IDF снижается.

Если W_i используется не часто, то n_i снижается и вследствие этого значение IDF возрастает.

$$TF(W_i, D_j) * IDF(W_i, D_c)$$

TF-IDF — умножение значений TF и IDF. Большой вес получают слова, которые встречаются в документе чаще, чем во всем остальном корпусе.

При реализации решения задачи я буду строить DataFrame, в котором буду находить слова с наибольшим весом в каждом предложении:

```
tmp = pd.DataFrame()
tmp['Text'] = sent_tokenize(txt)
tmp['Text'] = tmp['Text'].apply(lambda text: data_preprocessing(text, 1))
tmp_list = tmp['Text'].to_list()
for i in range(len(tmp_list)):
    tmp_list[i] = ' '.join(tmp_list[i])
print(tmp_list)

vectorizer_uni = TfidfVectorizer(analyzer='word', ngram_range=(1,1), stop_words = stop_words)
vectorizer_uni.fit(tmp_list)
tfidf_uni = vectorizer_uni.transform(tmp_list)
vectors_uni = vectorizer_uni.get_feature_names_out()
smatrix_uni = vectorizer_uni.transform(tmp_list)
dense_uni = smatrix_uni.todense()
dense_list_uni = dense_uni.tolist()
df_tfidf_uni = pd.DataFrame(dense_list_uni, columns=vectors_uni)
print(df_tfidf_uni)
```

Рис. 16: TF-IDF. Получение DataFrame с результатом работы алгоритма.

Также рассмотрим часть кода, в котором реализован отбор слов, с наивысшим значением TF-IDF.

```
lst_words_uni = []
tmp_uni = []
result_uni_tfidf = []
for i in range(df_tfidf_uni.shape[0]):
    lst_words_uni.append(df_tfidf_uni.iloc[i].sort_values(ascending=False)[0:3])
    tmp_uni.append('KWUP ' + str(i) + ': ' + ', '.join(lst_words_uni[i].index))
    result_uni_tfidf.append(' '.join(tmp_uni))
print(result_uni_tfidf)
```

Рис. 17: TF-IDF. Извлечение слов, с наивысшим значением TF-IDF из предложений.

Пример работы данного алгоритма можно видеть ниже.

```

['цель книга исследовать часть язык инструментарий python возможно знать', 'опытный разработчик наверное многие
знакомый большой ждать появиться изучение', 'верно случай работать сложиться система изменение архитектура
компонент ради воспользоваться новый возможность язык дело частый']
python архитектура большой верно возможно возможность \
0 0.341426 0.000000 0.000000 0.000000 0.341426 0.000000
1 0.000000 0.000000 0.333333 0.000000 0.000000 0.000000
2 0.000000 0.261906 0.000000 0.261906 0.000000 0.261906

воспользоваться дело ждать знакомый ... работать ради \
0 0.000000 0.000000 0.000000 0.000000 ... 0.000000 0.000000
1 0.000000 0.000000 0.333333 0.333333 ... 0.000000 0.000000
2 0.261906 0.261906 0.000000 0.000000 ... 0.261906 0.261906

разработчик система сложиться случай цель частый часть \
0 0.000000 0.000000 0.000000 0.000000 0.341426 0.000000 0.341426
1 0.333333 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
2 0.000000 0.261906 0.261906 0.261906 0.000000 0.261906 0.000000

язык
0 0.259663
1 0.000000
2 0.199186

[3 rows x 32 columns]
['KWUP 0: python, знать, часть', 'KWUP 0: python, знать, часть. KWUP 1: знакомый, ждать, многие', 'KWUP 0: python,
знать, часть. KWUP 1: знакомый, ждать, многие. KWUP 2: компонент, архитектура, частый']

```

Рис. 18: TF-IDF. Пример результата работы.

Также в моей реализации, как я и упоминал выше, есть работа с биграмами. Так можно будет сравнить качество извлечением, по сравнению с юниграммами.

В целом, с ознакомительной частью и реализации в моем коде мы ознакомились, давайте теперь перейдем к самой задаче, которую мы будем решать.

2.4 Постановка задачи

Сама задача звучит следующим образом: “Дан набор csv-файлов с русскоязычными текстами. Необходимо извлечь ключевые слова этих текстов.”

При решении данной задачи будем пользоваться уже полученными навыками по методам NLP – предобработаем текст, произведем векторизацию слов и получим ключевые слова.

Остается грамотно организовать извлечение данных из файлов и сохранение результатов. Для этого я пользуюсь библиотекой **OS** которая заточена под работу с операционной системой.

2.5 Решение задачи

Решение было реализовано следующим образом – прежде всего я реализовал то, о чем было сказано в части про реализацию алгоритмов NLP. Помимо всего, извлечение ключевых слов было реализовано как для юниграм, так и для биграм.

Поскольку файлов с данными было довольно много, было необходимо организовать грамотную работу с ними, в частности, автоматическое сохранение данных в DataFrame и последующую его обработку.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Социальн	Владелец	Текст	Эмоцион:	Дата	Коммент:	Лайков	Репостов	Просмот:	Негативн:	Позитивн:	Нейтраль:	Неопреде	Вежливы:	Юморист
2	Вконтакте	Обитель Г	Когда раз	Нейтраль	2021-08-1	2	33	11	1881	0	1	1	0	0	0
3	Вконтакте	Обитель Г	Будни HR-	Нейтраль	2021-08-1	0	42	14	2006	0	0	0	0	0	0
4	Вконтакте	TechRocks	Прокачай	Позитив	2021-08-1	0	3	4	14182	0	0	0	0	0	0
5	Вконтакте	TechRocks	Пытаюсь	Негатив	2021-08-1	9	225	60	10808	0	0	5	0	0	0
6	Вконтакте	IT Way Liv	Подкаст.	Позитив	2021-08-1	0	5	1	368	0	0	0	0	0	0
7	Вконтакте	IT-Academ	КТО, КАК,	Позитив	2021-08-1	0	2	1	163	0	0	0	0	0	0
8	Вконтакте	Web /> НТ	Мини-игр	Позитив	2021-08-1	8	91	115	6164	1	0	1	0	0	0
9	Вконтакте	Kaspersky	#KDaily@k	Нейтраль	2021-08-1	1	15	4	2505	0	0	1	0	0	0
10	Вконтакте	Web /> НТ	Когда реч	Нейтраль	2021-08-1	3	120	29	11123	0	0	2	0	0	0
11	Вконтакте	iT News	Даунгрей,	Нейтраль	2021-08-1	0	13	8	1552	0	0	0	0	0	0
12	Вконтакте	GEEK PICT	Любите к	Позитив	2021-08-1	0	13	14	33167	0	0	0	0	0	0
13	Вконтакте	Тинькофф	Грузия от.	Нейтраль	2021-08-1	57	159	137	31533	3	0	9	0	0	0
14	Вконтакте	TechSkills	Привет, д	Нейтраль	2021-08-1	0	38	81	3437	0	0	0	0	0	0
15	Вконтакте	Scream Sci	За какими	Нейтраль	2021-08-1	0	6	4	1059	0	0	0	0	0	0
16	Вконтакте	Берза	Междунац	Нейтраль	2021-08-1	0	0	1	28	0	0	0	0	0	0
17	Вконтакте	iT News	Сбылась	Нейтраль	2021-08-1	14	180	203	75944	3	1	3	0	0	0
18	Вконтакте	Конферен	Денис Бе	Нейтраль	2021-08-1	0	2	3	307	0	0	0	0	0	0
19	Вконтакте	Calltouch	Последни	Позитив	2021-08-1	0	25	0	4073	0	0	0	0	0	0

Для поставленной задачи важен столбец текст, с которым мы будем работать, все остальное является вторичным.

Давайте также рассмотрим, что мы получаем после работы алгоритма.

