Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»
Физтех-школа Прикладной Математики и Информатики
Кафедра банковских информационных технологий

**Направление подготовки / специальность:** 01.04.02 Прикладная математика и информатика (магистратура)
**Направленность (профиль) подготовки:** Банковские информационные технологии

# ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВОПРОСНО-ОТВЕТНОЙ СИСТЕМЫ НА ОСНОВЕ ГРАФОВ ЗНАНИЙ

(магистерская диссертация)

**Студент:**
Сомов Олег Дмитриевич

_____
*(подпись студента)*

**Научный руководитель:**
Воронцов Константин Вячеславович,
д-р физ.-мат. наук

_____
*(подпись научного руководителя)*

**Консультант** *(при наличии)***:**
_____
_____
*(подпись консультанта)*

Москва 2020

ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

В мире постоянно создается огромное количество данных. Но без инструмента по извлечению знаний информации из этих данных не приносили бы столько пользы сколько приносят сегодня. В мире существует много инструментов информационного поиска. Эти инструменты разделяются на два типа – для поиска в неструктурированных данных и в структурированных.

Пример неструктурированных данных – текст. Задачу поиска нужной информации в таких данных хорошо решают поисковые сервисы, вроде Google и Яндекса. Основной инструмент в этой задаче – полнотекстовый поиск. Структурированные данные имеют определенную структуру хранения, которая позволяет быстро получать доступ к нужным элементам и легко обрабатывается компьютером. Пример структурированных данных - таблицы, XML/JSON файлы, графы. Поиск в таком документе представляет собой построение некоторого пути до искомого элемента.

Современные методы поиска информации с помощью нейронных сетей, позволяют с высокой долей вероятности найти ответ на простой вопрос к неструктурированному документу. Такой способ имеет простую архитектуру, так как представляет собой одну компоненту – модель нейронной сети, которая на вход принимает вопрос и входной текст, а на выходе выдает наиболее вероятную подстроку в исходном тексте, которая содержит ответ.

Но с усложнением вопроса и увеличением количества текстовых документов возникают следующие проблемы:

1. Если модель ранее не видела входной текст, она с высокой вероятностью не сможет дать верный ответ на вопрос.
2. В коллекции документов ответ на вопрос может содержаться не прямым указанием в тексте, а косвенно. В таком случае модель не

уловит семантику текста и не сможет указать на нужную подстроку, потому что ее просто нет в тексте.

3. Документы, собранные из разных источников и объединенные в одну коллекцию, могут содержать противоречивые данные. Поэтому ответы модели могут быть неверными. Устранять противоречия в большой коллекции текстовых документов сложно.
4. Модель может не верно обрабатывать парафразы, и будет давать разные ответы на вопросы одинаковые по смыслу, но разные по формулировке.

Поиск ответа в структурированной информации обычно представляет собой комплексную систему. Такая система состоит из нескольких компонент, каждый позволяет распознать более глубоко семантику вопроса и в конце концов сформировать определенный путь в структуре документа, который приведет к ответу на вопрос пользователя.

Такой подход решает проблемы, с которыми сталкивается первый подход.

1. Модель на структурированных данных не опирается в поиске ответа на предоставленные данные, она разбирает входной запрос и далее строит путь в структурированном документе, который приведет к элементу с ответом.
2. Структурированный документ позволяет найти путь между двумя элементами в любом случае.
3. Несмотря на то, что любая такая система будет в большей степени опираться на предоставленные данные, структурированную информацию легче обновлять и изменять без нарушения общей логики.
4. Система на структурированных данных более устойчива к парафразам, потому что опирается более на саму структуру вопроса, а не на его текстовое представление.

Задача, которая решается в данной магистерской работе – построение open-domain вопросно-ответной системы по доменам. Сложность решения

ODQA заключается в многообразии знаний и их представлений. Для того чтобы эффективно решить задачу созданию такой вопросно-ответной системы, необходимо иметь огромный объем данных. Большие корпорации, вроде Google или Яндекса справляются с этой задачей благодаря большому количестве данных в поисковом индексе и большому количеству вычислительных мощностей.

В тоже время, существует большое количество открытых структурированных данных по многим областям знаний. Эти ресурсы позволяют построить вопросно-ответные системы по доменам, даже не обладая данными и мощностями поисковиков. Также возможно объединение построенных вопросно-ответных систем в одну open-domain систему.

В мире главный лидер по использованию графов знаний в вопросно-ответных системах это Google. В своей поисковой выдаче он опирается на результат полнотекстового поиска и собственного графа знаний. В России у компании Mail.ru также есть граф знаний, собранный из открытых источников. Еще есть реализации вопросно-ответных систем на основании структурированных данных у лабораторий ведущих институтов России – ИТМО и МФТИ (лаборатория Deep Pavlov). В тоже время Яндекс полагается полностью на полнотекстовый поиск в своем поисковом движке. Такое малое количество работ в России может быть обусловлено двумя факторами:

- На русском языке очень мало открытых структурированных данных. На английском это WikiData, DBPedia, Google Freebase, Yelp, Yago и другие. На русском есть только часть WikiData и DBPedia, которые оба являются частью проекта Wikipedia. Данных в этих двух базах знаний достаточно много, но очень мало по сравнению с количеством информации на английском языке. Для сравнения в Wikidata 89 млн сущностей на английском языке, в то время как на русском около 30 млн. Граф знаний Google был переведен на русский язык за счет

огромного количества корпусов текста доступных Google и большого количества вычислительных мощностей компании.

- Русский язык является синтетическим языком - в слове объединяются, образуя синтез, лексические и грамматические значения. Английский же – аналитический язык, где слово это передатчик лексического значения, а грамматические значения передаются отдельно: порядком слов в предложении, служебными словами, интонацией. Из-за этого в русском языке плохо работают алгоритмы анализа синтаксиса и морфологии, а подход с построением вопросно-ответной системы на структурированных данных как раз подразумевает такой разбор. К примеру, подходы по извлечению триплетов, которые хорошо работают на английском языке (на основании синтаксических парсеров) дают плохие результаты на русском.

Предмет данной работы - создание инструмента по созданию доменных вопросно-ответных систем. Мотивация этого исследования заключается в улучшении методологии разработки доменных вопросно-ответных систем. Главные причины необходимости улучшения:

- Большинство вопросно-ответных систем плохо масштабируемы и не позволяют добавить новый домен (область знаний) быстро и без ущерба для качества.
- Используется большой объем неструктурированных данных - обработка требует высоких вычислительных мощностей
- Сложно контролировать корректность текстовых данных и модифицировать их.

Объектом исследования данной работы выступает изучение методологий построения вопросно-ответных систем. В данной работе приведены алгоритмы, которые решают сопутствующие задаче создания вопросно-ответной системы проблемы.

Для решения задачи построения вопросно-ответной системы в данной работе используются методы NLP – архитектура трансформер для

векторизации и классификации текста, рекуррентные нейронные сети для извлечения именованных сущностей, синтаксические и морфологические анализаторы. Для разбора предложения также используются регулярные выражения и контекстно-свободные грамматики, там, где использование машинного обучения не целесообразно. Разработка системы велась на языке Python с использованием фреймворков для обучения нейронных сетей Keras и PyTorch.

В конце работы автор приводит сравнение разработанной системы с другими вопросно-ответными системами на основании структурированных данных. Это вопросно-ответная система QAnswer и вопросно-ответная система KBQA от DeepPavlov.

# ТИПЫ ВОПРОСНО-ОТВЕТНЫХ СИСТЕМ И ПОДХОДЫ К ИХ СОЗДАНИЮ

Создание вопросно ответной системы это комплексная задача обработки вопроса на естественном языке, которая может включает в себя задачи по излечению именованных сущностей, классификации типа вопроса или конца, начала ответа, сопоставление сущностей, разрешения таксономических отношений между сущностями и другие. С другой стороны, работа с вопросно-ответными системами часто подразумевает работу с данными. Данные должны быть полны, корректны и не должны противоречить друг другу.

