**ДИПЛОМНАЯ РАБОТА**

Построение семантической сети

по учебной дисциплине: «Организация ЭВМ и вычислительных систем»

Выполнил

Воронцов В.А. \_\_\_\_\_\_\_\_\_

(подпись)

Руководитель

Жданов А.Г. \_\_\_\_\_\_\_\_\_

(подпись)

Рекомендуемая оценка \_\_\_\_\_\_\_\_\_

Оценка работы \_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

МОСКВА 2023

Оглавление

[1 ГЛАВА. 5](#_Toc15294)

[1.1 Подготовка обзора современных программных средств и алгоритмов анализа текстов на естественном языке. 5](#_Toc8401)

[1.1.1 Комплекс моделей для обработки текстов на естественном языке Natasha 5](#_Toc2594)

[1.1.2 Комплекс моделей для обработки текстов на естественном языке Deeppavlov 6](#_Toc2049)

[1.1.3 Программное средство для работы с датасетами Corus 7](#_Toc5792)

[1.1.4 Библиотека для выделения именованных сущностей Pullenti 7](#_Toc28929)

[1.2 Реализация архитектуры модуля выделения именованных сущностей с помощью программного средства Pullenti 9](#_Toc12882)

[2 ГЛАВА. 12](#_Toc4981)

[2.1 Введение 12](#_Toc32119)

[2.2 Программная реализация модуля выделения именованных сущностей с помощью программного средства Pullenti 12](#_Toc24867)

[2.2.1 Выявление именованных сущностей и отношений между ними с помощью Pullenti 12](#_Toc13111)

[2.2.2 Анализ работы библиотеки Pullenti 14](#_Toc24355)

[2.2.3 Форматирование форматирование полученных данных данных 14](#_Toc26771)

[2.3 Реализация архитектуры модуля выделения именованных сущностей с помощью ИНС DeepPavlov 15](#_Toc2456)

[2.4 Программная реализация модуля выделения именованных сущностей с помощью ИНС DeepPavlov 17](#_Toc32705)

[2.5 Подготовка выборки данных для тестирования выделения именованных сущностей и связей между ними 24](#_Toc7691)

[2.5.1 Выгрузка датасета 24](#_Toc23434)

[2.6 Разработка и программная реализация способа графового представления именованных сущностей и связей между ними 25](#_Toc18924)

[2.6.1 Постороенние графа 25](#_Toc31322)

[2.6.2 Демонстрация графа в neo4j 25](#_Toc26048)

[2.7 Сохранение результата графового представления именованных сущностей в графовую БД 26](#_Toc7980)

[2.7.1 Тестирование работоспособности на запросах пользователя 26](#_Toc27390)

[Заключение 27](#_Toc7798)

[В ходе работы по построению семантической сети было выполнен ряд задач, перечисленных ниже. 27](#_Toc435)

[Список литературы 28](#_Toc22457)

[ПРИЛОЖЕНИЕ 1 Перечень тегов именнованых сущностей библиотеки Pullenti 29](#_Toc11361)

Введение

Семанти́ческая сеть — [информационная модель](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) [предметной области](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C" \o "Предметная область), имеет вид ориентированного [графа](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Граф (математика)). Вершины графа соответствуют объектам предметной области, а дуги (рёбра) задают отношения между ними.

Natural Language Processing – обработка естественного языка – подраздел информатики и AI, посвященный тому, как компьютеры анализируют естественные (человеческие) языки. NLP позволяет применять алгоритмы машинного обучения для текста и речи.

Целью работы является построение семантической сети, где вершинами графа будут именованные сущности, а ребрами отношения, связвывающие их. Данная сеть позволит получить информацию о фактах, локациях, организациях, личностях и т.д., а также о связях между ними. Для достижения цели работы необходимо решить следующие практические задачи:

1. Выявление именнованых сущностей
2. Выявления отношений между ними
3. Посторение и отображение семантической сети в виде графа

# ГЛАВА.

## Подготовка обзора современных программных средств и алгоритмов анализа текстов на естественном языке.

### Комплекс моделей для обработки текстов на естественном языке Natasha

**[Natasha](https://github.com/natasha/natasha)** решает базовые задачи обработки естественного русского языка: сегментация на токены и предложения, морфологический и синтаксический анализ, лемматизация, извлечение, нормализация именованных сущностей. Она объединяет несколько библиотек под своим началом

**[Yargy](https://github.com/natasha/yargy)** - Аналог яндексового [Томита-парсера](https://yandex.ru/dev/tomita). Использует словари и правила, извлекает структурированную информацию из текстов на естественном русском языке.

**[Razdel](https://github.com/natasha/razdel)** - Делит текст на слова и предложения.

**[Corus](https://github.com/natasha/corus)** - Коллекция ссылок на публичные русскоязычные датасеты.

**[Navec](https://github.com/natasha/navec)** - Набор компактных предобученных эмбеддингов для русского языка.

**[Slovnet](https://github.com/natasha/slovnet)** - Компактные модели для обработки естественного русского языка: морфологический теггер, синтаксический парсер, NER-теггер.

**[Ipymarkup](https://github.com/natasha/ipymarkup)** - Инструменты для визуализации NER-разметки и синтаксических связей.

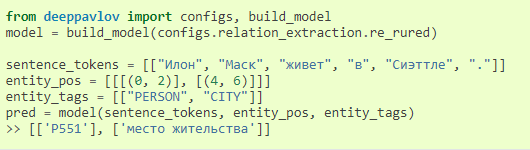
**[Nerus](https://github.com/natasha/nerus)** - Большой корпус с автоматической разметкой именованных сущностей, морфологии и синтаксиса.