	A	B	C	D	E	F	G	H	I	J	K
1		Text	Unigrams	Bigrams	Unigrams_BoW	Bigrams_BoW	Unigrams_BoW_result	Bigrams_BoW_result	TF-IDF.UNI	TF-IDF.BI	
2	0	Когда раз	['разбаа', '([''когда', '(['('разрабатыват	['([''когда', 'разз	['разрабатывать', 1], 'п	['([''когда', 'разрабатыв	KWUP	0: пр	KWBP	0: для своего	
3	1	Будни HR	['буди', '([''будни', '([''будни', 1], 'hr	['([''будни', 'hr'), '([''будни', 1], 'hr'), 1], 'ов	['([''будни', 'hr'), 1], '([''hr	KWUP	0: hr, KWBP	0: hr ов, буд			
4	2	Прокачай	['прокачай', '([''данные'), 3], '(['('data', 'scient	['('данные'), 3], 'data'), 3], '(['('data', 'scientist'), 2], 'KWUP	0: пр	KWBP	0: прокачай				
5	3	Пытаюсь	['пытаться', '(['пытаться', 1], '(['пытаться', '(['пытаться', 1], 'разобр	['пытаться', 1], 'разобр	KWUP	0: ош	KWBP	0: ош			
6	4	Подкаст	['подкаст', '(['https', 3], 'bit'), '(['bit', 'ly'), 3], '(['https', 3], 'bit'), 3], 'ly'), '(['bit', 'ly'), 3], '(['https', 3], 'KWUP	0: по	KWBP	0: 08 2021, ве					
7	5	КТО, КАК,	['бизнес', '([''кто', 'ка', '(['бизнес'), 5], '(['('b', 'it'), 3], '(['('бизнес'), 5], 'it'), 3], 'ar	['('b', 'it'), 3], '(['('бизнес', 5], 'it'), 3], '(['('бизнес', 5], 'KWUP	0: 02, KWBP	0: как когда, и					
8	6	Мини-игр	['мины', '([''мины', '([''игр', '([''мины', 'игра	['('мины', 'игра	['('мины', 1], 'игра'), 1], '(['('мины', 'игра'), 1], '([''и	KWUP	0: jay	KWBP	0: для детей,		
9	7	#KDaily	['k'daily', 'k', '(['k'daily', 'k', '(['k'daily', 'kasp	['('k'daily', 2], 'k'daily'), 1], 'k', '(['k'daily', 'kaspersky'), 1	KWUP	0: kas	KWBP	0: kaspersky p			
10	8	Когда реш	['решить'), '([''когда', '([''решить'), 1], 'п	['([''когда', 'реш	['решить'), 1], 'пофиски	['([''когда', 'решить'), 1], 'KWUP	0: ба	KWBP	0: ба	итого, и	
11	9	Даунгрей	['даунгрей', '([''rog', 6], 'phoni	['('rog', 'phone'), 6], '5	['('rog', 'phone'), 6], '6], 'p	KWUP	0: да	KWBP	0: 12	гб, от	
12	10	Любите к	['любить'), 'даунгрей', '(['киберспорт	['киберспорт	['киберспортивный'), 1], '([''киберспортивный'), 1], 'KWUP	0: лю	KWBP	0: любите	ки		
13	11	Грузия от	['грузия', '([''грузия', '([''грузия', 1], 'от	['([''грузия', 'отл	['('грузия', 1], 'отлично'), '(['('грузия', 'отлично'), 1	KWUP	0: ав	KWBP	0: автопутеш		
14	12	Привет, д	['привет', '([''привет', '(['python', 5], 'к	['([''это', 'книга	['('python', 5], 'книга'), 4], '([''это', 'книга'), 2], '([''e	KWUP	0: др	KWBP	0: прив	дру	
15	13	За каими	['професс', '(['за', 'ка', '([''профессия', 3], '([''профессия', '([''профессия', 3], 'буду	['('профессия', 'буду	KWUP	0: пр	KWBP	0: за каими,			
16	14	Междуна	['между', '(['между', '(['optimization', '([''optimization', '([''optimization', 2], '202	['('optimization', '2021') KWUP	0: sec	KWBP	0: се	KWBP	0: се	конфе	
17	15	Сбылась	['сбыться', '(['sбыться', '(['galaxy', 7], 's2	['([''galaxy', 's22	['('galaxy', 7], 's22'), 6], 's	['([''galaxy', 's22'), 6], 'sr	KWUP	0: ме	KWBP	0: сб	
18	16	Денис Бе	['денис', '([''денис', '([''nor', 4], 'click	['([''nor', 'b'), 3], '([''nor', 4], 'clickhouse', 3	['([''nor', 'b'), 3], '([''kafka', 3], 'KWUP	0: бе	KWBP	0: до	клад	на	
19	17	Последн	['послед', '([''рольф', 2], 'в	['([''последний', '([''рольф', 2], 'выпус	['([''последний', 'шанс'), 1], 'KWUP	0: ро	KWBP	0: ме	от, др		
20	18	Бесплатн	['бесплат', '(['бесплат', '(['vue', 4], 'js'), 4], '(['vue', 'js'), 4], '(['vue', 4], 'js'), 4], 'бес	['('vue', 'js'), 4], 'бес	KWUP	0: ч	KWBP	0: ж			
21	19	Помните	['помнить', '([''пиво', 6], 'бел	['([''бельгийски	['('пиво', 6], 'бельгийски	['('бельгийский', 'брас	KWUP	0: ра	KWBP	0: рассказы	
22	20	Красочн	['красочн', '(['красочн', '(['красочный'), 1], '([''красочный', '([''красочный', 1], 'тутор	['('красочный', 'тutori	KWUP	0: be	KWBP	0: believe			
23	21	Очередн	['очеред', '(['очеред', '(['очередной'), 1], '([''очередной', '([''очередной', 1], 'инте	['('очередной', 'интер	KWUP	0: by	KWBP	0: by codin			
24	22	#educati	['educati', '(['educati', '(['работ', 3], '([''education', '([''работ', 3], 'карьер	['('education', 'changeli	KWUP	0: car	KWBP	0: careerplat			
25	23	Пользова	['пользов', '(['пользов', '(['пользователь'), '([''коммуника	['('пользователь', 3], '([''коммуникация', 'c'), KWUP	0: зн	KWBP	0: это	еще, i			
26	24	Demon Bu	['demon', '([''demon', '([''demon', 1], 'bust	['([''demon', 'bus	['('demon', 1], 'bust'), 1], '([''demon', 'bust'), 1], '([''KWUP	0: bus	KWBP	0: bust	chief, i		
27	25	Воскрес	['воскрес', '([''воскрес', '([''000', 6], 'http', '([''http', 'amp'), 6], 'http'), 6], 'http'), 6], 'am	['([''http', 'amp'), 5], '([''am	KWUP	0: су	KWBP	0: ко	мо		
28	26	BlackBerry	['blackber', '(['blackber', '(['blackberry', 3], '([''blackberry', '([''blackberry', 3], 'passp	['('blackberry', 'passport	KWUP	0: bla	KWBP	0: клавиату			

16

У каждого столбца есть собственное назначение и результат:

- **Text** – исходный текст, с которым мы будем работать.
- **Unigrams** – полученные юниграммы, после предварительной обработки текста.
- **Bigrams** – полученные биграмы, после предварительной обработки текста.
- **Unigrams_BOW** – полученная оценка для всех юниграмм, методом BOW.
- **Bigrams_BOW** – полученная оценка для всех биграмм, методом BOW.
- **Unigrams_BOW_result** – 5 юниграмм из корпуса (текста), с наивысшей оценкой по методу BOW.
- **Bigrams_BOW_result** – 5 биграмм из корпуса (текста), с наивысшей оценкой по методу BOW.
- **TF-IDF.UNI** – результат извлечения 3 юниграмм из каждого документа (предложения) в корпусе с наибольшим показателем TF-IDF.
- **TF-IDF.BI** – результат извлечения 3 биграмм из каждого документа (предложения) в корпусе с наибольшим показателем TF-IDF.

2.6 Применение результатов извлечения ключевых слов

После того, как мы создали инструмент и получили некоторые результаты, давайте воспользуемся ими.

Например, одним из применений полученных данных можно считать извлечение самых популярных слов из файлов.

Для реализации алгоритма будем складывать все юниграммы и биграмы в соответствующие им словари, а после чего будем извлекать и записывать в DataFrame 10 наиболее популярных слов среди всех элементов данного xlsx-файла:

```
for i in range(df_w['Unigrams'].shape[0]):
    wrd_uni_bow = FreqDist(df_w['Unigrams'][i]).most_common(len(FreqDist(df_w['Unigrams'][i])))
    for j in range(len(wrd_uni_bow)):
        if wrd_uni_bow[j][0] in dict_uni:
            dict_uni[wrd_uni_bow[j][0]] += wrd_uni_bow[j][1]
        else:
            dict_uni[wrd_uni_bow[j][0]] = wrd_uni_bow[j][1]
df_kwe[LoF[FileIndex] + ' unigram keywords'] = Counter(dict_uni).most_common(10)
dict_uni.clear()
```

Рис. 21: Извлечение самых популярных слов.

После обработки всех `xlsx`-файлов, мы получим еще один `xlsx`-файл с названием `ROF_MPW` (*Result of finding most popular words – результат поиска самых популярных слов*)

	A	B	C
1		Август 1 половина.xlsx unigram keywords	Август 1 половина.xlsx bigram keywords
2	0	('https', 994)	(('bit', 'ly'), 175)
3	1	('компания', 602)	((('https', 'bit'), 170)
4	2	('ru', 494)	((('один', 'из'), 123)
5	3	('новый', 445)	((('а', 'также'), 116)
6	4	('работа', 366)	((('ссылка', 'https'), 112)
7	5	('2021', 352)	((('clck', 'ru'), 112)
8	6	('проект', 333)	((('по', 'ссылка'), 108)
9	7	('август', 281)	((('https', 'clck'), 108)
10	8	('1', 256)	((('http', 'amp'), 99)
11	9	('система', 249)	((('amp', 'gs'), 99)

Рис. 22: Результат извлечения самых популярных слов.

В результате обработки данных получим самые популярные юниграммы и биграмы из `xlsx`-файлов.

3 Заключение

3.1 Анализ полученных данных

Давайте проанализируем то, что мы получили.

1. Мы имеем набор `xlsx` – файлов, с извлеченными ключевыми словами двумя способами – BOW и TF-IDF (при учете разбиения случаев на юниграммы и биграмы мы получаем 4 столбца с результатами).
2. Помимо всего, мы имеем `xlsx` – файл, с наиболее популярными словами, из `csv` – файлов.