Существуют следующие способы по построению вопросно-ответных систем к любому домену знаний:

1) **Поиск ближайшего отвеченного вопроса**. Входной вопрос сравнивается с каждым вопросов среди набора вопросов и среди всех отбирается самый ближайший вопрос на основании метрики. Благодаря современным методам векторизации и технологий поиска ближайших соседей (например, FAISS от Facebook (Johnson, 2019)) возможно среди миллионов вопросов искать ближайший к входному вопросу вопрос из коллекции и давать на него ответ. Проблема с этим подходом состоит в том, что для построения полной вопросной ответной системы по какому либо домену знаний, необходимо не только сгенерировать огромное количество вопросов, но и расположить их в пространстве таким образом, чтобы находить тот самый вопрос, на который есть верный ответ.
2) **Reader Ranker модель**, например DrQA (Chen, 2017). Модель сначала делает ранжирование по документам на основании TF-IDF метрики относительно входного запроса $q$ и далее за счет нейронной сети предсказывает строку в документе $x_k$, где содержится ответ. Обычно такая система очень долго работает из-за того, что в Reader модель нужно передать входной вопрос вместе с найденным

документом. Из-за этого входной вектор в модель нельзя предрассчитать для каждого вопроса, а необходимо каждый раз пропускать через сеть $[q, x_k]$. Результат работы системы – подстрока в тексте с найденным ответом.

Алгоритм предсказания подстроки в тексте (Devlin, 2019) следующий. Над векторами состояния базовой нейронной сети строится еще одна нейронная сеть, которая на основании эмбеддинга вопроса и документа выучивает начальный $S$ и конечный $E$ вектор состояния. В качестве базовой модели используется либо рекуррентная нейронная сеть, либо трансформер. Далее для каждого слова $T_i$ рассчитывается скалярное произведение $S \cdot T_i$ и $E \cdot T_i$. Вероятность данного слова быть началом/концом ответа в предложении рассчитывается, как (1).

$$P_i^S = \frac{e^{S \cdot T_i}}{\sum e^{S \cdot T_j}}, \qquad P_i^E = \frac{e^{E \cdot T_i}}{\sum e^{E \cdot T_j}} \tag{1}$$

Предсказание нужной подстроки в тексте рассчитывается как максимальное значение $S \cdot T_i + E \cdot T_j$, для таких объектов $T$, где $i < j$. Данная нейронная сеть обучается с помощью логистической функции ошибки.

На этапе ранжирования извлекается top-N документов, поэтому такую операцию нужно выполнить несколько раз. Сами документы обычно небольшие, так как нейронные сети обычно не могут обрабатывать последовательности более определённой длины. Например, максимальная длина токенизированной последовательности нейронной модели BERT составляет всего 512 токенов.

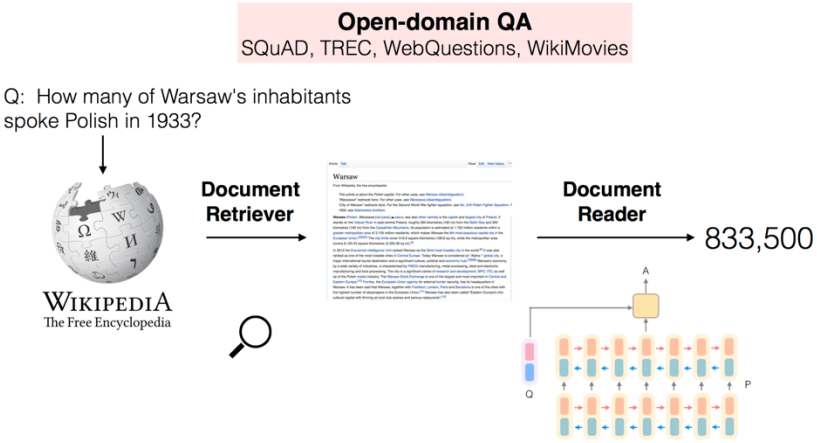


*Рисунок 1. Reader Ranker модель*

3) **Преобразование вопроса в запрос к базе данных.** Входной запрос преобразовывается в SQL к базе данных. Такие системы обычно состоят из множества компонент и моделей – извлечение именованных сущностей, классификация типа запроса, определение типа сущности и так далее. В тоже время существуют такие архитектуры нейронных сетей, которые позволяют решать задачу перевода из вопроса на естественном языке в одну компоненту.

   Основной плюс такого подхода — это структурированность данных, которая позволяет строить гибкую и сложную систему и полный контроль над получаемой информацией и входным запросом.

В данной работе автор использует третий подход – преобразование вопроса на естественном языке в запрос к графовой БД (Unger, 2014). Существует также несколько подходов к решению в рамках этого подхода к задаче.

1) **Машинный перевод из естественного языка в SQL запрос** (Soru, 2017). Входной вопрос представляется в виде линеаризованного графа. Далее с помощью Seq2Seq архитектуры данная последовательность переводится в некоторый SQL запрос. Данный подход подразумевает извлечение S-P-O триплетов из вопроса, что достаточно проблематично в русском языка из-за нестрогого синтаксиса.

2) **Text2SQL** (Hwang, 2019). Система представляет собой нейронную сеть, которой на вход подается вопрос к реляционной таблице на естественном языке и названия колонок таблицы. Обучение происходит без использования контента таблицы. Система состоит из нескольких моделей, где каждая предсказывает определенный компонент SQL запроса – SELECT, AGGREGATION, COLUMN или WHERE. Проблема данного подхода – ограниченное количество вопросов, которые можно задать к такой системе. Также сложно использовать данную систему для построения вопросно-ответной системы из-за ограниченного количества возможных SQL запросов и сложности добавления оператора JOIN.
3) **Комплексная система преобразования вопроса в запрос** (Tahri, 2013). Система состоит из числа компонент, где каждый решает свою задачу для построения конечного SQL запроса – выделение именованных сущностей, нечеткий поиск, классификация типа вопроса, морфологический и синтаксический анализ. Этот подход представлен в этой работе

Автор решает задачу построения вопросно-ответной системы переводом входного вопроса в соответствующий SQL запрос. Решить задачу построения вопросно-ответной системы таким образом позволяет гибкая структура данных.

В работе в качестве базовой структуры данных автор использует граф. Графовая структура данных позволяет для каждого вопроса определить подграф, в котором находится ответ на вопрос. Входной запрос преобразуется в SPARQL запрос к графовой базе данных. В части работы про построение графа будет подробно рассказано о преимуществах использования графа в качестве структуры данных и о том, каким образом был создан граф для разрабатываемой системы.

# ПОСТРОЕНИЕ ГРАФА ЗНАНИЙ

## ОНТОЛОГИИ И ГРАФЫ ЗНАНИЙ

Онтология – часть философии, изучающая фундаментальные принципы бытия, его наиболее общие сущности и категории, структуру и закономерности [20].

В информатике онтология это попытка формализовать определенную область знаний с помощью концептуальной схемы. Схема включает в себя отношения между сущностями, типы связей и правила, на основании которых эти связи были установлены, а также иерархию классов.

Онтологии обладают следующими свойствами:

- Онтология легко обрабатывается компьютером
- Онтология всегда создается чтобы решить определенную задачу
- Онтологии состоят из:
    - **Экземпляров** – основная единица онтологии. Экземпляр — это узел графа.
    - **Классов** – абстракции объектов. Классы — это узлы графа, которые позволяют строить таксономические или иерархические отношения.
    - **Атрибуты** – атрибуты экземпляров и объектов. Атрибуты — это свойства узлов или ребер графа.
    - **Предикаты** – отношения между экземплярами. Предикат — это направленное ребро графа.

Обычно онтология представляется в виде – субъект-предикат-объект. Пример онтологии – *Пушкин – место рождения – Москва*. Онтология — это направленный граф, где направление задают предикаты. Объект и субъект в онтологии — это узлы графа, предикат – связь. Текстовые описания онтологии – Пушкин, Москва, место рождения – являются метаинформацией графа.

Графы знаний состоят из онтологий. Приведем свойства графа знаний:

- **Структурированный** – представлен в виде графовой структуры. В информатике граф представляется с помощью матрицы смежности.
- **Стандартизированный** – состоит из двух типов элементов – сущности и предикаты. В графе сущности это узлы графа, а предикаты – ребра. В графах знаний ребра направленные.
- **Связанный** – каждый узел графа связан как минимум с другим одним узлом в графе.
- **Декларативный** – граф знаний самодостаточен сам по себе и не требует дополнительных источников информации.
- **Аннотированный** – каждая сущность графа имеет текстовое описание.
- **Большой** – обычно графы знаний содержат миллионы сущностей.
- **Цикличный** – в графе знаний могут возникать циклы.
- **Мульти-граф** – каждая сущность может иметь более одного типа связей с другой сущностью.

В информатике ориентированный граф обычно представлен матрицей смежности. Граф знаний – это ориентированный граф вместе с метаданными.