### Комплекс моделей для обработки текстов на естественном языке Deeppavlov

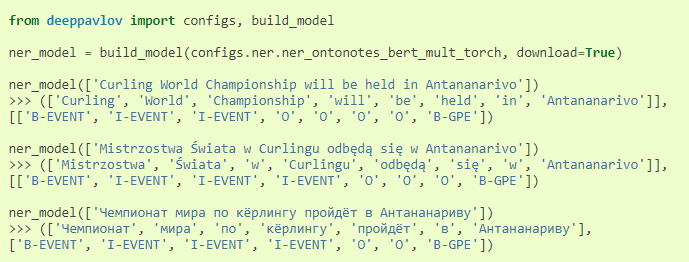
**DeepPavlov** - это библиотека разговорного ИИ с открытым исходным кодом, построенная на TensorFlow, Keras и PyTorch. DeepPavlov предназначен для разработки готовых чат-ботов и сложных разговорных систем, исследования в области НЛП и, в частности, диалоговых систем.

Основные возможности библиотеки в рамках задач куросовой работы

[Извлечение отношений() - модель для работы с Русским языком обученная дампе Lenta.ru](https://github.com/deepmipt/DeepPavlov/blob/0.17.1/deeppavlov/configs/relation_extraction/re_rured.json)



Распознование именнованых сущностей(Named Entity Recognition) - Существует три основных типа доступных моделей: стандартная модель на основе RNN, модель на основе BERT (на TensorFlow и PyTorch) и гибридная модель.



### Программное средство для работы с датасетами Corus

[Библиотека](https://github.com/natasha/corus) **[Corus](https://github.com/natasha/corus)** — часть проекта Natasha, коллекция ссылок на публичные русскоязычные NLP-датасеты + Python-пакет с функциями-загрузчиками.



### Библиотека для выделения именованных сущностей [Pullenti](https://github.com/pullenti)

**[PullEnti](http://pullenti.ru/)** — библиотека на C# для NER на русском языке. В 2016 году заняла первое место на [соревновании factRuEval-2016](https://github.com/dialogue-evaluation/factRuEval-2016). В 2018 автор [портировал код на Java и Python](https://habr.com/post/354942/).

Достоинства PullEnti:

* Ряд встроенных сущностей. Не только базовые PERSON, ORG, GEO, но ещё DATE, MONEY, PHONE, ADDRESS и ~10 других.
* Подробный разбор. Например, для фразы "руководителю Гринписа Ивану Залесову" пользователь получит должность, организацию, отдельно имя и фамилию, всё в нормальной форме.
* Кроме русского есть английский, украинский языки.
* Не зависит от других библиотек, не нужны большие модели, словари для использования.

Недостатки PullEnti:

* Некоммерческое использование.
* Медленно работает в Python. Инициализируется ~20 секунд, текст одной новости с lenta.ru разбирается ~2 секунды.
* Не очень привычный для Python интерфейс. Код получается трансляцией из C#.

Удалось связаться в telegram c одним из разработчиков, он пояснил как происходит выявление именованных сущностей в PullEnti :

Как таковых известных правил нет (типа "алгоритм Ахо-Корасик"). Правила представляют собой по сути большое число разных if-then-else. На практике оказался эффективным следующий подход, применяемый практически ко всем типам сущностей: на базе исходных токенов сначала выделяются подпоследовательности специализированных метатокентов, а уже на их основе - сущности. Например, для персон такими спец. метатокенами выступают: 1) инициал 2) слово с большой буквы (для него вычисляется доп. информация, может ли это быть фамилией, именем или отчеством. Причём здесь не только отдельные слова, но и могут быть несколько слов (например, "де ля Фер", "Петров-Водкин", "Махмуд оглы"). И уже имея последовательность таких элементов, можно проверять разные гипотезы типа "Фамилия Имя Отчество", "Имя Фамилия", "Фамилия Инициал" и т.п. Для каждой такой гипотезы вычисляется коэффициент на основе кучи признаков, и если он превысил пороговое значение, то это считается сущность. Аналогичный подход, например, для населенных пунктов, когда метатокеном может быть ключевое слово ("город", "посёлок", ...), известный крупный город из встроенного словаря/отнологии или просто слово с большой буквы. И тут также проверяются гипотезы. Так практически для всех сущностей.

## Реализация архитектуры модуля выделения именованных сущностей с помощью программного средства Pullenti

1. Программа разработанная на языке python
   1. Модуль pullenti\_client для python

Клиент для [PullentiServer](https://github.com/pullenti/PullentiServer). Предоставляет удобное Python API для результатов работы сервера.

* 1. Модуль ipymarkup для python
  2. Модуль neo4j для python
  3. Функции и процедуры форматирования полученных в результате работы вышеуказанных модулей

1. Сервер в виде Docker-контейнера

Простая обёртка HTTP-сервер для библиотеки [PullEnti для .NET Core 2.0](https://github.com/pullenti/PullentiNetCore). Запросы обрабатываются последовательно, нет ограничений на время обработки, число запросов, объём текста в одном запросе. Предполагается, что сервер используется локально в связке с [pullenti-client](https://github.com/pullenti/pullenti-client).

Сервер распространяется в виде Docker-контейнера, настраивается XML-конфигурацией.

custom.xml:

<?xml version="1.0" encoding="utf-8" ?>

<conf>

<listen>

<host>\*</host>

<port>8080</port>

</listen>

<langs>

<lang>ru</lang>

<lang>en</lang>

</langs>

<analyzers>

<analyzer>geo</analyzer>

<analyzer>org</analyzer>

<analyzer>person</analyzer>

</analyzers>

</conf>

Cписок доступных langs: ru, ua, by, en, it, kz.

Список доступных analyzers: money, uri, phone, date, keyword, definition, denomination, measure, bank, geo, address, org, person, mail, transport, decree, instrument, titlepage, booklink, business, named, weapon.

Предполагается, что на порту 8080 работает PullentiServer. Пример команды для запуска:

docker run -it --rm -p 8080:8080 pullenti/pullenti-server

Как указать список сущностей и языков написано в репозитории [PullentiServer](https://github.com/pullenti/PullentiServer).