Теперь рассмотрим сами полученные данные:

3.2 Извлечение ключевых слов

Что же можно сказать о качестве извлечения ключевых слов? Говоря про конкретные данные, порой алгоритму очень непросто работать по той простой причине, что в значительной части текстов нет повторяющихся слов. Поэтому алгоритмам труднее определять ключевые слова, но давайте проанализируем полученный результат:

1. Например, вот такой текст: “*Комик и телеведущий Хуан Хойя Борха, известный во всем мире как «хохочущий испанец» или Эль Риситас (El Risitas), умер 28 апреля в возрасте 65 лет.*”

BOW_UNIGRAMS	BOW_BIGRAMS	TF-IDF.UNI	TF-IDF.BI
[('комик', 1), ('телеведущий', 1), ('хуан', 1), ('хойя', 1), ('борх', 1)]	[(('комик', 'и'), 1), (('и', 'телеведущий'), 1), (('телеведущий', 'хуан'), 1), (('хуан', 'хойя'), 1), (('хойя', 'борх'), 1)]	KWUP 0: 28, 65, хуан	KWBP 0: 28 апреля, 65 лет, хуан хойя

Причиной такого извлечения ключевых слов как раз таки является уникальность каждого из слов – поэтому не получается из текста вычленить какие-то особенные слова.

2. Давайте рассмотрим более удачный пример:

“Лекция Сергея Гимельрейха: «Идея игры через ментальное моделирование».

Сергей поделится своей новой методикой придумывания и развития игрового процесса и механик, опираясь на заданную тему.

Лекция будет полезной для всех авторов игр, в частности для участников джемов

Сергей Гимельрейх — основатель Индикатора, преподаватель ИТМО и ВШБИ, игровой продюсер, гейм-дизайнер и популяризатор гейм-дизайна в России.

Смотри лекцию на нашем Ютуб-канале, 22 апреля, в 19:00 (Мск):

<https://www.youtube.com/c/IndikatorSpace>

#Лекция@indieonline #Джем@indieonline”

BOW_UNIGRAMS	BOW_BIGRAMS	TF-IDF.UNI	TF-IDF.BI
[('лекция', 4), ('сергей', 3), ('гимельрейх', 2), ('игра', 2), ('игровой', 2)]	[(('сергей', 'гимельрейх'), 2), (('лекция', 'сергей'), 1), (('гимельрейх', 'идея'), 1), (('идея', 'игра'), 1), (('игра', 'через'), 1)]	KWUP 0: ментальный, моделирование, идея. KWUP 1: методика, поделится, механик. KWUP 2: гейм, продюсер, автор.	KWBP 0: гимельрейха идея, через ментальное, лекция сергея KWBP 1: развития игрового, своей новой, поделится
		KWUP 3: indieonline, лекция, 00	своей KWBP 2: дизайна россии, популяризатор гейм, игровой продюсер KWBP 3: 00 мск, апреля 19, смотри лекцию

Здесь результат уже гораздо лучше и понятнее – есть некий *Сергей Гумельрейх*, который ведет *лекции*, по *игровой* тематике.

3.3 Извлечение самых популярных слов (ROF_MPW)

По итогу анализа наиболее популярных слов можно сделать некоторые выводы. Например, почти во всех строках (полный файл ROF_MPW находится по ссылке) можно видеть, что наиболее популярным словом является “https”, то есть ссылки пользователи вставляют любят.

Также, например, логичным является то, что во второй половине декабря такие биграммы, как “новый год”, “2021 год” являются наиболее популярными.

Декабрь 2 половина.xlsx unigram keywords	Декабрь 2 половина.xlsx bigram keywords
('https', 1680)	((('https', 'www'), 312)
('ru', 945)	((('новый', 'год'), 207)
('компания', 864)	((('2021', 'год'), 205)
('новый', 816)	((('по', 'ссылка'), 178)
('работа', 555)	((('https', 'clck'), 168)
('проект', 541)	((('clck', 'ru'), 168)
('2021', 522)	((('а', 'также'), 164)
('it', 439)	((('2022', 'год'), 157)
('данные', 369)	((('ссылка', 'https'), 146)
('python', 367)	((('один', 'из'), 144)

Рис. 23: Результат извлечения самых популярных слов. Декабрь.

Август 1 половина	Август 2 половина	Апрель 1 половина	Апрель 2 половина.
('https', 994)	('https', 1235)	('https', 952)	('https', 1026)
('компания', 602)	('компания', 744)	('компания', 566)	('ru', 519)
('ru', 494)	('ru', 658)	('ru', 498)	('компания', 465)
('новый', 445)	('новый', 532)	('it', 424)	('2021', 418)
('работа', 366)	('проект', 465)	('апрель', 415)	('работа', 416)
('2021', 352)	('2021', 454)	('проект', 380)	('проект', 388)
('проект', 333)	('работа', 432)	('работа', 366)	('it', 327)
('август', 281)	('команда', 356)	('2021', 326)	('новый', 313)
('1', 256)	('сентябрь', 321)	('новый', 301)	('com', 281)
('система', 249)	('программа', 309)	('команда', 278)	('команда', 279)

Рис. 24: Некоторые результаты из ROF_MPW.

В заключении можно сказать, что зачастую результат извлечения ключевых слов зависит от самого текста – вычленять ключевые слова в коротких постах – задача отнюдь непростая.

Но, я считаю, что мы смогли ознакомиться с основами мира NLP.

3.4 Приложение

Ссылка на проект и полученные файлы: <https://github.com/HpPpL/NLP.git>.

Часть II

Выявление превалирующих качественных характеристик ключевых слов (автор: Батарева Софья)

4 Выявление превалирующих качественных характеристик ключевых слов

4.1 Введение

Анализ и обработка текстов на естественном языке является постоянно актуальной задачей. Текстовая классификация является одной из самых популярных практических задач NLP.

Одна из классических задач в текстовой классификации — сентиментный анализ. Например, классификация отзывов на положительные, отрицательные и нейтральные. Эта задача выполняется с помощью выявления превалирующих качественных характеристик ключевых слов.

Тут нам на помощь приходят другие инструменты – морфологические анализаторы. В python два самых популярных: `rumorphy` и `rumystem`. На вход подаем слово, а на выходе получаем его в нормализованном виде. Такой подход называется Лемматизация. Наша задача в конечном итоге - лемматизировать текст. Данную часть проекта было удобнее всего выполнить, используя библиотеку ‘`Rumorphy2`’. (**Rumorphy2** - морфологический анализатор для русского языка, написанный на языке Python и использующий словари из OpenCorpora.)

4.2 Надо установить библиотеку ‘Py morphology2’ с помощью pip:

```
[ ] pip install pymorphy2

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pymorphy2
  Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
    | 55 kB 1.4 MB/s
Requirement already satisfied: docopt>=0.6 in /usr/local/lib/python3.7/dist-packages (from pymorphy2) (0.6.2)
Collecting dawg-python>=0.7.1
  Downloading DAWG_Python-0.7.2-py2.py3-none-any.whl (11 kB)
Collecting pymorphy2-dicts-ru<3.0,>=2.4
  Downloading pymorphy2_dicts_ru-2.4.417127.4579844-py2.py3-none-any.whl (8.2 MB)
    | 8.2 MB 10.9 MB/s
Installing collected packages: pymorphy2-dicts-ru, dawg-python, pymorphy2
Successfully installed dawg-python-0.7.2 pymorphy2-0.9.1 pymorphy2-dicts-ru-2.4.417127.4579844
```

Рис. 25: Импорт модулей библиотек.

4.3 Затем импортируем в программу модули библиотек:

1. Py morphology2 – уже была описана.
2. Re – для регулярных выражений (специальная последовательность символов, которая помогает сопоставлять или находить строки python с использованием специализированного синтаксиса, содержащегося в шаблоне).
3. Codecs — реестр кодировок и базовых классов¶(этот модуль определяет базовые классы для стандартных кодировок Python (кодеры и декодеры) и обеспечивает доступ к внутреннему реестру кодировок Python, который управляет процессом поиска кодировок и обработкой ошибок).

```
✓ 0 сек. #Перед началом работы работы надо создать объект класса MorphAnalyzer
import pymorphy2
import pymorphy2.tokenizers as tokenizers
import re
import codecs

morph = pymorphy2.MorphAnalyzer()
```

Рис. 26: Список используемых библиотек и функций.

4.4 Подключаем к программе два текстовых файла:

'1.txt' – файл, в который мы будем записывать тест для анализа

'Synonym.txt' – файл, в который программы будет выводить итоговый словарь

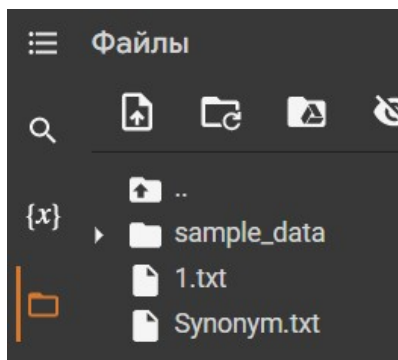


Рис. 27: Текстовые файлы.

4.5 Токенизируем текст:

Токенизация – процесс разбиения текста на текстовые единицы (на слова или на предложения). В данном случае, на слова, следовательно, разделителем является пробел (функция `line.split()`).