Для представления связанных данных в интернете была представлена модель данных RDF (Resource Description Framework) (Decker, 2000). Каждая сущность представляет собой лишь ID узла. Все текстовые описания является еще 1 узлом в данном графе, связанные с соответствующим узлом графа. Каждая сущность является либо объектом, либо субъектом, либо и объектом, и субъектом одновременно. Существует несколько форматов файлов модели RDF - turtle, n3, n-triples. RDF формат данных используется для графов знаний.

```
@prefix ex: <http://example.com/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:msg1 rdf:type ex:msg;
        ex:to "bob";
        ex:from "alice" ;
        ex:subj "alice to bob 1";
        .
```

*Рисунок 2. Пример формата данных Turtle*

Существует и другое представление графа, которое используется в графовых базах данных - Property graph. В такой модели данных каждая сущность графа (узел, связь) обладает произвольным набором описывающих ее свойств. Это отличает эту модель от RDF, где каждая сущность представлена лишь идентификатором. Текстовое описание в RDF формате является еще одним узлом в графе, на который через предикат ссылается сущность. Такая модель представления используется там, где в первую очередь важны связи между сущностями, а свойства узлов графа играют роль дополнительной информация. Модель property graph используется для хранения социальных графов или графов транзакций.

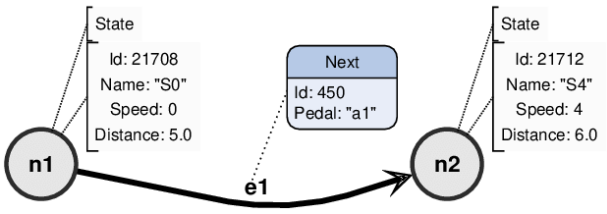


*Рисунок 3. Пример Property graph модели графа для хранения в базах данных*

В тоже время в графах знаний бывает очень полезно для предикатов иметь свойства. Это помогает избежать дублирования и порождения лишних типов связей. Поэтому в RDF также было добавлен механизм реификации. Реификация это утверждение об утверждении, которое представляет собой не только факт, но и дополнительную сущность, которая указывает на свойство предиката.

Например, утверждение *“Барак Обама стал президентом США первый раз в 20 января 2009 года”* с помощью реификации в онтологии возможно выразить так – Барак Обама – президент -США – 20 января 1999. Далее в

работе будет использована механика реификации для обогащения предикатов графа знаний.

## ОПРЕДЕЛЕНИЕ СТРУКТУРЫ ГРАФА И СОЗДАНИЕ ГРАФА ЗНАНИЙ

В этом исследовании автор проектирует вопросно-ответную систему для русского языка для свободного количества доменов на основании базового графа. В качестве базового графа был использован граф знаний WikiData (Vrandečić, 2014). В интернете WikiData это веб-сервис, который представляет интерфейс над графом знаний. Страница сущности представляет собой узел в графе, где описаны отношения смежных сущностей в WikiData к данной. На рисунке 4 представлено как выглядит интерфейс WikiData и выделены его основные элементы для одной сущности WikiData.

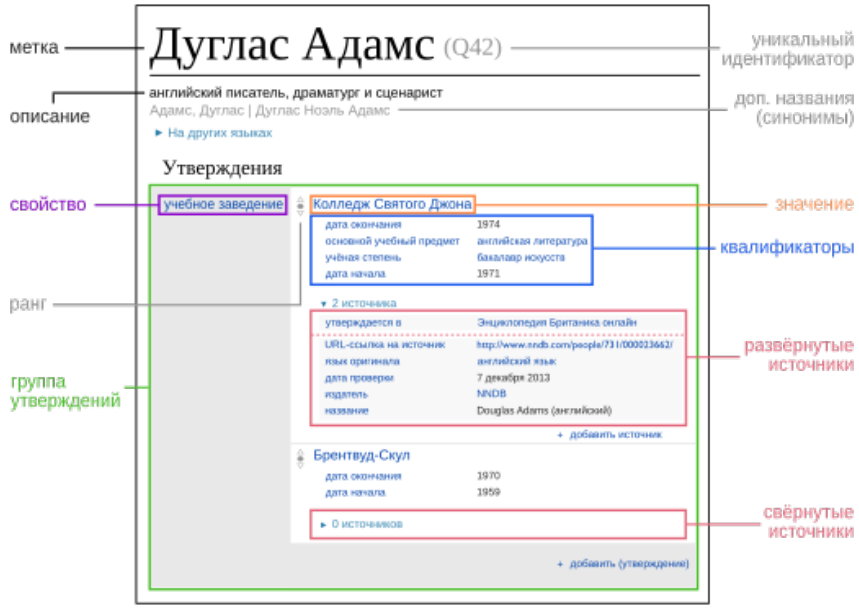


*Рисунок 4. Пример элемента WikiData*

Объясним основные элементы графа знаний WikiData:

- **Свойство** – это предикат субъекта сущности WikiData. На примере это учебное заведение.
- **Уникальный идентификатор** – индекс в базе знаний WikiData. Каждая сущность в WikiData имеет свой уникальный идентификатор. Индекс обозначает узел графа. В данном случае это индекс субъекта.
- **Группа утверждений** – каждый элемент группы утверждений является объектом в модели RDF. Каждый элемент является смежным

узлом субъекту. Каждый объект в свою очередь ведет на свою страницу WikiData.

- **Квалификатор** – это элемент WikiData, который позволяет выразить утверждение об утверждении. В случае если нужно чтобы предикат имел свойство, в триплет RDF добавляется еще 1 элемент – квалификатор. Тогда триплет RDF выглядит следующим образом – субъект – предикат – объект – квалификатор. Добавление квалификатора есть операция реификации. В то время как в property graph модели возможно произвольное множество свойств у предикатов, в RDF модели данных WikiData возможен только одно свойство у предиката.

Представленная в работе вопросно-ответная система подразумевает масштабируемость на новые домены, поэтому необходим стандартизируемый процесс для создания еще одного домена знаний вопросно-ответной системы. Рассмотрим процесс создания графа знаний для одного домена знаний, например для кино. Так как разрабатываемая система планируется к использованию в виртуальном ассистенте, сбор возможных вопросов к системе происходит на краудсорс платформе, где пользователи имитируют будущих пользователей вопросно-ответной системы.

Первый этап заключается в анализе вопросов будущих пользователей к домену. Проводится исследование, которое помогает понять какие типы вопросов к домену могут быть, про что пользователь спрашивает больше всего, какие типы сущностей важны для домена, какие типы связей необходимы. В первую очередь собирается большая репрезентативная выборка возможных вопросов. На краудсорс платформе запускается задание следующего вида:

*Представьте, что вы разговариваете с виртуальным ассистентом.*

*Спросите его что-либо про кино.*

Когда собрано достаточное количество вопросов (обычно около 4000-5000 примеров), проводится анализ данных и ищутся ответы на следующие вопросы:

1. **Какие типы вопросов возможны в домене?** – Ответ позволяет понять какие вопросы пользователи задают к системе и какие категории вопросов в домене есть.
2. **Какие типы сущностей чаще всего фигурируют в вопросах?** – Ответ позволяет понять из каких типов сущностей будет состоять граф домена. В домене фильмы это фильмы, люди, страны, жанры, в музыке – музыкальные произведения (альбомы, треки), исполнители, группы, жанры, страны и другие.
3. **Какие типы связей чаще всего фигурируют в вопросах?** – Ответ позволяет понять какие предикаты возможны у сущностей графа.
4. **Необходимо ли подключать еще какие-то источники знаний или WikiData достаточно?** – Если в вопросах часто фигурируют вопросы с сущностями, которых нет в базовом графе, в нашем случае WikiData, тогда используются другие источники данных. В графе знаний, построенном в этой работе, используется два дополнительных источника данных – MovieDB и Musicbrainz. MovieDB использовался для получения дополнительно информации о домене кино, а Musicbrainz – для получения информации о треках и альбомах исполнителей.

Первый вопрос дает понять сложность входных запросов. Так как система подразумевает семантический и синтаксический разбор предложения, чем сложнее входной вопрос, тем сложнее будет провести полный разбор предложения. Например, вопрос *“Кто снял фильм Титаник?”* – очень прост для разбора, так как состоит из триплета *“x – снял – Титаник”*. А вопрос – *“Какой последний фильм режиссера, который родился в 1954 году?”* уже сложнее для разбора. Потенциальные вопросы, которые пользователи могут

задать в вопросно-ответную систему для первых двух доменов позволили понять каким образом строить систему, которая сможет отвечать на большинство вопросов пользователей.