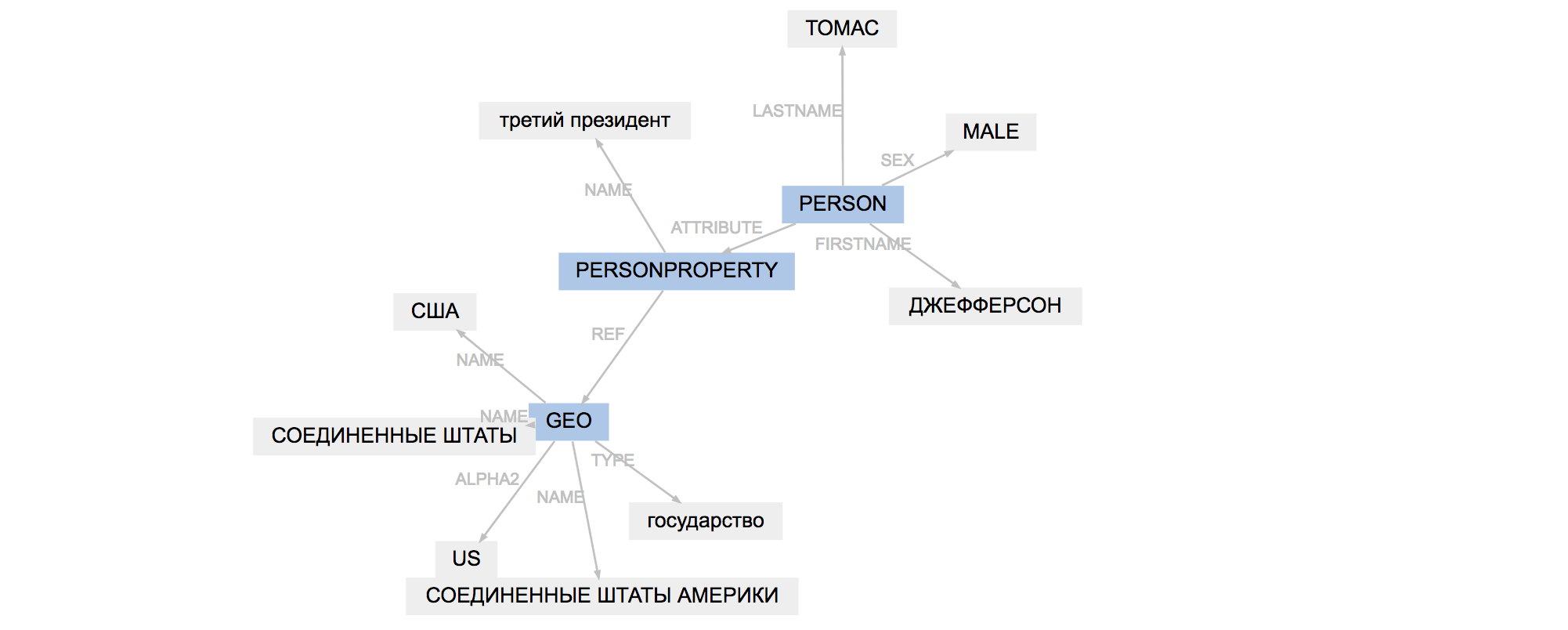
from pullenti\_client import Client

client = Client('localhost', 8080)

text = 'В США прядь волос третьего президента Соединенных Штатов Томаса Джефферсона продали на аукционе в Техасе за 6,9 тысячи долларов, передает Life. Локоны бывшего лидера США ушли с молотка почти через 190 лет после его смерти. Их покупатель пожелал остаться неизвестным. Перед началом аукциона волосы Джефферсона оценивали в 3 тысячи долларов. В январе 2015 года прядь волос 16-го президента США Авраама Линкольна продали за 25 тысяч долларов на аукционе в Далласе. Выставленную на аукцион прядь срезал начальник медицинской службы Армии США Джозеф Барнс после того, как Линкольн был застрелен 14 апреля 1865 года Джоном Бутом. Томас Джефферсон был автором Декларации независимости США и третьим президентом страны (1801-1809 годы). Авраам Линкольн — первый президент от Республиканской партии США в 1861-1865 годах, национальный герой США.'

result = client(text)

display(result)



# ГЛАВА.

## Введение

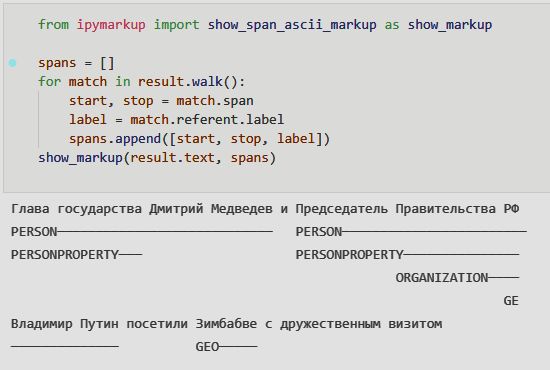
Во второй главе приведены результаты экспериментов по выявлению именнованых сущностей и отношений между ними с использованием выбранных выше библиотек и программных средств.

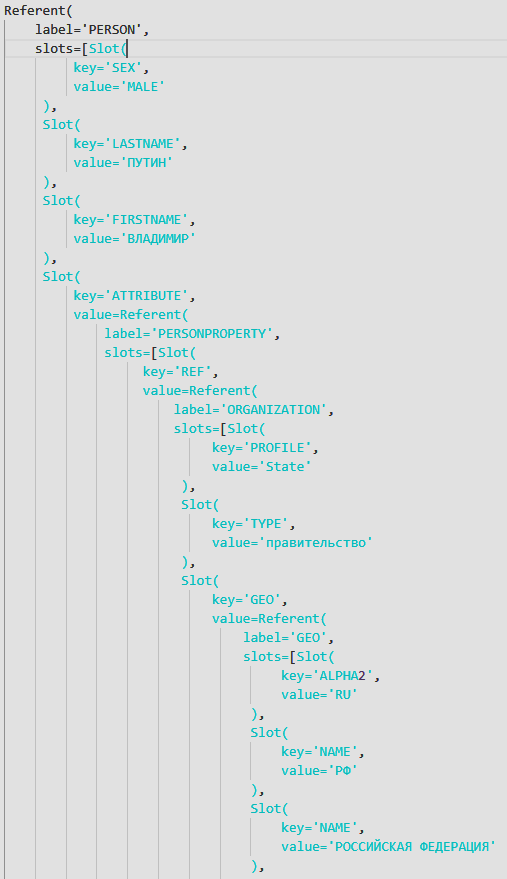
Для работы с датасетами использованa [библиотека Corus](https://github.com/natasha/corus).