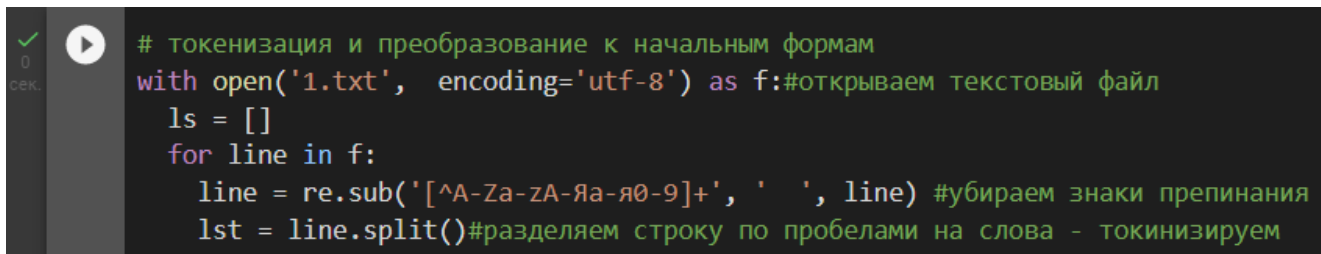


Рис. 28: Модуль токенизации.

Перед токенизацией убираем знаки препинания функцией `re.sub`.

Уберем знаки препинания и токенизируем текст из файла:

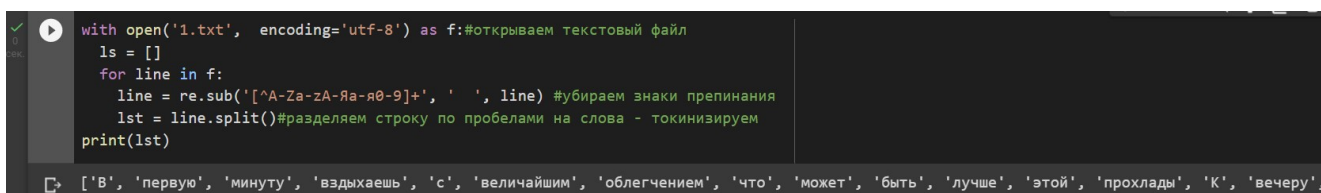


Рис. 29: Вывод функции.

4.6 Теперь надо лемматизировать получившиеся слова (привести их к начальной форме):

Над каждым словом проводим морфологический анализ и выявляем его начальную форму.

Например, *лечу*, *летят*, *летели* имеют начальную форму *лететь*.

Воспользуемся функцией `morph.parse(word)` для разбора слова ‘лечу’:

```
morph.parse("лечу")
[Parse(word='лечу', tag=OpenCorporaTag('VERB,impf,intr sing,1per,pres,indc'), normal_form='лететь', score=0.5, methods_stack=((DictionaryAnalyzer(), 'лечу', 1940, 1))),
Parse(word='лечу', tag=OpenCorporaTag('VERB,impf,tran sing,1per,pres,indc'), normal_form='лечить', score=0.5, methods_stack=((DictionaryAnalyzer(), 'лечу', 1943, 1)))]
```

Рис. 30: Морфологический разбор слова.

Метод `parse` возвращает список объектов `parse`, которые обозначают виды грамматических форм анализируемого слова. Такой объект обладает следующими атрибутами:

1. Tag обозначает набор грамем. В данном случае слово *хочу* — это глагол (VERB) несовершенного вида (impf), переходный (tran), единственного числа (sing), 1 лица (1per), настоящего времени (pres), изъявительного наклонения (indc);
2. Normal form — нормального форма слова;
3. Score — оценка вероятности того, что данный разбор правильный;
4. Methods_stack — тип словаря распарсенного слова с его индексом.

Теперь выбираем нормальную форму слова. По умолчанию объекты Parse сортированы в порядке убывания значения score. Поэтому из списка надо выбрать 1-й элемент (нулевой):

```
morph.parse("лечу")[0]
Parse(word='лечу', tag=OpenCorporaTag('VERB,impf,intr sing,1per,pres,indc'), normal_form='лететь', score=0.5, methods_stack=((DictionaryAnalyzer(), 'лечу', 1940, 1),))
```

Рис. 31: Таким образом, мы получим одно слово из разных его форм

4.7 Токенизируем и лемматизируем текст из файла:

```
# токенизация и преобразование к начальным формам
with open('1.txt', encoding='utf-8') as f: #открываем текстовый файл
    ls = []
    for line in f:
        line = re.sub('[^A-Za-zА-Яа-я0-9]+', ' ', line) #убираем знаки препинания
        lst = line.split() #разделяем строку по пробелами на слова - токенизируем

        words = []
        for word in lst:
            p = morph.parse(word)[0] # делаем разбор
            words.append(p.normal_form)

        ls.append(words)

print(ls)

[['жаркий', 'июльский', 'день', 'когда', 'луч', 'солнце', 'так', 'жгучий', 'хотется', 'найти', 'место', 'прохладный', 'далеко', 'не', 'лёгкий',
```

Рис. 32: Обработанный текст из файла.

4.8 Создаем словарь начальных форм слов:

Первая грамма *tag* – часть речи.

Нас интересуют качественные характеристики, следовательно мы должны рассматривать качественные прилагательные. Они в основном стоят перед существительными, к которым относятся. Поэтому, мы будем записывать в словарь только те прилагательные, после которых стоят существительные:

```
result = []
dict_adjf = {} #Создаем словарь
for i in range(len(ls)): # i - номер предложения строки
    for j in range(len(ls[i])): # j - номер слова в строке
        if morph.parse(ls[i][j])[0].tag.POS == "ADJF": # первая(нулевая) позиция tag - ADJF
            if j <= len(ls[i]) - 1:
                if morph.parse(ls[i][j+1])[0].tag.POS == "NOUN": # первая(нулевая) позиция tag следующего слова - NOUN
                    result.append(ls[i][j])
                    if ls[i][j] in dict_adjf:
                        dict_adjf[ls[i][j]] += 1 # Если в словаре уже есть такое слово увеличиваем количество
                    else:
                        dict_adjf[ls[i][j]] = 1 # Если в словаре такого слова нет, добавляем
# print(result)
dict_adjf# выводим словарь

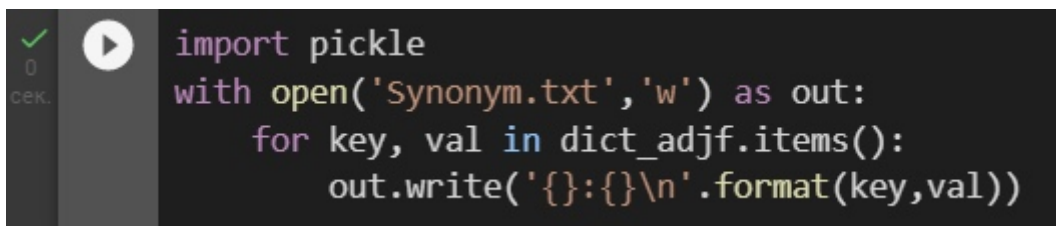
{'великий': 1,
 'восточный': 1,
 'глиняный': 1,
 'голубой': 1,
 'грушевый': 1,
 'гусиный': 1,
 'дальний': 1,
 'далёкий': 1,
 'деревянный': 1,
 'дерзкий': 1,
 'домашний': 1,
 'другой': 1,
 'знойный': 1,
 'июльский': 1,
 'каменный': 1,
 'кумачовый': 1,
```

```
'огромный': 1,  
'первый': 1,  
'рыбацкий': 1,  
'сероватый': 1,  
'собачий': 1,  
'соломенный': 1,  
'таинственный': 1,  
'туманный': 1,  
'холщовый': 1,  
'этот': 1}
```

Рис. 33: Список начальных форм качественных прилагательных.

4.9 Переносим полученный словарь в файл 'Synonym.txt'

Импортируем в программу модуль '*pickle*' (он реализует двоичные протоколы для сериализации и десериализации структуры объекта Python).



```
import pickle  
with open('Synonym.txt','w') as out:  
    for key, val in dict_adjf.items():  
        out.write('{}: {}\n'.format(key, val))
```

Рис. 34: Импорт модуля pickle.