Следующие два вопроса отвечают на вопрос – из каких сущностей и связей формировать граф знаний. Граф WikiData имеет большое количество таксономических отношений внутри себя, так как задачей этого проекта является создание универсального графа знаний проекта Wikipedia. Например, сущность фильм является наследником последовательности абстракций WikiData *"визуальное произведение искусства – произведение искусства – творческая работа – продукт умственного труда – произведение- результат усилий – объект – философское понятие - понятие"* для построения полной иерархии внутри графа. Для решения представленной задачи создания ВОС — это лишнее и только усложнит разработку системы. Четвертый вопрос дает понять какие структурированные источники стоит еще подключить для максимального покрытия вопросов к домену, если информации, представленной в базовом графе недостаточно для построения полноценной ВОС.

Правила построения графа знаний для вопросно-ответной системы, следующие:

1. Все ID узлов графа знаний являются элементами из одного множества
2. Из одного узла графа знаний можно попасть в другой, не проходя при этом через узел, который задает таксономические/иерархические правила
3. Основная индексация узлов графа идет из WikiData. Добавление нового источника данных подразумевает создание нового пространства имен со своими индексами, таким образом что из узла под индексом WikiData можно попасть в узел нового источника данных, не проходя через узлы, которые задают таксономические/иерархические правила.

Ввиду правил, указанных выше, анализ возможных вопросов к системе дает понять какие типы сущностей необходимо взять из базового графа для добавления в граф знаний для ВОС. Для графов фильмы и музыка это

- *Фильм*
- *Сериал*
- *Серия фильмов*
- *Человек*
- *Книга*
- *Персонаж*
- *Место*
- *Жанр*
- *Награда*
- *Исполнитель*
- *Группа*
- *Релиз*
- *Альбом*
- *Трек*

Аналогично анализ вопросов помогает определить типы связей, которые надо добавить в граф. В основном это все предикаты, которые есть в WikiData у отобранных типов сущностей.

- Информация о человеке (место рождения, год рождения и т.д.)
- Информация о произведении (год выпуска, актеры, длительность и т.д.)
- Связь с другими сущностями графа (награда, жанр)

Первый вопрос позволяет ответить на вопрос – на какие подграфы можно разделить полученный граф знаний для того, чтобы возможно было решать задачу классификации области нахождения ответа в графе. Эта информация не только даст нам возможность определить какие подграфы будут

присутствовать в разрабатываемой ВОС, но и каким образом будет создаваться граф знаний для ВОС.

В предыдущих вопросах, данные, собранные на краудсорс платформе, позволили понять, каким образом будет выглядеть граф знаний. Были определены узлы и ребра графа, дополнительные источники данных и возможные типы вопросов в систему.

Но полученные сущности и связи между ними еще не позволяют построить ВОС. Выявляются следующие проблемы:

1) В WikiData определенные области вопросов и ответов находятся в несколько переходов графов от сущности в вопросе (из-за усложненной иерархии WikiData). Из-за этого процесс генерации вопроса на естественном языке к графу WikiData значительно усложняется.
2) Данные имеют такой формат, который не поддерживается базой данных. Например, представлены в таком формате, который поддерживается только базой данных WikiData. Например, в некоторых базах данных не поддерживаются квалификаторы.
3) В графе WikiData нет информации, которая позволяет ответить возможные вопросы к системе. Пример вопроса, на который было бы который сложно ответить с помощью начального графа WikiData – *“Самые известные фильмы Джека Николсона”* так как в графе знаний WikiData нет меры “популярности” сущностей.
4) Каждый домен знаний обладает своей спецификой и без анализа области нельзя формировать доменные графы в ВОС.

Для того чтобы решить указанные выше проблемы, было решено формировать граф из подграфов. Из данных разметки было выделено 20 областей графа к домену фильмы и 24 области графа к домену музыки, к которым задаются вопросы на естественном языке. Примеры подграфов:

- **Атрибуты фильмов** – год выпуска, бюджет, кассовые сборы и т.д.

- **Атрибуты человека** – год рождения, место рождения, родственники и т.д.
- **Персонажи произведения**
- **Музыкальные группы** – атрибуты музыкальной группы (участники, начало карьеры, награды и т.д.)
- **Фильмы по книге** – по какой книге был снят тот или иной фильмы
- **Саундтреки фильмов** – что за музыка была в фильме и кто ее исполнял
- **Атрибуты музыкальных релизов** – продюсер, награды, дата выхода, список треков
- **Совместное сотрудничество** – совместная работа 2ух и более человек из мира кино или музыки.
- **Фильмы человека** – популярные фильмы с которыми работал актер
- **Популярные исполнители** – популярные исполнители внутри какого-либо жанра
- **Ограничения по возрасту** – ограничения по возрасту на фильм
- **Фильмы, которые скоро выйдут на экраны**

Подграф - узлы базового графа WikiData, которые соединены 1-2 переходами между друг другом. Рассмотрим, как выглядит подграф – Фильмы человека. Главный элемент – узел обозначающий подграф. Он связан с 3 другими подграфами – недавние фильмы кинодеятеля, популярные фильмы кинодеятеля, знаменитые фильмы кинодеятеля. Те, в свою очередь уже смежны с базовыми сущностями графа (фильмы, люди и т.д.) Далее в работе такой элемент будет называться *View*. Благодаря такой логике, возможно точно определить из какого подграфа необходимо извлечь ответ на заданный вопрос.

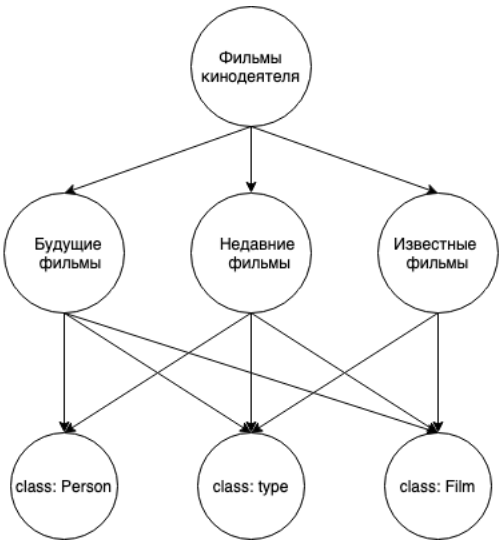


*Рисунок 5. Пример View "Фильмы кинодеятеля"*

Рассмотрим процесс создания такой структуры. Для создания одного View нужно сформировать 4-5 SPARQL запросов, которые извлекут необходимую информацию, касающуюся данного View, из базового графа. В SPALQL можно указать какие сущности графа и какие типы связей необходимо извлечь из базового графа.

На основании подграфов данные WikiData группируются в 1 файл формата RDF (обычно это turtle), который далее с помощью движка графовой БД (например, Apache Jena (Jena, 2007)) загружается в финальный граф знаний. Извлечение подграфов позволяет не только извлечь только самое необходимое из базового графа, но и сокращает количество переходов в запросе с 3-5 до 1-2.

Приведем последовательность этапов из которых состоит формирования графа знаний после того, как произведен анализ вопросов пользователей:

1) Последнее состояние графа знаний WikiData загружается в локальную базу данных.
2) С помощью SPARQL извлекаются подграфы из загруженного графа знаний.
3) Результат набора SPARQL запросов загружается в новый граф знаний.

4) В граф, через новые предикаты к созданным подграфам добавляются новые источники данных.
5) Полученный граф знаний проверяется на корректность и целостность. Это значит проверяется что граф не имеет ID, которые не имеют текстового значения, то есть терминального узла.

В итоге финальный граф знаний это результат исполнения сформированных на основании пользовательских вопросов SPARQL запросов. В общем, процесс формирования графа знаний выглядит как на рисунке 6.