В качестве датасета был выбран дамп Lenta.ru.

## Программная реализация модуля выделения именованных сущностей с помощью программного средства Pullenti

### Выявление именованных сущностей и отношений между ними с помощью [Pullenti](https://github.com/pullenti)



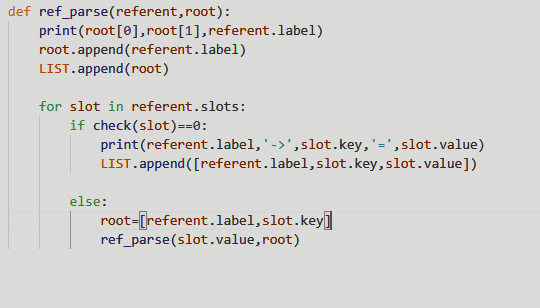


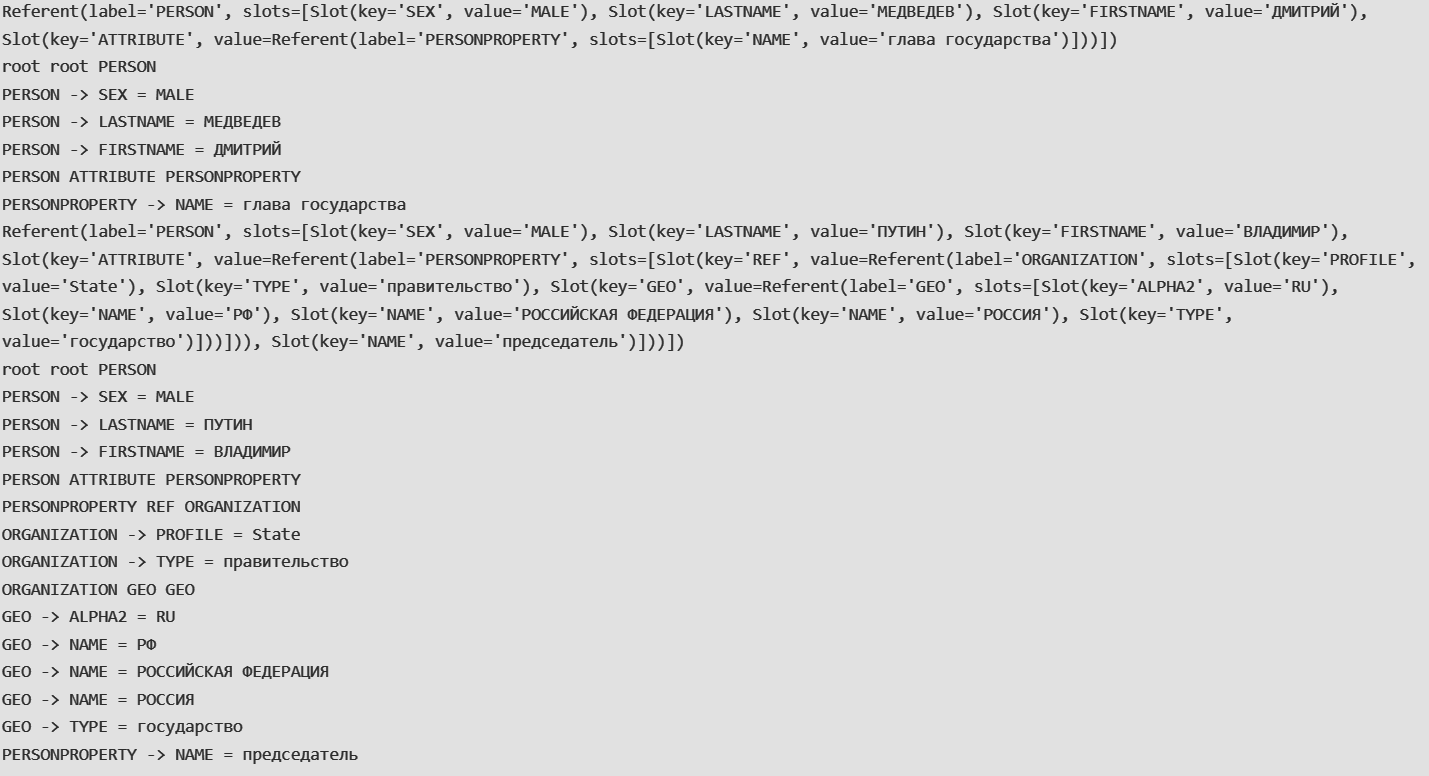
### Анализ работы библиотеки Pullenti

На примере данной фразы продемонстрировано выделение сущностей типа PERSON (....), PERSONPROPERTY (..), GEO() и OGRANIZATION().

PERSON имеет ключи SEX, LASTNAME, FIRSTNAME и значения MALE, Путин, Владимир соответственно.Аналогично для PERSONPROPERTY, GEO и OGRANIZATION. Такой выходной результат позволяет строить и заполнять семантическую сеть в виде графа.

### Форматирование форматирование полученных данных данных





## Реализация архитектуры модуля выделения именованных сущностей с помощью ИНС DeepPavlov

DeepPavlov - это библиотека с открытым исходным кодом основанная на TensorFlow, Keras и PyTorch.