4.10 Результат работы программы:

4.10.1 Тест 1:

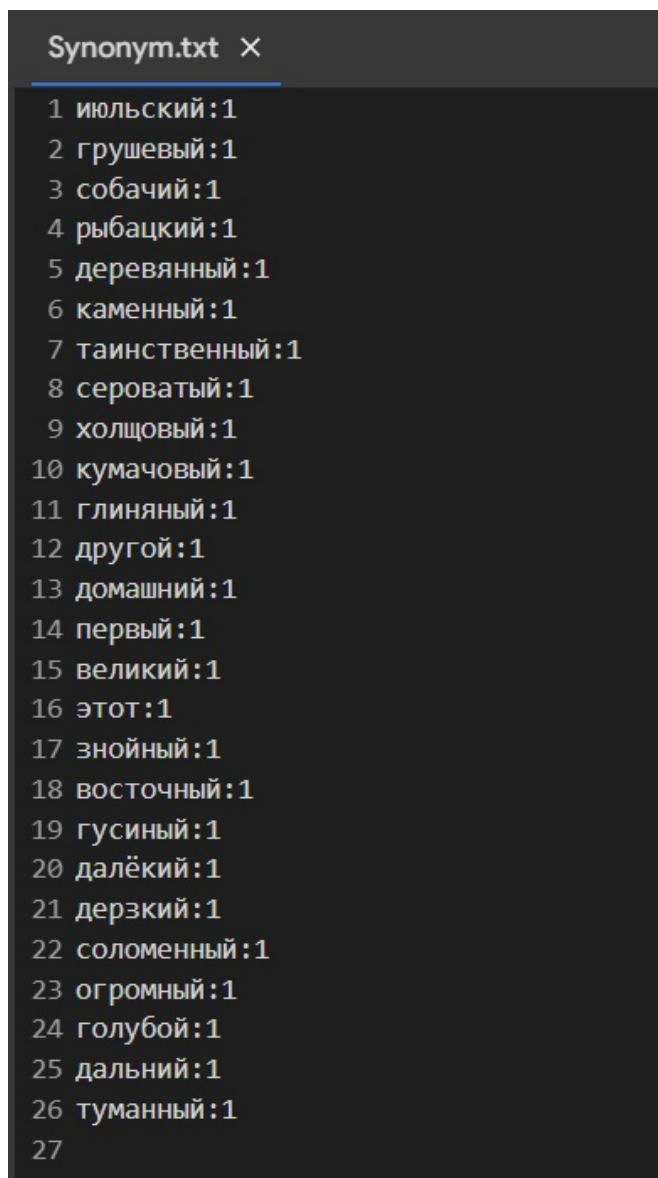


Рис. 35: Результат обработки файла '1.txt' в первом случае.

Жарким июльским днем, когда луч солнца так жгуч, хочется найти место прохладнее. Далеко не легкое это дело. Из большого грушевого сада, который наполнен зноем, идешь через двор мимо собачьей будки и рыбацких сетей, поднимаешься на деревянное крыльцо иходишь в прохладный каменный дом.

Ставни прикрыты, и поэтому здесь царят тишина и таинственный полумрак. Приглядевшись, начинаешь различать длинную сероватую лавку, холщовую рубашку и кумачовую косынку на ней, глиняный кувшин на столе и другую бабушкину домашнюю утварь.

В первую минуту вздыхаешь с величайшим облегчением: что может быть лучше этой прохлады?... К вечеру, после утомительного знойного дня, собирается гроза. Порывистый северо-восточный ветер налетает на сад, доносит гусиные крики с реки и далекие раскаты грома. Дерзкие порывы теребят соло-

менную крышу сарая и гонят серо-синюю огромную тучу, закрывшую нежно-голубой небосвод. Вот уже дальний лес скрылся за туманной пеленой дождя. . .

4.10.2 Тест 2:

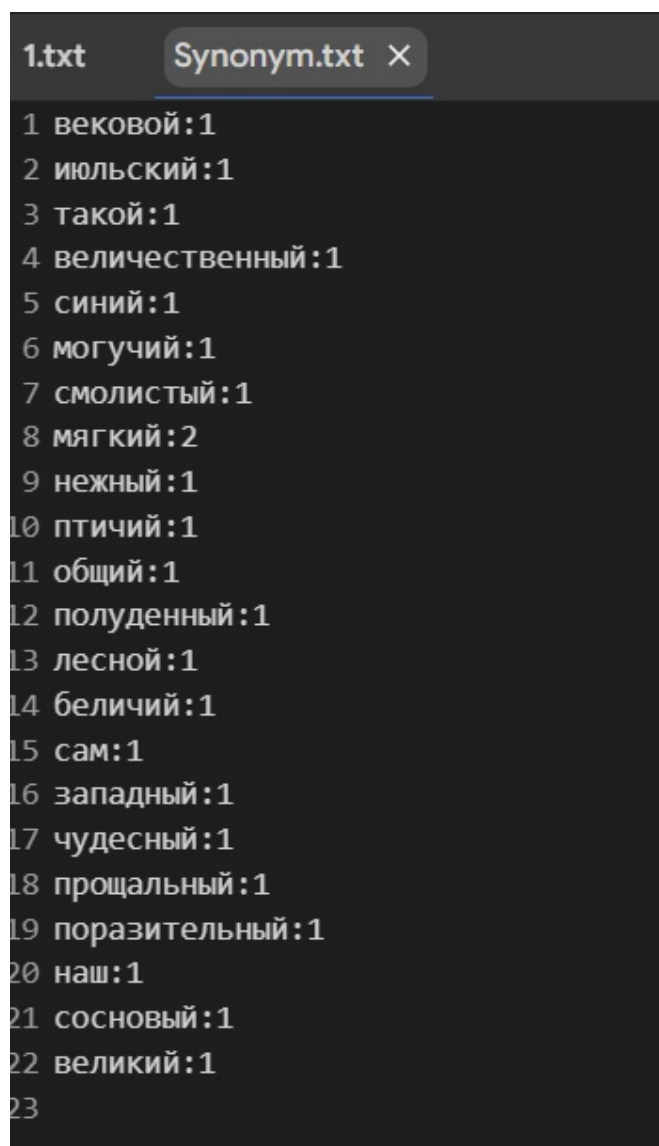


Рис. 36: Результат обработки файла '1.txt' во втором случае.

Хорош вековой бор погожим июльским днем. Чувствуешь себя в таком лесу как в величественном соборе: в синюю вышину гордо вознесли деревья свои роскошные могучие кроны. Лучи солнца пронизывают их, и воздух напоен смолистым запахом. Потоки горячего света льются вниз на мягкую подстилку из опавшей хвои. Кажется, что вместе с ними струятся нежнейшие звуки птичьих голосов, не нарушая общей торжественности и покоя. Как только полуденный жар начинает спадать, лесные звуки становятся слышнее. Около неуклюжего пня на мягкой земле заметны беличьи следы. Тени стволов становятся длиннее, а сами стволы с западной стороны светлее. Но самое чудесное зрелище – это бор в прощальных лучах заходящего солнца. Поразительное впечатление от как будто пылающих сосен долго не изгладится из нашей памяти. Совершив прогул-

ку по сосновому лесу, понимаешь, что природа – наше величайшее богатство и мы должны его беречь.

4.11 Текст программы:

```
pip install pymorphy2
import pymorphy2
import pymorphy2.tokenizers as tokenizers
import re
import codecs
#Перед началом работы работы надо создать объект класса MorphAnalyzer
morph = pymorphy2.MorphAnalyzer()
# токинизация и преобразование к начальным формам
with open('1.txt', encoding='utf-8') as f:#открываем текстовый файл
    ls = []
    for line in f:
        line = re.sub('[\A-Za-zА-Яа-я0-9]+', ' ', line) #убираем знаки препинания
        lst = line.split()#разделяем строку по пробелами на слова - токинизируем
        words = []
        for word in lst:
            p = morph.parse(word)[0] # делаем разбор
            words.append(p.normal_form)
        ls.append(words)
print(ls)
result = []
dict_adjf = {} #Создаем словарь
for i in range(len(ls)):# i - номер предложения строки
    for j in range(len(ls[i])):# j - номер слова в строке
        if morph.parse(ls[i][j])[0].tag.POS == "ADJF":# первая(нулевая) позиция tag - ADJF
            if j <= len(ls[i]) - 1:
                if morph.parse(ls[i][j+1])[0].tag.POS == "NOUN":# первая(нулевая) позиция tag следующего слова - NOUN
                    result.append(ls[i][j])
                    if ls[i][j] in dict_adjf:
                        dict_adjf[ls[i][j]] += 1# Если в словаре уже есть такое слово увеличиваем количество
                    else:
                        dict_adjf[ls[i][j]] = 1# Если в словаре такого слова нет, добавляем
# print(result)
dict_adjf# выводим словарь
import pickle
with open('Synonym.txt','w') as out:
    for key, val in dict_adjf.items():
```

```
out.write('{}:{}'.format(key,val))
```

4.12 Итог:

Данная программа предназначена для выявления преобладающих качественных характеристик ключевых слов, на основе составления словаря качественных прилагательных.

Часть III

Извлечение именованных сущностей (NER) (автор: Тупицкий Даниил)

5 NER

5.1 Что такое NER

Одна из самых популярных задач NLP это NER – поиск ключевых слов.

Имена людей, названия организаций, книг, городов, и другие имена собственные называют «именованные сущности» (named entities), а саму задачу — «распознавание именованных сущностей». По-английски «Named entity recognition» или коротко NER; это сокращение регулярно используется и в русскоязычных текстах.

5.2 Задача

За одной задачей NER, на самом деле, стоит две: 1) обнаружить, что какая-то последовательность слов (спан) — это именованная сущность; 2) понять, к какому классу (имя человека, название организации, город и т.п.) эта именованная сущность относится. На каждом из этапов возникают свои сложности.

Без решения задачи NER тяжело представить себе решение многих задач NLP, допустим, разрешение местоименной анафоры или построение вопросно-ответных систем. Местоименная анафора позволяет нам понять, к какому элементу текста относится местоимение. Например, пусть мы хотим проанализировать текст “Прискакал Чарминг на белом коне. Принцесса выбежала ему навстречу и поцеловала его”. Если мы выделили на слове “Чарминг” сущность Персона, то машина сможет намного легче понять, что принцесса, скорее всего, поцеловала не коня, а принца Чарминга.