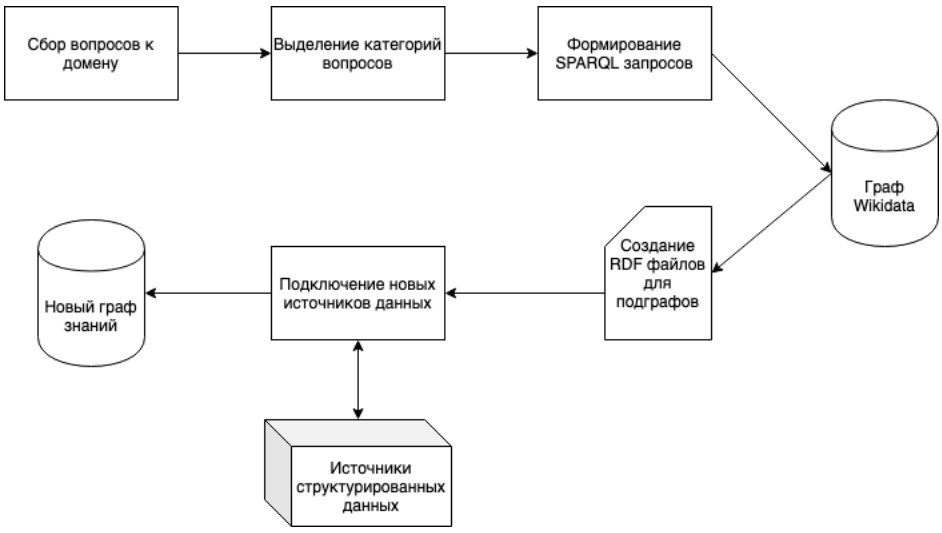


*Рисунок 6. Процесс формирования графа знаний.*

В результате повторения этого процесса для двух доменов был создан граф знаний. В нем нет лишних сущностей (которые не касаются двух доменов – музыка и фильмы). Новые источники данных – MovieDB и Musicbrainz подключены к узлам Wikidata через новые предикаты. Всего в графе знаний 23 млн сущностей, а занимаемый им объём – 1.5 гигабайта.

# ВЫБОР ГРАФОВОЙ БАЗЫ ДАННЫХ

Очень важная часть системы – это непосредственное хранение данных. Есть несколько способов хранить графы знаний. В главе про онтологии и графы знаний было упомянуто 2 формата хранения – RDF хранилище и Property graph. Такие форматы используются для хранения в статичных базах данных. Другой подход – это хранения графа в формате Property graph в оперативной памяти RAM. Так как система подразумевает промышленную эксплуатацию, нужно очень ответственно подойти к выбору подходящей БД.

Приведем плюсы и минусы каждого подхода к хранению графа знаний:

1. Хранение данных в базе данных
    a. Хранение графа в базе данных позволяет сократить затраты на разработку алгоритмов поиска и хранения данных. Современные базы данных осуществляют быстрый поиск данных и способны выдерживать высокую нагрузку на сервис. Также базы данных при текущих объемах потребления потребляют небольшое количество оперативной памяти и CPU, что позволяет экономить на вычислительных мощностях. Для взаимодействия с БД, можно использовать REST API и взаимодействовать через язык запросов с графом знаний.
    b. Добавление еще 1 компоненты системы повышает риск выхода из строя системы. Для предотвращения таких ситуаций к базе данных подключаются сервисы логирования и мониторинга.
2. Хранение данных в оперативной памяти компьютера
    a. Хранение графа в оперативной памяти компьютера позволяет снизить количество компонент системы и уменьшить риск выхода из строя сервиса.
    b. Хранение графа в оперативной памяти значительно увеличивает потребление CPU и RAM. Плюсы, которые были упомянуты в преимуществах хранения в базе данных, также

отсутствуют при такой реализации – обход графа и запросы к нему нужно реализовывать самостоятельно, нет поддержки языка запросов.

На основании выводов выше, в данной работе использовалась графовая база данных, а не инструменты для хранения графа в оперативной памяти компьютера.

Следующий этап выбора базы – оценка подходящих баз данных, основанная на фреймворке оценки графовых баз данных Iguana (Conrads, 2017). Алгоритм оценки базы данных приведен на схеме. В качестве технологического стека оценки были использованы Telegraph агент для мониторинга CPU/RAM сервера, Grafana для визуализации потребления ресурсов, Influx DB для сохранения метрик и Jmeter управления процессом нагрузочного тестирования. В качестве slave инфраструктуры были использованы вычислительные мощности Google Cloud.

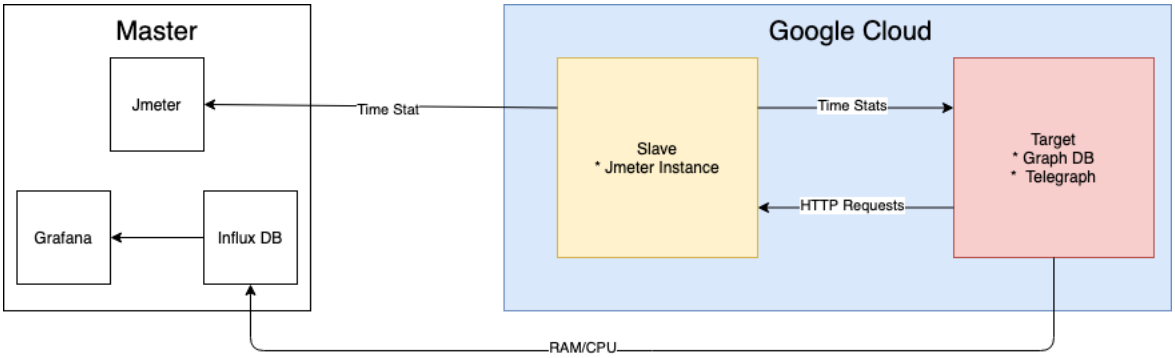


*Рисунок 7. Схема тестирования*

На основании экспертной количественной оценки было выбрано 3 оцениваемых базы данных – OpenLink Virtuoso (Erling, 2012), BlazeGraph (SYSTAP, 2015) и GraphDB (Popov, 2004).

Система спроектирована таким образом, что пользователь отправляет в базу сгенерированные ей SPARQL запросы. Для оценки системы имитируется поведение пользователя и создаются различные варианты возможных SPARQL запросов в систему. Вопросы создаются к различным подграфам графа знаний и в них вставляются возможные предикаты и сущности.

Приведем примеры таких запросов:

- Какие фильмы снял Квентин Тарантино?

```
SELECT ?xLabel WHERE {
        ?x wdt:P57 wd:Q3772.
        ?x rdfs:label ?xLabel.
}
```

- Какие фильмы выйдут в эту зиму?

```
Select ?xLabel  WHERE {
        ?upcomingFilms a dr:upcoming_films_subgraph ;
        dr:has_film_info [ dr:has_film ?x ] ;
        dr:has_period “winter” .
        ?x rdfs:label ?xLabel.
}
```

- С какого возраста можно смотреть фильм Аватар?

```
Select ?xLabel WHERE {
        ?film a dr:film_rating_subgraph ;
        dr:has_film wd:Q24871;
        dr:has_min_age ?xLabel.
}
```

Запросы отправляются на HTTP сервер базы данных от множества потоков, где каждый поток есть 1 пользователь. Система тестирования настроена таким образом, чтобы база подвергается нагрузке в 300 tps (transactions per second). В таких условиях, ожидается что база будет возвращать ответ в районе 5-10 мс.

Таким образом, замеряется время, которое требуется базе данных вернуть ответ (принимая во внимание передачу данных по HTTP). Во все время тестирования фиксируется время ответа базы, а также потребление RAM и CPU.

| | Open Link Virtuoso | Ontotext GraphDB | Blazegraph |
|---|---|---|---|
| *Query #1* | **5** | 6 | 7 |
| *Query #2* | **10** | 12 | 13 |
| *Query #3* | **7** | 9 | 11 |

*Рисунок 8. Частичная схема результатов времени ответа баз данных*

Основными требованиям к базе являются:

1. Время ответа в миллисекундах
1. 99 перцентиль времени ответа базы
2. Потребление CPU/RAM

Нагрузочное тестирование проводилось в течении 12 часов.