DeepPavlov содержит в себе ряд модулей основанных на обученных моделях. В рамках курсовой работы будут изучены:

* entity linking
* relation extractor
* named entity recognition
* Question answering

Ответ на вопрос по набору данных SQuAD — это задача найти ответ на вопрос в заданном контексте (например, абзац из Википедии), где ответ на каждый вопрос является сегментом контекста:

В DeepPavlov есть две модели для этой задачи: на основе BERT и R-Net. Обе модели предсказывают начальную и конечную позицию ответа в заданном контексте

Список доступных моделей SQuAD:

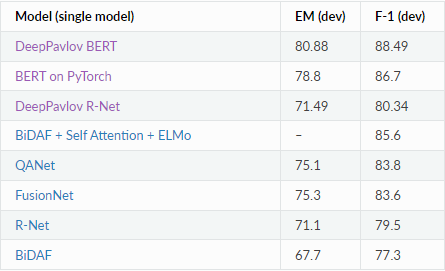


Рисунок – Список моделей SQuAD

Список доступных моделей SDSJ

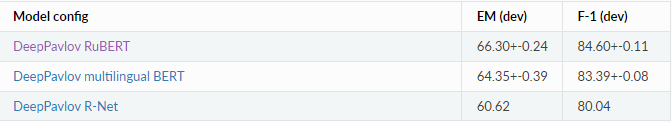


Рисунок – Список моделей SDSJ

Для named entity recognition существует три основных типа моделей: стандартная модель на основе RNN, модель на основе BERT (на TensorFlow и PyTorch) и гибридная модель.

Список доступных моделей:

* [ner\_rus\_bert\_torch](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_rus_bert_torch.json)
* [ner\_collection3\_m1](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_collection3_m1.json)
* [ner\_rus](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_rus.json)
* [ner\_ontonotes\_bert\_torch](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_ontonotes_bert_torch.json)
* [ner\_ontonotes\_m1](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_ontonotes_m1.json)
* [ner\_ontonotes](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_ontonotes.json)
* [ner\_conll2003\_bert](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_conll2003_bert.json)
* [ner\_conll2003\_torch\_bert](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_conll2003_torch_bert.json)
* [ner\_conll2003](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_conll2003.json)
* [conll2003\_m1](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/conll2003_m1.json)
* [ner\_dstc2](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/ner_dstc2.json)
* [vlsp2016\_full](https://github.com/deepmipt/DeepPavlov/blob/0.17.4/deeppavlov/configs/ner/vlsp2016_full.json)

Пример использования:

from deeppavlov import configs, build\_model

ner\_model = build\_model(configs.ner.ner\_ontonotes\_bert\_mult\_torch, download=True)

ner\_model(['Чемпионат мира по кёрлингу пройдёт в Антананариву'])

>>> (['Чемпионат', 'мира', 'по', 'кёрлингу', 'пройдёт', 'в', 'Антананариву'],

['B-EVENT', 'I-EVENT', 'I-EVENT', 'I-EVENT', 'O', 'O', 'B-GPE'])

## Программная реализация модуля выделения именованных сущностей с помощью ИНС DeepPavlov

DeepPavlov располагает возможностью работы через Docker

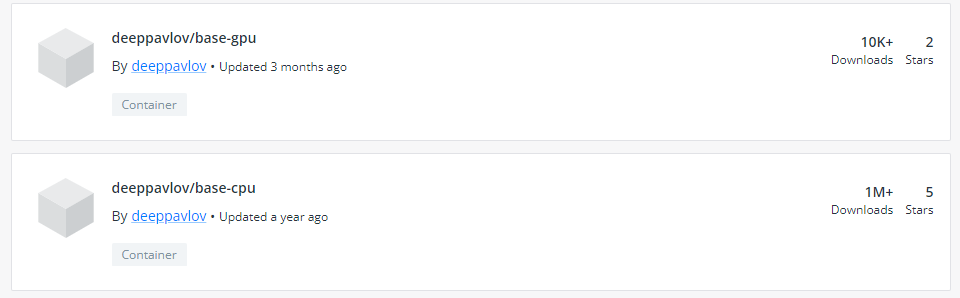


Рисунок 4 – DeepPavlov на DockerHub

Был выбран способ работы основанный на GPU т.к. это быстрее и работа через CPU на практике данной работы оказалась менее стабильна

Листинг №1 Запуск GPU Based:

docker run -e CONFIG=<dp\_config\_name> -p <your\_port>:5000 \

--runtime=nvidia \

-v <dp\_components\_volume>:/root/.deeppavlov \

-v <host\_venv\_dir>:/venv \

deeppavlov/base-gpu:<dp\_version>

Переменная CONFIG позволяет запустить контейнер только с одной определенной предобученой моделью, но не более.