5.3 Актуальность

Зачем решать задачу NER? Например вам приходит письмо на электронную почту, хорошо бы сделать сниппет только той части, где находится что-то важное а не просто «Здравствуйте, Даниил Сергеевич». Или другой пример: задаем любимому поисковику вопрос “Кто играл Дарта Вейдера в фильме Империя наносит ответный удар”, сразу же получаем ответ. Это тоже делается с помощью NER: выделяем сущности (фильм, персонаж и т. п.), понимаем что нас спрашивают, и дальше ищем ответ в базе данных.

Один из сценариев, когда решение задачи в классической постановке все-таки может понадобиться, — структуризация неструктурированных данных. Пусть у вас есть какой-то текст (или набор текстов), и данные из него нужно ввести в базу данных (таблицу). Классические именованные сущности могут соответствовать строкам такой таблицы или же служить содержанием каких-то ячеек. Соответственно, чтобы правильно заполнять таблицу, нужно перед этим выделить в тексте те данные, которые вы будете в нее вносить (обычно после этого есть еще один этап — идентификация сущностей в тексте, когда мы понимаем, что спаны “ООН” и “Организация Объединенных Наций” относятся к одной и той же организации; однако, задача идентификации или *entity linking* — это уже другая задача.

Как видим, постановка задачи очень гибкая. Другими словами, никто не заставляет нас выделять именно локации, персоны и организации. Мы можем выделять любые нужные нам непрерывные фрагменты текста, которые чем-то отличаются от остального текста. В результате можно подобрать свой набор сущностей для конкретной практической задачи, приходящей от заказчика, разметить корпус текстов этим набором и обучить модель. Это и делает NER одной из самых часто решаемых задач NLP.

5.4 Сложности решения

Почему NER не используется повсеместно? Классической сложностью являются разного рода неоднозначности в тексте (например омонимы и многозначные слова). Вторая сложность — техническая. У вас скорее всего возникнут пограничные случаи — когда нужно выделять сущность, когда не нужно, включать слово в спан или не включать и т. д.

Славянские языки, в том числе и русский, считаются довольно сложными для обработки. В основном, из-за богатой системы окончаний, свободного порядка слов и других морфологических и синтаксических явлений. Распознавание именованных сущностей (далее, NER) представляется трудной задачей для славянских языков, где синтаксические зависимости часто маркируются морфологическими чертами, нежели определенным порядком словоформ. Поэтому NER сложен для этих языков в сравнении с германскими или романскими языками.

Исходя из вышесказанного можно понять, что квалифицированные разметчики — это дорого, и работают они, обычно, не очень оперативно. Деньги вы потратите точно, но совсем не факт, что получится идеальная разметка, ведь если инструкция сложная, даже квалифицированный человек может ошибиться и что-то неправильно понять. Для борьбы с этим используют многократную разметку одного текста разными людьми, что еще увеличивает цену разметки и время, за которое она готовится. Избежать этого процесса или даже серьезно сократить его не выйдет: чтобы обучаться, нужно иметь качественную обучающую выборку разумных размеров.

5.5 Метрика

Основная метрика для нашей задачи – это строгая f-мера. Это среднее гармоническое точности и полноты, стандартная метрика, где полнотой мы считаем долю выделенных нами сущностей, которые также есть и в эталоне, относительно всех сущностей в эталоне, а точностью – долю выделенных нами сущностей, которые присутствуют в эталоне, относительно всех выделенных нами сущностей.

Пример очень точного, но неполного классификатора – это классификатор, который выделяет в тексте один правильный объект и больше ничего. Пример очень полного, но вообще неточного классификатора – это классификатор, который выделяет сущность на любом отрезке текста (таким образом, помимо всех эталонных сущностей, наш классификатор выделяет огромное количество мусора).

6 Инструменты для решения NER для русского языка

6.1 Natasha

Natasha Решает базовые задачи обработки естественного русского языка: сегментация на токены и предложения, морфологический и синтаксический анализ, лемматизация, извлечение, нормализация именованных сущностей.

Для новостных статей качество на всех задачах сравнимо или превосходит существующие решения. Например с задачей NER Natasha справляется на 1 процентный пункт хуже, чем DeepPavlov BERT NER (F1 PER 0.97, LOC 0.91, ORG 0.85), модель весит в 75 раз меньше (27МБ), работает на CPU в 2 раза быстрее (25 статей/сек), чем BERT NER на GPU.

На вход подается непреработанный (регистры, знаки и т.д. сохранены) текст, а на выходе мы хотим увидеть спаны. Всю работу осуществляем в Colab. Установим Natasha (поддерживает Python версии 3.5 и выше). Соберем бессмысленную новость, где будет много именованных сущностей:

Можем заметить некоторые ошибки: в токене ул. Крузенштерна Крузенштерн выделяется как персона, а токен МПЗ выделился как организация.

Тем не менее, на мой взгляд Natasha сейчас один из главных проектов NLP для русского языка. Он имеет долгую историю, и начинался с rule based решения, которое развивалось через популярный Yargy Parser, и сейчас решает основные задачи NLP для русского языка: токенизацию, сегментация предложения, лемматизация, нормализация фразы, синтаксический разбор, NER-тегирование, извлечение фактов. Проект сделанный людьми говорящими на русском языке для русского языка. Результат, к сожалению, не стабилен, но все же, проект активно развивается, и показывает достойный результат.

```
!pip install natasha

Requirement already satisfied: natasha in /usr/local/lib/python3.7/dist-packages (1.4.0)
Requirement already satisfied: yargy>=0.14.0 in /usr/local/lib/python3.7/dist-packages (from natasha) (0.15.0)
Requirement already satisfied: slovnet>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from natasha) (0.5.0)
Requirement already satisfied: pymorphy2 in /usr/local/lib/python3.7/dist-packages (from natasha) (0.9.1)
Requirement already satisfied: ipymarkup>=0.8.0 in /usr/local/lib/python3.7/dist-packages (from natasha) (0.9.0)
Requirement already satisfied: navec>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from natasha) (0.10.0)
Requirement already satisfied: razdel>=0.5.0 in /usr/local/lib/python3.7/dist-packages (from natasha) (0.5.0)
Requirement already satisfied: intervaltree>=3 in /usr/local/lib/python3.7/dist-packages (from ipymarkup>=0.8.0->natasha) (3.1.0)
Requirement already satisfied: sortedcontainers<3.0,>=2.0 in /usr/local/lib/python3.7/dist-packages (from intervaltree>=3->ipymarkup>=0.8.0->natasha) (2.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from navec>=0.9.0->natasha) (1.21.6)
Requirement already satisfied: pymorphy2-dicts-ruc3.0,>=2.4 in /usr/local/lib/python3.7/dist-packages (from pymorphy2->natasha) (2.4.417127.4579844)
Requirement already satisfied: docopt>=0.6 in /usr/local/lib/python3.7/dist-packages (from pymorphy2->natasha) (0.6.2)
Requirement already satisfied: dawg-python>=0.7.1 in /usr/local/lib/python3.7/dist-packages (from pymorphy2->natasha) (0.7.2)

from natasha import (
    Segmenter,

    NewsEmbedding,
    NewsMorphTagger,
    NewsSyntaxParser,

    Doc
)

segmenter = Segmenter()

emb = NewsEmbedding()
morph_tagger = NewsMorphTagger(emb)
syntax_parser = NewsSyntaxParser(emb)

text = 'Новый герой проекта «Ровесники Вышки» – Вардан Арутюнян, младший научный сотрудник Центра языка и мозга, аспирант Высшей школы экономики...'
doc = Doc(text)

doc.segment(segmenter)

from natasha import NewsNERTagger

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

Один из самых ярких политиков постсоветского пространства, первый президент Украины Леонид Кравчук умер в возрасте 88 лет. Именно он в
LOC— PER—
декабре 1991 года от имени Украины подписал вместе с президентом
LOC—
России Борисом Ельциным и председателем Верховного Совета Белоруссии
LOC— PER— ORG— LOC—
Станиславом Шушкевичем Беловежские соглашения о прекращении
PER—
существования СССР.
LOC—
```

6.2 Stanza

Stanza – это набор мощных и инструментов, выделяющийся простотой в освоении. Наиболее яркой особенностью является так же широкая мультиязычность, что делает предпочтительным использование данного инструмента в проектах, где необходимо автоматизировать анализ текстов на разных языках.

Stanza построена на основе библиотеки глубокого обучения PyTorch. Так же проект включает предварительно обученные модели, соответствующие 66 естественным языкам (английский, испанский, русский и т.д.) Для решения задачи для текста на русском языке можно в коде программы применить метод `download('ru')`, который загрузит указанную языковую модель.

Все сущности выделились корректно.