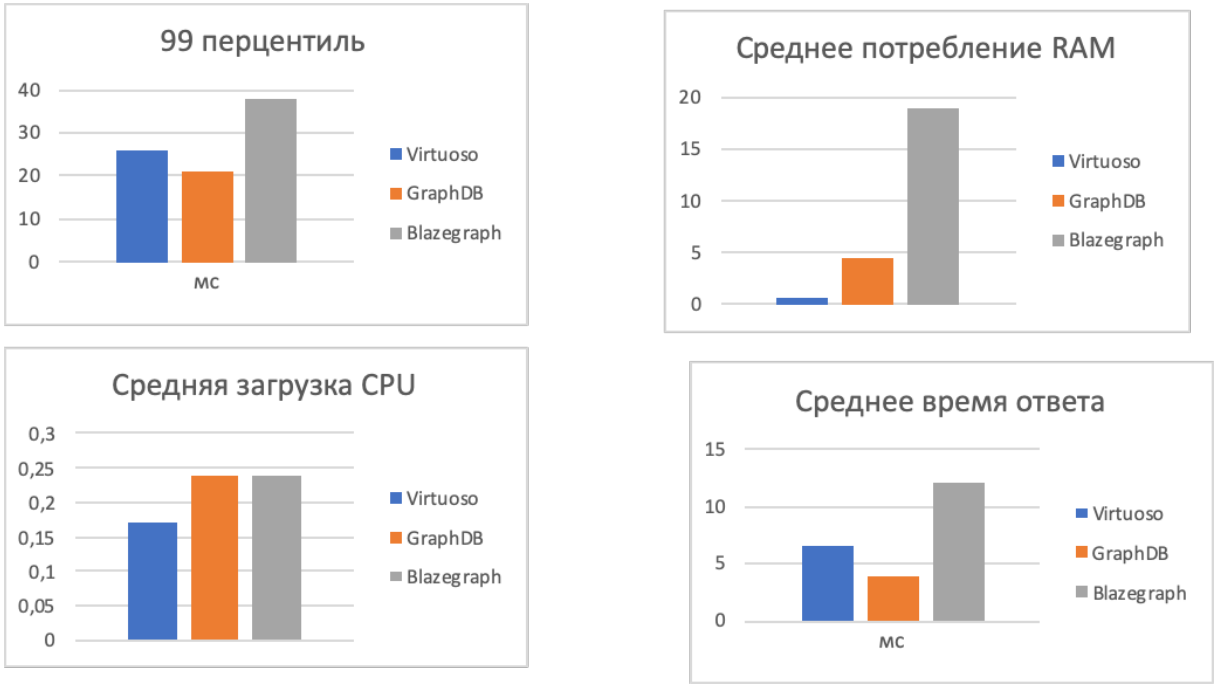


*Рисунок 9 Результаты нагрузочного тестирования графовых баз данных*

Самой эффективной и производительной базой данных показала себя OpenLink Virtuoso со средним временем ответа 7 мс и очень низким потреблением оперативной памяти. Стоит отметить, что OpenLink Virtuoso используется в проекте DBPedia (Auer, 2007). DBPedia – граф знаний проекта Wikipedia.

# РАЗРАБОТКА ВОПРОСНО-ОТВЕТНОЙ СИСТЕМЫ

## ТРЕБОВАНИЯ И ОГРАНИЧЕНИЯ К ВОПРОСНО-ОТВЕТНОЙ СИСТЕМЕ

Представленная вопросно-ответная система — это комплексный процесс преобразования вопроса на естественном языке в запрос к базе данных. В этой части работы будет представлен подробный разбор всех компонент системы. Логика системы построена на основании созданного графа. Так как система представляет собой продукт, который в будущем будет использоваться в промышленной эксплуатации, представим требования к такой системе:

- На вопросы одинаковые по смыслу, но разные по формулировке должен быть дан 1 ответ.
- Объем используемых данных не более 1 Гб на одну доменную область. Это правило необходимо, так как с повышением объема данных, возрастают требования к производительности системы. Данное ограничение на объём данных является оптимальным в плане ресурсопотребления и производительности.
- Используемые данные должны содержать большинство информации о домене и не должны противоречить друг другу.
- Система должна обладать высоким ответным качеством выше 50%
- Входные вопросы быстро обрабатываются и ответ возвращается пользователю за 200-300 мс.

Также при разработке были установлены следующие ограничения на систему:

- Данные домена структурированы в формате RDF.
- Система не поддерживает комплексные вопросы, которые состоят одновременно из двух вопросов. Например, какой исполнитель родился в Америке и снялся в фильме отряд Самоубийц?

- В системе нет диалогового менеджера, который позволяет следить за состоянием диалога и задавать уточняющие вопросы.
- Есть возможность использовать доменную краудсорс разметку.

На основании этих ограничений и требований и была разработана представленная система.

# ПЛАН СИСТЕМЫ

Разработанная вопросно-ответная система — это способ навигации графа знаний с помощью естественного языка. Способ навигации это SPARQL запросы. Система преобразует входной вопрос на естественном языка в SPARQL запрос, который исполняется в графовой базе данных и извлекает из графа знаний ответ на вопрос.

Граф знаний состоит из подграфов, каждый подграф содержит ответ на определенную категорию вопросов домена. Граф знаний имеет небольшую иерархию сущностей внутри себя. Практически все сущности графа — это сущности которые фигурируют в вопросах пользователей.

Любой прямой фактологический вопрос каждого типа представим в виде некоторого пути в графе. В системе нет свободной генерации SPARQL запроса, а есть несколько этапов классификации, которые позволяют определить соответствующий запросу SPARQL шаблон.

Ввиду упрощенной иерархии сущностей внутри графа в системе нет этапа разрешения таксономии. Определение классов сущности графа происходит на основании модели извлечения сущностей и модуля нечеткого поиска. Например, если из вопроса было извлечено имя актера, то тот факт, что извлеченная сущность является подклассом человека не важен, в следствии чего не нужно включать компонент, который бы за это отвечал, в систему. Это значительно отличает созданный граф знаний от графа знаний WikiData. WikiData была создана для хранения знаний, для удобства представления этих знаний в ней фигурирует множество иерархических сущностей и связей. Граф для ВОС необходим только для поиска ответа по нему.

Представим все компоненты системы. Для удобства разметим порядок выполнения операций цифрами.

1. Классификация домена позволяет определить какая нейронной модель по извлечению сущностей будет пользоваться для извлечения сущностей.
2. NER модуль выделяет именованные сущности из входного вопроса. В доменной модели свой набор именованных сущностей.
3. Нечеткий поиск именованных сущностей определяет какому ID в базе знаний соответствует извлеченная именованная сущность.
4. Классификация подграфа отвечает за классификацию входного вопроса в подграф, который содержит ответ на данный вопрос
5. Определение шаблона запроса помогает определить какому SPARQL запросу внутри подграфа соответствует входной запрос.
6. Обработчик запроса выделяет детали запроса вопроса, которые помогают сформулировать его конечную цель, такие как “сколько”, “первый/второй/последний” и т.д. Обработчик запроса также взаимодействует с нечетким поиском, если вопрос имеет несколько сущностей и необходимо искать сущности зависимо друг от друга.
7. Генерация ответа – на основании входных сущностей, сущностей из вопроса и заготовленных шаблонов ответа генерируется вопрос на естественном языке.

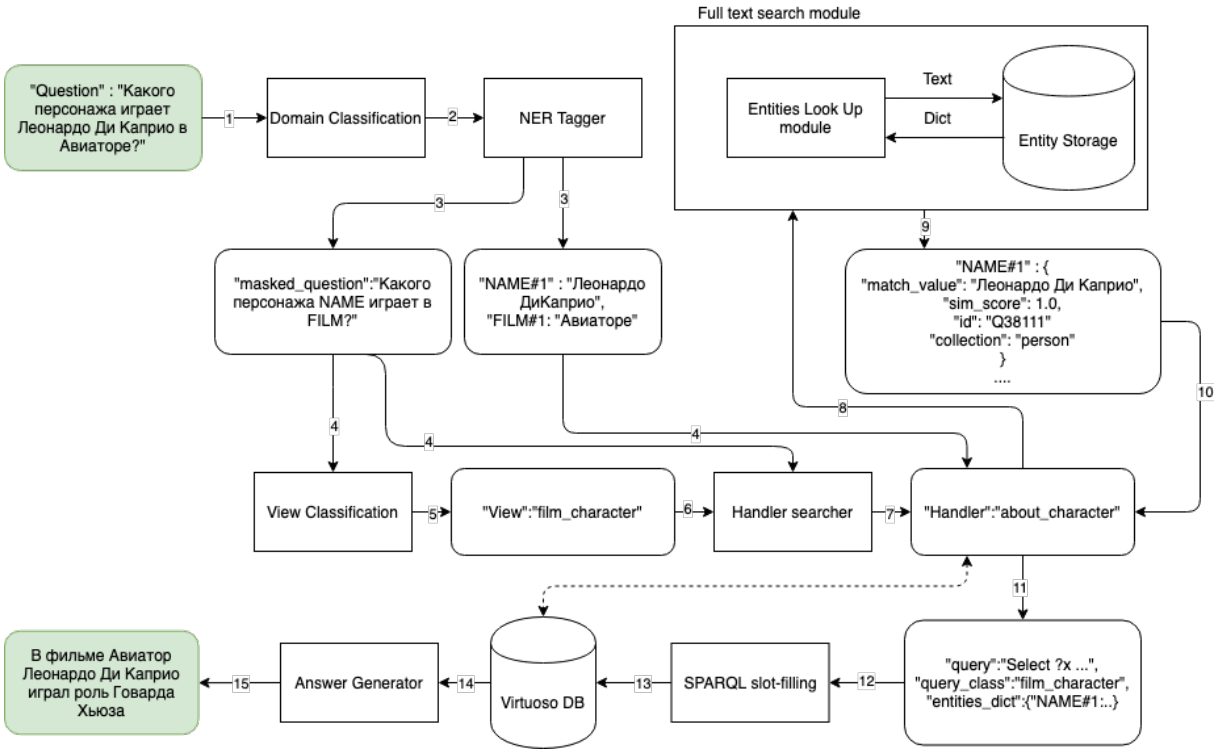


*Рисунок 10. План системы*

Далее в работе будет разобрана каждая компонента системы, приведены разработанные алгоритмы и получившиеся метрики.