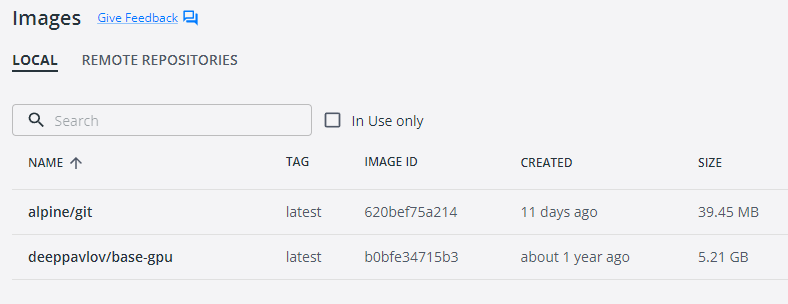


Рисунок 5 – DP Base gpu Image

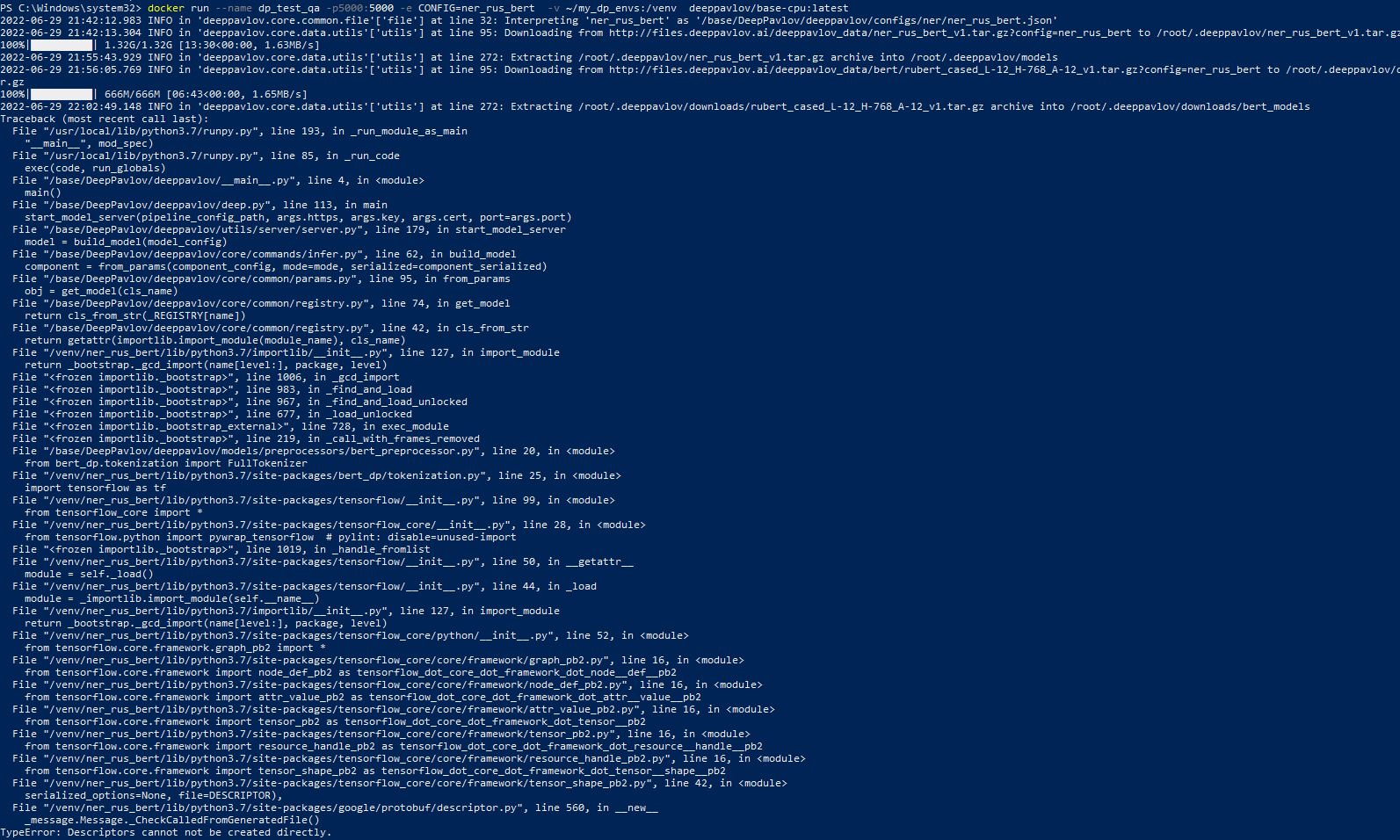


Рисунок 6 – DP сpu Image

Для работы с довольно объемными моделями DeepPavlov , было принято решение выбрать среду разработки GoogleColab, благодаря ресурсам данной среды и изолированности системы данный способ работы не уступает работе с Docker.

С использованием моделей ner\_ontonotes\_bert\_mult\_torch и relation\_extraction.re\_rured, а так же corus для работы с датасетом новостей, было разработано программное средство для поиска исенновынай сущностей и извлечения отношений между ними.

Листинг №1:

!pip install deeppavlov

!pip install transformers

!pip install corus

 !wget https://github.com/yutkin/Lenta.Ru-News-Dataset/releases/download/v1.0/lenta-ru-news.csv.gz

from deeppavlov import configs, build\_model

ner\_model = build\_model(configs.ner.ner\_ontonotes\_bert\_mult\_torch, download=True)

from deeppavlov import configs, build\_model

model = build\_model(configs.relation\_extraction.re\_rured, download=True)

from corus import load\_lenta

path = 'lenta-ru-news.csv.gz'

records = load\_lenta(path)

next(records)

text\_lenta=next(records).text

print(text\_lenta)

from typing import Counter

list\_ner=ner\_model([text\_lenta])

lenta\_list=list\_ner[0][0] #########

counter=0

list\_of\_ent=[]

list\_of\_pos=[]

for i in list\_ner[1][0]:

  if i!='O':

    list\_of\_ent.append(i)

    list\_of\_pos.append(counter)

  counter=counter+1

print(list\_of\_ent)

print(list\_of\_pos)

counter=1

pos=[]

tag=[]

for j in list\_of\_ent:

  temp\_list\_pos=[]     #Подсписок для сохранения позиции найденых именованных сущностей

  temp\_list\_ent=[]   # Подсписок для сохранения тегов найденых именованных сущностей

  if (counter==len(list\_of\_ent[:-1])):

    break

  if (list\_of\_ent[counter-1][0] == "I") and (list\_of\_ent[counter][0] == "B"):

    if (list\_of\_ent[counter+1][0] == "I")and(counter<len(list\_of\_ent[:-1])):

      print(list\_of\_ent[counter-2],list\_of\_ent[counter-1],list\_of\_ent[counter],list\_of\_ent[counter+1])

      print(list\_of\_pos[counter-2],list\_of\_pos[counter-1],list\_of\_pos[counter],list\_of\_pos[counter+1])

      temp\_list\_pos.append(list\_of\_pos[counter-2])

      temp\_list\_pos.append(list\_of\_pos[counter-1])

      temp\_list\_pos.append(list\_of\_pos[counter])

      temp\_list\_pos.append(list\_of\_pos[counter+1])

      ###########################################

      temp\_list\_ent.append(list\_of\_ent[counter-2])

      temp\_list\_ent.append(list\_of\_ent[counter-1])

      temp\_list\_ent.append(list\_of\_ent[counter])

      temp\_list\_ent.append(list\_of\_ent[counter+1])

    else:

        print(list\_of\_ent[counter-2],list\_of\_ent[counter-1],list\_of\_ent[counter])

        print(list\_of\_pos[counter-2],list\_of\_pos[counter-1],list\_of\_pos[counter])

        temp\_list\_pos.append(list\_of\_pos[counter-2])

        temp\_list\_pos.append(list\_of\_pos[counter-1])

        temp\_list\_pos.append(list\_of\_pos[counter])