Можно отметить, что по сравнению с Natasha не выделяются организации, но выделяются прочие именованные сущности (MISC).

```

1 import stanza
2 stanza.download('ru')
3
4 txt = 'Власти Москвы выделили 110 млрд рублей на поддержку населения, системы здравоохранения \
5 и городского хозяйства. Об этом сообщается на сайте мэра столицы https://www.sobyanin.ru/ в пятницу, 1 мая. \
6 По адресу Алтуфьевское шоссе д.51 (основной вид разрешенного использования: производственная деятельность, склады) \
7 размещен МПЗ? Подпоручик Кижэ управля автомобилем BA32107 перевозил автомат АК47 с целью ограбления банка BT524, \
8 как следует из записей. \
9 Взыскать с индивидуального предпринимателя Иванова Костантипа Петровича дата рождения 10 января 1970 года, \
10 проживающего по адресу город Санкт-Петербург, ул. Крузенштерна, дом 5/1А 8 000 (восемь тысяч) рублей 00 копеек \
11 гос. пошлины в пользу бюджета РФ Жители требуют незамедлительной остановки МПЗ и его вывода из района. \
12 Решение было принято по поручению мэра города Сергея Собянина в связи с ограничениями из-за коронавируса.'
13
14 def stanza_nlp_ru(text):
15     nlp = stanza.Pipeline(lang='ru', processors='tokenize,ner')
16     doc = nlp(text)
17     print(*[f'entity: {ent.text}\ttype: {ent.type}' for sent in doc.sentences for ent in sent.ents], sep='\n')
18     stanza_nlp_ru(txt)

```

```

=====
| Processor | Package |
=====
| tokenize | syntagrus |
| ner      | wikiner   |
=====

2022-05-12 14:51:14 INFO: Use device: cpu
2022-05-12 14:51:14 INFO: Loading: tokenize
2022-05-12 14:51:14 INFO: Loading: ner
2022-05-12 14:51:16 INFO: Done loading processors!
entity: Москвы type: LOC
entity: Алтуфьевское шоссе type: LOC
entity: Кижэ type: PER
entity: BA32107 type: MISC
entity: АК47 type: MISC
entity: BT524 type: MISC
entity: Иванова Костантипа Петровича type: PER
entity: Санкт-Петербург type: LOC
entity: ул. Крузенштерна type: LOC
entity: РФ type: LOC
entity: МПЗ type: LOC
entity: Сергея Собянина type: PER

Process finished with exit code 0

```

6.3 Polyglot

Одна из старейших библиотек, быстрая работа и большое количество поддерживаемых языков делают ее по-прежнему популярной. С другой стороны вирусная GPLv3 license не позволяет ее использовать в коммерческой разработке в полную силу.

Результат, откровенно говоря, плохой, но скорость работы и хорошая поддержка позволяют улучшить его, если приложить усилия.

```
[ ] russian_text = 'Власти Москвы выделили 110 млрд рублей на поддержку населения, системы здравоохранения \
и городского хозяйства. Об этом сообщается на сайте мэра столицы https://www.sobyanin.ru/ в пятницу, 1 мая. \
По адресу Алтуфьевское шоссе д.51 (основной вид разрешенного использования: производственная деятельность, склады) \
размещен МПЗ? Подпоручик Кихе управляя автомобилем ВА32107 перевозил автомат АК47 с целью ограбления банка ВТБ24, \
как следует из записей. \
Взыскать с индивидуального предпринимателя Иванова Константина Петровича дата рождения 10 января 1970 года, \
проживающего по адресу город Санкт-Петербург, ул. Круженштерна, дом 5/1А 8 000 (восемь тысяч) рублей 00 копеек \
гос. пошлины в пользу бюджета РФ Жители требуют незамедлительной остановки МПЗ и его вывода из района. \
Решение было принято по поручению мэра города Сергея Собянина в связи с ограничениями из-за коронавируса.'
```

```
!pip3 install -U git+https://github.com/abosamoor/polyglot.git@master
!polyglot download embeddings2.ru ner2.ru
from polyglot.text import Text

!polyglot download embeddings2.ru ner2.ru
for ent in Text(russian_text).entities:
    print(ent[0],ent.tag)
```

```
[polyglot_data] Downloading package embeddings2.ru to
[polyglot_data] /root/polyglot_data...
[polyglot_data] Downloading package ner2.ru to /root/polyglot_data...
ВТБ24 I-ORG
Иванова I-PER
Санкт I-LOC
Круженштерна I-PER
РФ I-ORG
Сергей I-PER
```

6.4 PullEnti

Библиотека C# для NER на русском языке. В 2016 году она заняла первое место на конкурсе factRuEval-2016. В 2018 году автор портировал код на Java и Python.

Наверно, самое симпатичное решение для русского языка.

Быстро, глубоко, со вниманием к деталям. Решение rule based, что естественным образом ограничивает его развитие, однако его автономность, скорость и результаты позволяют надеяться на развитие проекта.

Найдем именованные сущности в уже использованном нами тексте:

```

▶ russian_text = 'Власти Москвы выделили 110 млрд рублей на поддержку населения, системы здравоохранения \
и городского хозяйства. Об этом сообщается на сайте мэра столицы https://www.sobyanin.ru/ в пятницу, 1 мая. \
По адресу Алтуфьевское шоссе д.51 (основной вид разрешенного использования: производственная деятельность, склады) \
размещен МПЗ? Подпоручик Кижее управляя автомобилем ВА32107 перевозил автомат АК47 с целью ограбления банка ВТБ24, \
как следует из записей. \
Взыскать с индивидуального предпринимателя Иванова Константина Петровича дата рождения 10 января 1970 года, \
проживающего по адресу город Санкт-Петербург, ул. Круженштерна, дом 5/1А 8 000 (восемь тысяч) рублей 00 копеек \
гос. пошлины в пользу бюджета РФ Жители требуют незамедлительной остановки МПЗ и его вывода из района. \
Решение было принято по поручению мэра города Сергея Собянина в связи с ограничениями из-за коронавируса.'

[1] !pip install pullenti-wrapper
!pip install graphviz

Collecting pullenti-wrapper
  Downloading pullenti_wrapper-0.9.0-py3-none-any.whl (7.1 kB)
Collecting pullenti-client==0.6.0
  Downloading pullenti_client-0.6.0-py2.py3-none-any.whl (8.2 kB)
Collecting pullenti==3.23
  Downloading pullenti-3.23-py3-none-any.whl (15.3 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.3 MB 7.0 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from pullenti-client==0.6.0->pullenti-wrapper) (2.23.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->pullenti-client==0.6.0->pullenti-wrapper) (3.0.4)
Requirement already satisfied: certifi<2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->pullenti-client==0.6.0->pullenti-wrapper) (2021.10.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->pullenti-client==0.6.0->pullenti-wrapper) (2.10)
Requirement already satisfied: urllib3<1.25.0,!<1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->pullenti-client==0.6.0->pullenti-wrapper) (1.24.3)
Installing collected packages: pullenti-client, pullenti, pullenti-wrapper
Successfully installed pullenti-3.23 pullenti-client-0.6.0 pullenti-wrapper-0.9.0
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (0.10.1)

▶ from pullenti_wrapper.processor import Processor,
    MONEY,
    URI,
    PHONE,
    DATE,
    KEYWORD,
    DEFINITION,
    DENOMINATION,
    MEASURE,
    BANK,
    GEO,
    ADDRESS,
    ORGANIZATION,
    PERSON,
    MAIL,
    TRANSPORT,
    DECREE,
    INSTRUMENT,
    TITLEPAGE,
    BOOKLINK,
    BUSINESS,
    NAMEIDENTITY,
    WEAPON,

processor = Processor([PERSON, ORGANIZATION, GEO, DATE, MONEY])
text = russian_text
result = processor(text)
result.graph
```

Результат:

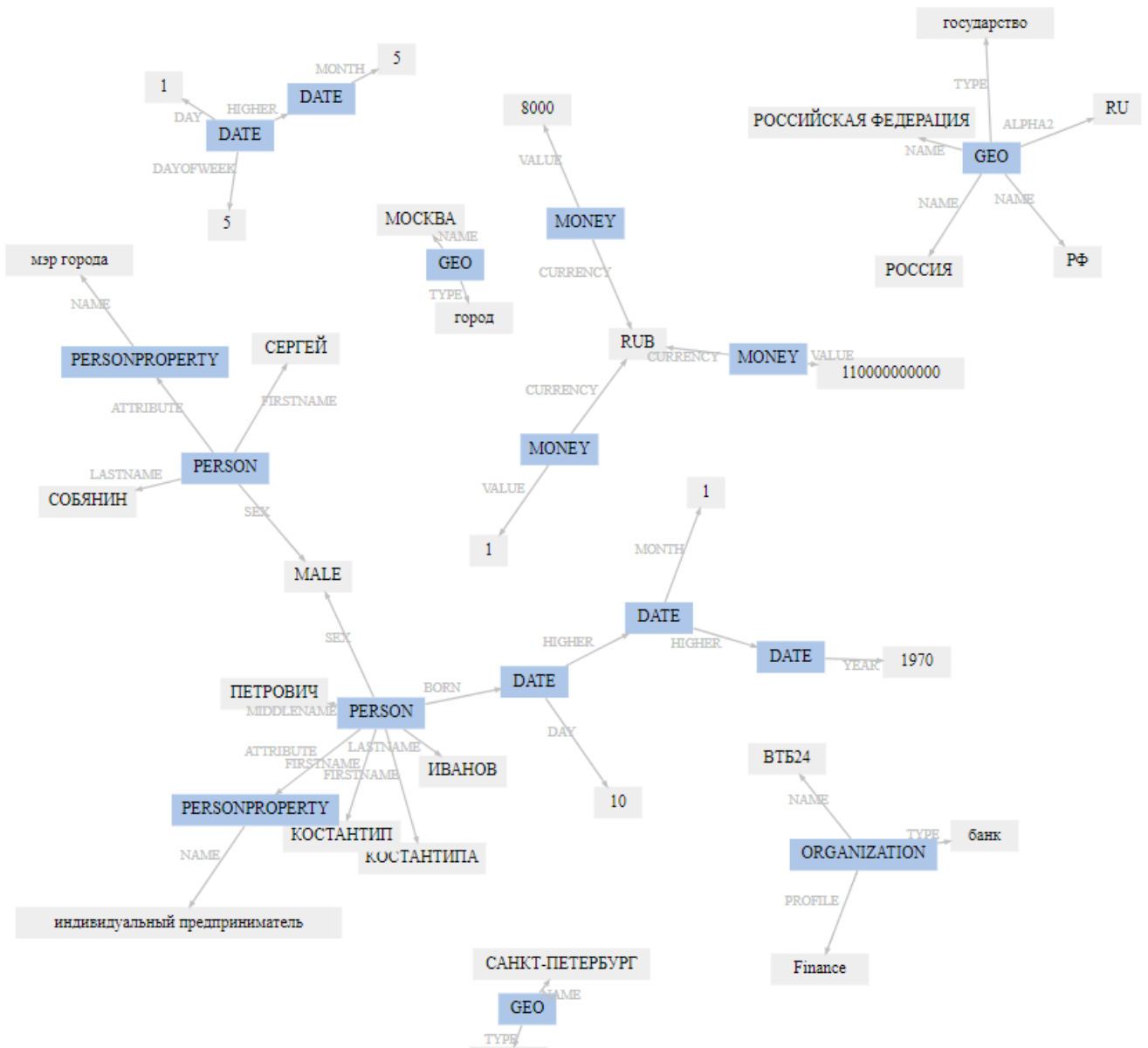
Result(text='Власти Москвы выделили 110 млрд рублей на поддержку населения системы здравоохранения и городского хозяйства. Об этом сообщается на сайте мэра столицы <https://www.sobyanin.ru/> в пятницу, 1 мая. По адресу Алтуфьевское шоссе д.51 (основной вид разрешенного использования: производственная деятельность, склады) размещен МПЗ? Подпоручик Кижее управляя автомобилем ВА32107 перевозил автомат АК47 с целью ограбления банка ВТБ24, как следует из записей. Взыскать с индивидуального предпринимателя Иванова Константина Петровича дата рождения 10 января 1970 года, проживающего по адресу город Санкт-Петербург, ул. Круженштерна, дом 5/1А 8 000 (восемь тысяч) рублей 00 копеек гос. пошлины в пользу бюджета РФ Жители требуют незамедлительной остановки МПЗ и его вывода из района. Решение было принято по поручению мэра города Сергея Собянина в связи с ограничениями из-за коронавируса.', matches=[Match(referent=GeoReferent(label='GEO', slots=[Slot(key='NAME', value='МОСКВА'), Slot(key='TYPE', value='город')]), span=Span(start=7, stop=13), children=[]), Match(referent=MoneyReferent(label='MONEY', slots=[Slot(key='VALUE', value='110000000000')]), span=Span(start=2, stop=38), children=[]), Match(referent=DateReferent(label='DATE', slots=[Slot(key='DATE', value='1'), Slot(key='DAYOFWEEK', value='5'), Slot(key='HIGHER', value=DateReferent.HIGHER)])], result.graph)

```

slots=[Slot(key='MONTH', value='5'))]])), span=Span(start=179, stop=193),
children=[]), Match(referent=OrganizationReferent(label='ORGANIZATION',
slots=[Slot(key='TYPE', value='банк'), Slot(key='PROFILE', value='Finance'
Slot(key='NAME', value='БТБ24'))]), span=Span(start=411, stop=422), childre
Match(referent=PersonReferent(label='PERSON', slots=[Slot(key='SEX', value
Slot(key='LASTNAME', value='ИВАНОВ'), Slot(key='FIRSTNAME', value='КОСТАНТ
Slot(key='FIRSTNAME', value='КОСТАНТИПА'), Slot(key='MIDDLENAME', value='И
Slot(key='ATTRIBUTE', value=PersonPropertyReferent(label='PERSONPROPERTY',
slots=[Slot(key='NAME', value='индивидуальный предприниматель'))]), Slot(k
value=DateReferent(label='DATE', slots=[Slot(key='DAY', value='10'), Slot(
value=DateReferent(label='DATE', slots=[Slot(key='MONTH', value='1'),
Slot(key='HIGHER', value=DateReferent(label='DATE', slots=[Slot(key='YEAR'
value='1970'))]]))]]))]])), span=Span(start=459, stop=553), children=[Match(
slots=[Slot(key='NAME', value='индивидуальный предприниматель'))]), span=Sp
stop=490), children=[]), Match(referent=DateReferent(label='DATE', slots=[
value='10'), Slot(key='HIGHER', value=DateReferent(label='DATE', slots=[Sl
value='1'), Slot(key='HIGHER', value=DateReferent(label='DATE', slots=[Slo
value='1970'))]]))]])), span=Span(start=534, stop=553), children=[])),
Match(referent=GeoReferent(label='GEO', slots=[Slot(key='NAME', value='САН
Slot(key='TYPE', value='город'))]), span=Span(start=578, stop=599), childre
Match(referent=MoneyReferent(label='MONEY', slots=[Slot(key='CURRENCY',
value='RUB'), Slot(key='VALUE', value='1')]), span=Span(start=625, stop=62
children=[]), Match(referent=MoneyReferent(label='MONEY', slots=[Slot(key=
value='RUB'), Slot(key='VALUE', value='8000')]), span=Span(start=628,
stop=665), children=[]), Match(referent=GeoReferent(label='GEO', slots=[Sl
value='RU'), Slot(key='NAME', value='РФ'), Slot(key='NAME', value='РОССИЙС
ФЕДЕРАЦИЯ'), Slot(key='NAME', value='РОССИЯ'), Slot(key='TYPE', value='рос
span=Span(start=696, stop=698), children=[]), Match(referent=PersonReferen
slots=[Slot(key='SEX', value='MALE'), Slot(key='LASTNAME', value='СОБЯНИН'
Slot(key='FIRSTNAME', value='СЕРГЕЙ'), Slot(key='ATTRIBUTE', value=PersonP
slots=[Slot(key='NAME', value='мэр города'))]])), span=Span(start=803,
stop=830), children=[Match(referent=PersonPropertyReferent(label='PERSONNPR
slots=[Slot(key='NAME', value='мэр города'))]), span=Span(start=803, stop=8
children=[]))]]))

```

Решение можно оформить более симпатично с помощью graphviz:



Результат положительный, самый детализированный. Pullenty умеет определять вид деятельности сущности (PersonProperty), да и вообще разделяет имя, фамилию, даже пол. Например Сергей собянин (PER), но при этом label='PERSONPROPERTY', slots=[Slot(key='NAME', value='мэр города')].

6.5 Вывод

Мы рассмотрели базовые возможности открытых библиотек, которые быстро и просто помогут решить NER задачи на русском языке. Как говорилось ранее, на мой взгляд «из коробки» лучше всех показал себя Pullenty. Хуже всех показала себя библиотека Polyglot. Важно отметить, что любую библиотеку можно дообучить при необходимости для конкретной задачи.

Pullenti имеет более лучшие показатели в задаче распознавания при сравнении с конкурентами. Одним из плюсов является то, что в тексте осуществляется поиск адреса и привязка к объектам из ГАР ФИАС. То есть если адрес распознан, то значит он точно существует. Natasha зависима от расположения ключевых слов и чувствительна к расположению ключевых слов и их сокращений, не всегда видит названия, состоящие из нескольких слов. Но т.к. Natasha использует словари и правила для распознавания сущностей, то их можно обогатить новыми данными, что позволит расширить функциональные возможности библиотеки, подстроив ее под конкретные задачи.

Stanza использует предварительно обученные и настроенные модели и языки, которые для использования требуется предварительно загрузить, что не всегда бывает удобно (для русского языка она требовала 524мб памяти). Нужно отметить небольшое количество тэгов (2 + MISC), но стоит заметить, что Stanza проста в использовании: библиотеку достаточно импортировать, загрузить модель, вызвать функцию, а на выходе получить объект, из которого после выполнения необходимо извлечь нужные нам атрибуты.

Список литературы

- [1] Плавное введение в Natural Language Processing (NLP)
- [2] Understand NLP Model Building Approach with Python
- [3] Natural Language Processing With Python's NLTK Package
- [4] Basic Concepts of Natural Language Processing (NLP) Models and Python Implementation
- [5] Морфологический анализатор pymorphy2
- [6] pymorphy2
- [7] Morphological Analyzer
- [8] Руководство
- [9] Лемматизируй это быстрее (PyMorphy2, PyMystem3 и немного магии)
- [10] Контекстная лемматизация Python
- [11] Предобработка текста в NLP