## КЛАССИФИКАЦИЯ ДОМЕНА

Первый этап вопросно ответной системы – классификация домена вопроса. Хотя представленная система предусматривает частичную возможность задавать кросс-доменные вопросы, например такие как *“Сколько альбомов записал актер кино Джаред Лето?”*, это происходит за счет архитектуры системы, чем за счет кросс-доменной функциональности.

Для того чтобы в общем хорошо отвечать на вопросы пользователей в разработанной вопросно-ответной системе для каждого домена есть своя модель для извлечения именованных сущностей. Для того чтобы использовать верную модель необходимо определить домен входного запроса. Это делается с помощью модели классификации домена.

Модель классификации домена представляет собой двухслойную нейронную сеть с логистической функцией ошибки.

$$logloss(a, y) = -\sum y * log\, a \qquad (1)$$

Модель оптимизировалась с помощью метода адаптивной оценки момента Adam (Kingma, 2014) с шагом 1e-3 и гиперпараметрами $\beta_1 = 0.95$ и $\beta_2 = 0.995$ .

В качестве базовых эмбеддингов используются вектора языковой модели BERT (Devlin, 2019). BERT – Bidirectional Encoder Representations from Transformers – языковая модель обученная на новой архитектуре нейронной сети – трансформер. BERT это стек из трансформеров. Эта модель обучается на двух типах задач – маскирование слова и предсказание логичности следующего предсказания.

Преимущество нейронной модели BERT перед использованием рекуррентных нейронных моделей вроде GRU (gated recurrent unit) (Chung, 2014) или LSTM (long short term memory) (Schmidthuber, 1999) это способность нейронной сети увидеть весь контекст предложения сразу с помощью механизма внимания.

Обыкновенные Seq2seq модели, которые используют упомянутые выше модели плохо справляются с этим, так как сжимают всю информацию в один вектор и далее пытаются из этого вектора получить информацию для решения своих задач. Проблема в том, что при работе с длинными последовательностями, качество работы такого алгоритма значительно ухудшается, так как нейронная сеть не может восстановить необходимую последовательность из вектора энкодера, например в задаче языкового моделирования.

Механизм внимания (Vaswani, 2017) позволяет определить важность других слов в предложении относительно текущего обрабатываемого слова. Механизм внимания реализуется следующим образом:

$$Attention(Q, K, V) = softmax\left(\frac{Q * K^T}{\sqrt{d_k}}\right)V \qquad (2)$$

$Q, K, V$ – это матрицы векторов, которые получаются скалярным умножением эмбеддинга $X$ рассматриваемого слова на матрицы $W^Q, W^K, W^V$. Они обучаются одновременно с другими весами нейронной сети. Расшифруем эти вектора:

- Q(query) – матрица которая является новым векторным представлением, которое получилось после линейного преобразования входного эмбеддинга слова $X$. Далее именно этот вектор используется для расчета линейной зависимости текущего слова и других слов предложения
- K(key), V(value) – представляет собой еще одно векторное представление слов, относительно которых вычисляется близость одного слова и другого. В теории, K и V могут быть равными.

В BERT используется multi-head механизм внимания. В таком случае у нас появляется несколько векторных представлений матрицы внимания, что позволяет улучшить качество того насколько модель понимает каким образом слова связаны в предложении. В сети для этого добавляются новые

вектора $Q, K, V$ . После того как были вычислены матрицы внимания, результаты операций всех механизмов внимания конкатенируются в 1 матрицу и подаются на последующий полносвязный слой трансформера.

Нейронная модель BERT состоит из стека трансформеров. Трансформер – архитектура модели, состоящая из механизма внимания и полносвязной нейронной сети.

Для нормализации векторов между механизмом внимания и полносвязной нейронной сеть добавляется метод батч-норм (3).

$$\overline{x}^k = \frac{x^k - E(x^k)}{\sqrt{D(x^k)}} \tag{3}$$

Батч-норм позволяет ускорить сходимость нейронной сети и быстрее помочь достичь глобального минимума оптимизируемой функции.

В обучении нейронной модели BERT есть две фазы – предобучение и оптимизация. В процессе предобучения нейронной сети, BERT обучается за счет суммы логистических ошибок на двух задачах – маскирование слов (LM) и предсказание логичности следующего предложения (NSP). В таком случае, на вход нейронной сети BERT подается две предложения разделённые токеном SEP, для обучения с помощью задачи NSP. После этапа токенизирования входного предложения, 15% слов предложений маскируется для обучения на задаче LM. Функция ошибки при обучении нейронной модели BERT- сумма функций потерь на двух задачах — NSP и LM.

Обычно, после этапа предобучения, BERT оптимизируется под определенную задачу. Обычно оптимизируется 1-4 последних слоя BERT (трансформеры). Функция ошибки в таком случае выбирается исходя из задачи, под которую BERT оптимизируется. В данной работе используемый BERT был обучен на корпусе русской Wikipedia, без оптимизации ни под какую определенную задачу.

За счет архитектуры и логики обучения BERT позволяет достаточно точно закодировать предложения в вектор, сохранив семантику предложения.

Например, на рисунке эмбеддинги BERT позволяют разнести по пространству вектора, которые относятся к разным типам обращения в колл-центр. Исходное пространство BERT размером 768 было снижено с помощью алгоритма t-SNE (Hinton, 2008) до двумерного.

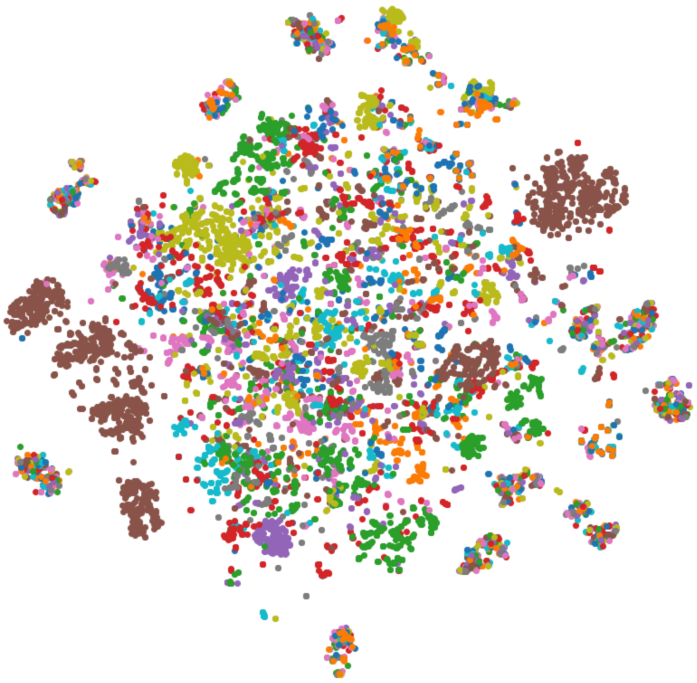

*Рисунок 11. Пример размещения обращений клиентов колл-центра, закодированных с помощью BERT после нелинейного снижения размерности*

Видно, что BERT хорошо разносит объекты различных типов обращений друг от друга в $R^2$ пространстве. Ввиду большой репрезентативной способности BERT в задаче классификации обращений можно использовать какой-либо метрический алгоритм, например KNN и уже достичь высокого качества в определении вида обращений.

В задаче классификации домена 2 класса кино и музыка, модель классификации решает бинарную задачу классификации, в тоже время при масштабировании системы на N доменов, будет решаться задача мультиклассовой классификации.

Для двух доменов в данной работе получены следующие метрики:

| | F1-macro |
|---|---|
| Кино | 0.98 |
| Музыка | 0.97 |

*Таблица 1. Метрики классификации домена*

# ИЗВЛЕЧЕНИЕ ИМЕННОВАННЫХ СУЩНОСТЕЙ

Ключевой этап в формировании SPARQL запроса – извлечение информации о сущностях упомянутых в вопросе. В SPARQL запросе используются ID узлов графа которые в свою очередь соответствуют именованным сущностям упомянутым в запросе. Получение ID сущности упомянутой в вопросе происходит в два этапа – извлечение именованной сущности из вопроса и поиск соответствующего данной сущности ID в базе знаний. Данная глава диплома посвящена извлечению именованных сущностей из вопроса.