        #########################################

        temp\_list\_ent.append(list\_of\_ent[counter-2])

        temp\_list\_ent.append(list\_of\_ent[counter-1])

        temp\_list\_ent.append(list\_of\_ent[counter])

else:

      print(list\_of\_ent[counter-2],list\_of\_ent[counter-1],list\_of\_ent[counter],list\_of\_ent[counter+1])

      print(list\_of\_pos[counter-2],list\_of\_pos[counter-1],list\_of\_pos[counter],list\_of\_pos[counter+1])

      temp\_list\_pos.append(list\_of\_pos[counter-2])

      temp\_list\_pos.append(list\_of\_pos[counter-1])

      temp\_list\_pos.append(list\_of\_pos[counter])

      temp\_list\_pos.append(list\_of\_pos[counter+1])

      ###########################################

      temp\_list\_ent.append(list\_of\_ent[counter-2])

      temp\_list\_ent.append(list\_of\_ent[counter-1])

      temp\_list\_ent.append(list\_of\_ent[counter])

      temp\_list\_ent.append(list\_of\_ent[counter+1])

  counter=(counter+1)

  if len(temp\_list\_pos):

    pos.append(temp\_list\_pos)

  if len(temp\_list\_ent):

    tag.append(temp\_list\_ent)

print(pos)

print(tag)

re\_pos\_list=[]

for pos\_ in pos:

  if len(pos\_) == 4:

    re\_pos\_0=(pos\_[0],pos\_[1])

    re\_pos\_1=(pos\_[2],pos\_[3])

    re\_pos\_temp=[[],[]]

    re\_pos\_temp[0]=[re\_pos\_0]

    re\_pos\_temp[1]=[re\_pos\_1]

    re\_pos\_temp=[re\_pos\_temp]

    print(re\_pos\_temp)

    re\_pos\_list.append(re\_pos\_temp)

  else:

    re\_pos\_0=(pos\_[0],pos\_[1])

    re\_pos\_1=(pos\_[2]-1,pos\_[2]+1)

    re\_pos\_temp=[[],[]]

    re\_pos\_temp[0]=[re\_pos\_0]

    re\_pos\_temp[1]=[re\_pos\_1]

    re\_pos\_temp=[re\_pos\_temp]

    print(re\_pos\_temp)

    re\_pos\_list.append(re\_pos\_temp)

print(re\_pos\_list)

def tag\_fix(tag):

  if tag == 'ORG':

    tag='ORGANIZATION'

    return tag

  else:

    return tag

re\_tag\_list=[]

for tag\_ in tag:

  tag\_1=tag\_fix(tag\_[0][2:])

  tag\_2=tag\_fix(tag\_[2][2:])

  tag\_temp=[[tag\_1,tag\_2]]

  print(tag\_temp)

  re\_tag\_list.append(tag\_temp)

print(re\_tag\_list)

counter=0

for tag in re\_tag\_list:

  sentence\_tokens = [lenta\_list]

  entity\_pos = re\_pos\_list[counter]

  entity\_tags = tag

  pred = model(sentence\_tokens, entity\_pos, entity\_tags)

  print(entity\_tags)

  print(pred)

  counter=counter+1

Примеры двух сущностей и отношению между ними:

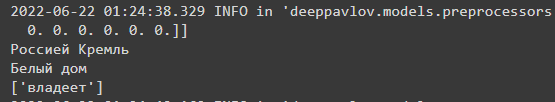


Рисунок 7 – Пример найденного отношения

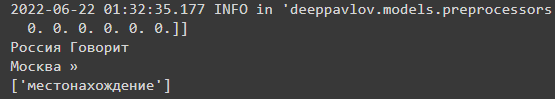


Рисунок 8 – Пример найденного отношения

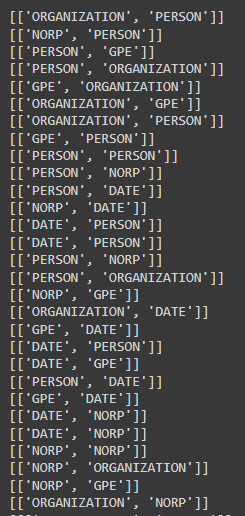


Рисунок 9 – Теги пар сущностей

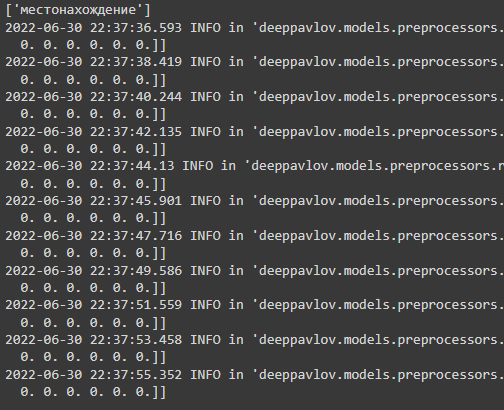


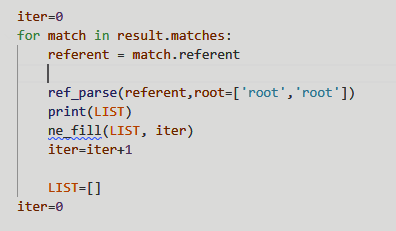
Рисунок 10 – Вывод программы

## Подготовка выборки данных для тестирования выделения именованных сущностей и связей между ними

### Выгрузка датасета

## Разработка и программная реализация способа графового представления именованных сущностей и связей между ними

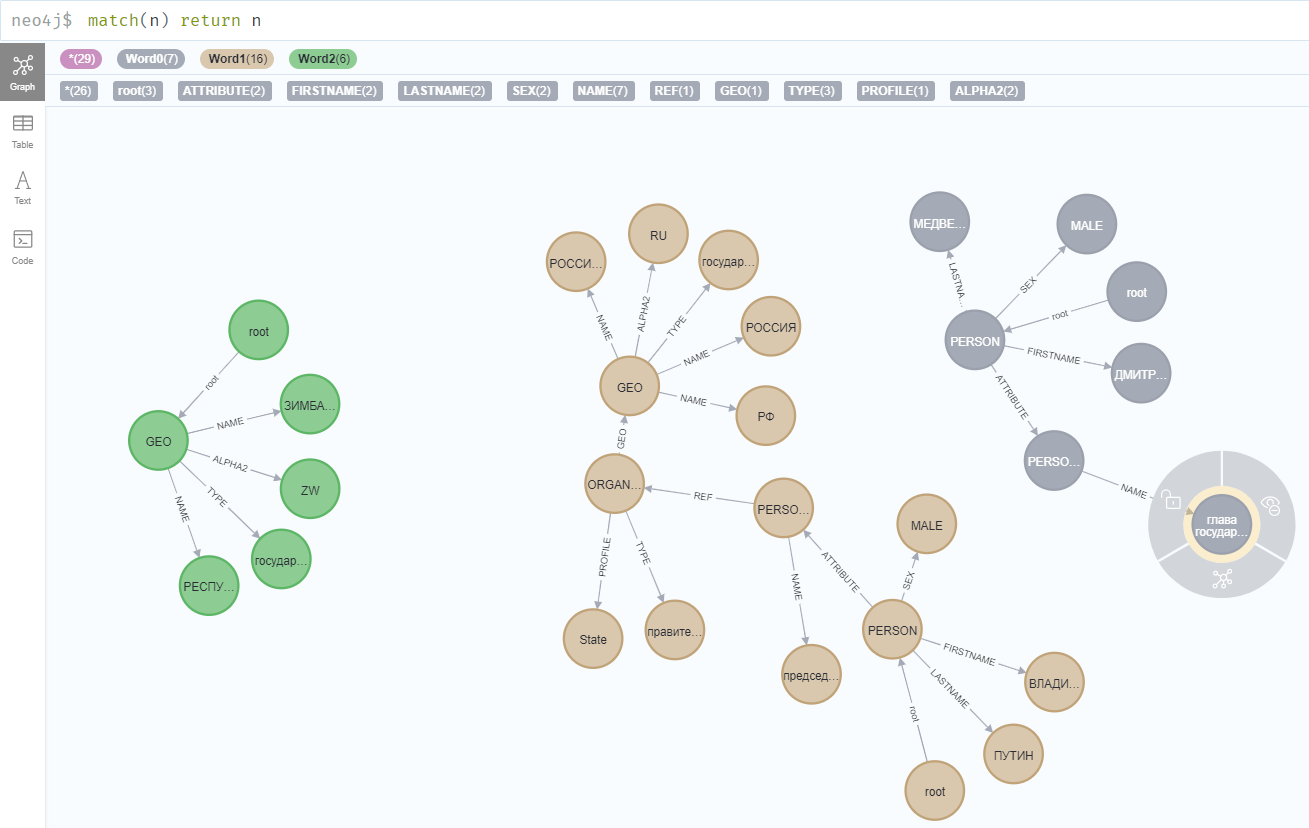
### Постороенние графа



### Демонстрация графа в neo4j

Запрос :



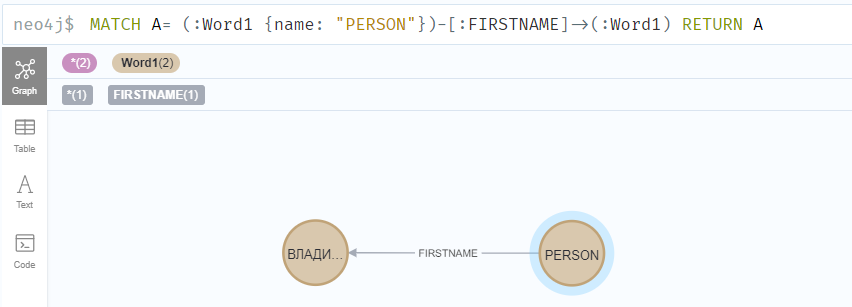


Результат:

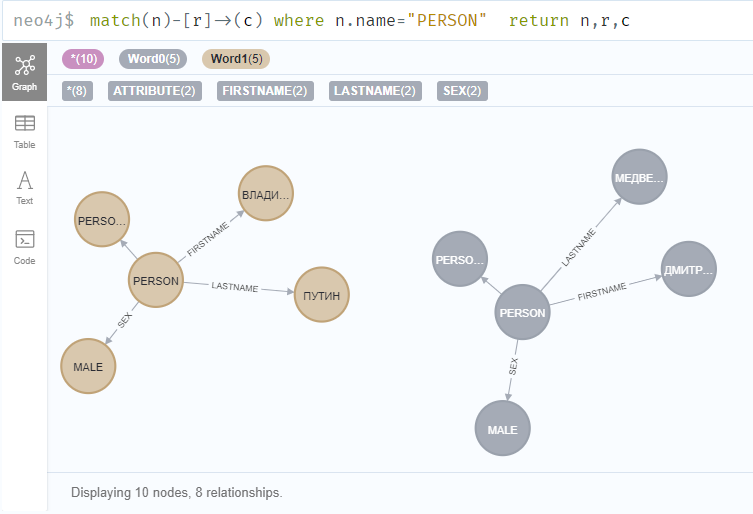
## Сохранение результата графового представления именованных сущностей в графовую БД

### Тестирование работоспособности на запросах пользователя

#### Отображение имени



#### Отображение всех людей и информации них



# Заключение

## В ходе работы по построению семантической сети было выполнен ряд задач, перечисленных ниже.

Направлением дальнейших исследований может быть повышение количества найденных именованных сущностей и отношений между ними, комбинирование различных NLP решений, создание своих правил выявления отношений, дообучения готовых моделей.

# **Список литературы**

# ПРИЛОЖЕНИЕ 1 Перечень тегов именнованых сущностей библиотеки Pullenti