В работе используются один подход по извлечению именованных сущностей - извлечение именованных сущностей с помощью нейронных сетей. На прошлом этапе был определен домен вопроса, поэтому алгоритмы по извлечению именованных сущностей работают с учетом этой информации.

Модель извлечения именованных сущностей изображена на рисунке 12.

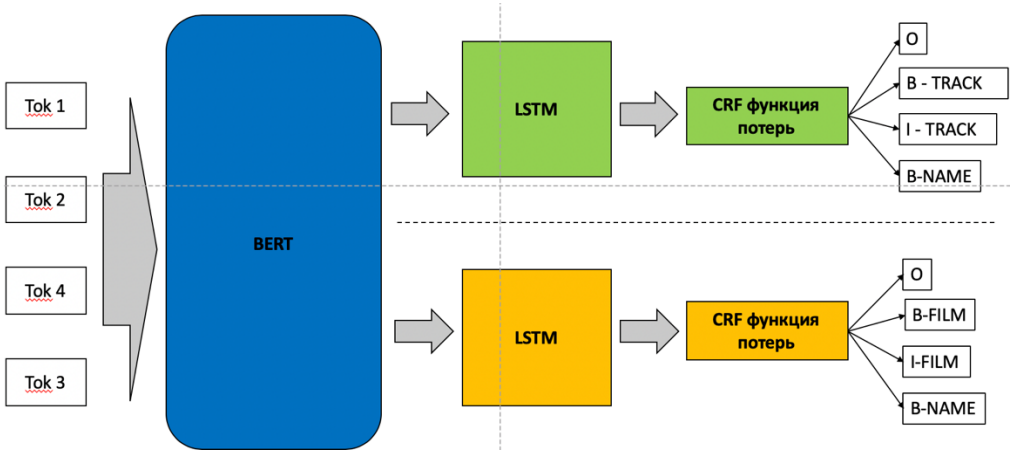


*Рисунок 12. Архитектура модели распознавания именованных сущностей*

Для решения задачи извлечения именованных сущностей используются эмбеддинги нейронной модели BERT для токенов. Далее используется нейронная архитектура Bi-LSTM и CRF функция (Lample, 2016) потерь на основании BERT эмбеддингов для извлечения именованных сущностей. Для каждого домена обучена своя модель извлечения именованных сущностей.

Bi-LSTM это тип рекуррентной нейронной сети, которая способна выучивать длительные зависимости между словами в предложении. Входные эмбеддинги подаются на два направления LSTM сети, тем самым позволяя нейронной сети “запомнить контекст” для каждого слова. Над Bi-LSTM построен скрытый слой $f_1$ куда передается матрица векторов контекстных представлений $h_t$ Bi-LSTM сети. На выходе слоя $f_1$ получается матрица которая далее передается в CRF функцию потерь. Построенная модель состоит из 3 слоев – Bi-LSTM, полносвязный слой, дропаут и CRF функция потерь. Модель обучалась методом адаптивной оценки моментов с шагом 1e-2 и гиперпараметрами реализации Adam в фреймворке глубокого обучения Keras по умолчанию.

CRF (Conditional Random Fields) – графическая дискриминативная нейронная модель, которая является разновидностью марковских случайных полей. Данная модель строит не совместное распределение $p(y, x)$, а условное $p(y|x)$. Совместное распределение набора случайных величин $x$ в марковском случайном поле вычисляется по формуле (4).

$$p(y|x) = \frac{1}{Z(X)} \prod_c \phi_c(y, x) \quad (4)$$

$\phi_c(y, x)$ – потенциальная функция аргументом которой является состояние вершин на $c$ группе смежных вершин.

В линейном условном случайном поле потенциальная функция имеет вид (5). В случае с нейронными сетями потенциальная функция – это 1 слой нейронной сети.

$$\phi_c(x_k) = exp(\sum_t \lambda_s f_s(y_t, y_{t-1} x_t)) \quad (5)$$

Где s – индекс функции $f_s$, $\lambda_s \in R$

Коэффициент нормализации $Z$ вычисляется по формуле (6).

$$Z(x) = \sum_i \prod_c \phi_c(\mathrm{y_i}, x) \quad (6)$$

CRF слой выучивает распределение вероятностей для классов $A \in R^{N,N}$, где N количество меток классов, что позволяет моделировать условную вероятность $p(y|x)$. Для последовательности токеном $x \in R_n$ на выходе Bi-LSTM получается матрица векторов скрытых состояний Bi-LSTM, которая умножается на матрицу весов $P_{i,y_i}$.

Для данной последовательности предсказаний $y_1, \dots y_P$ формула вычисления предсказания определяется следующим образом – сумма условной вероятности данного объекта и его класса плюс сумма условной вероятности данного класса и класса следующего за ним объекта (7).

$$S(x,y) = \sum_{i=1}^{T-1} A_{y_i,y_{i+1}} + \sum_{i=1}^{T} P_{i,y_i} \quad (7)$$

Вероятность считается вычислением по softmax формуле (8).

$$p(y|x) = \frac{e^{S(x,y)}}{\sum_{y' \in y} e^{S(x,y')}} \quad (8)$$

Функция потерь для обучения нейронной сети (9) – это отрицательная логарифмическая вероятность.

$$ln\, p(y|x) = S(x,y) - ln \sum_{y' \in y} e^{S(x,y')} \quad (9)$$

Первоначальный подход представлял собой одну мультиклассовую модель, но такая модель имела множество ошибок первого рода. В результате, был проведен эксперимент с построением модели, когда для одного тега в предложении было предсказано несколько классов различными моделями, а не один с наибольшей вероятностью, это привело к повышению метрик.

В задаче предсказания для каждого слова предсказывается тег слова с префиксом B (начало именованной сущности) и I (продолжение

именованной сущности) в метриках представлено среднее качество по метрике F1 для B и I классов сущности.

| | F1-macro |
|---|---|
| FILM | 0.85 |
| NAME | 0.83 |
| GROUP | 0.75 |
| GENRE | 0.95 |
| COUNTRY | 0.98 |
| RELEASE | 0.86 |
| BOOK | 0.85 |
| CHARACTER | 0.78 |

*Таблица 2. Метрики модели по извлечению именованных сущностей*

## НЕЧЕТКИЙ ПОИСК СУЩНОСТЕЙ

Следующий этап – поиск соответствующего идентификатора извлеченной из вопроса сущности в базе знаний для заполнения SPARQL запроса.

Сущности графа домена разделены в наборы коллекций. Например, в домене фильмов есть коллекция кинодеятелей, фильмов, жанров, стран. В музыке – альбомы, треки, исполнители, группы. Коллекция – это набор сущностей графа, представленные в формате – ID – (текстовое описание сущности, признаки сущности). С помощью коллекции можно найти либо главную сущность, от которой можно осуществлять поиск смежных сущностей которые указаны в вопросе, либо просто сущность, которая фигурирует в вопросе, независимо от других сущностей. В качестве признаков сущности используется количество просмотров на русской Wikipedia.

В поиске именованных сущности используется нейронная модель BERT. Текстовое описание сущности векторизуется с помощью нейронной модели BERT. Коллекция векторизованных сущностей сохранятся в индекс библиотеки приближенного поиска ближайших соседей. В данной работе используется библиотека FAISS. Так как в графе знаний для двух доменов больше 20 миллионов сущностей, а сущностей каждого типа более 100 000 такое решение считается обоснованным.

Алгоритм данной реализации поиска, следующий - для запроса $x$ находят $w$ ближайших центроидов. Для центроидов собираются списки векторов, до которых рассчитываются расстояния. Затем выбираются $k$ ближайших соседей среди этих векторов.

В данной работе используется следующий алгоритм нечеткого поиска сущностей графа знаний.

Извлеченная сущность векторизуется с помощью BERT и загружается в FAISS для поиска ближайших $N$ соседей на основании определённого порога $t$. Далее $N$ соседей ранжируются на основании метрики, полученной

из алгоритма сравнения строк. Для большинства коллекций оптимальным алгоритмом для сравнения строк является расстояние Левенштейна. В тоже время, для определенных коллекций (например имен людей) наилучшим алгоритмом был алгоритм нечеткого поиска Жаро-Винклера. Алгоритм нечеткого поиска, $N$, $t$ – гиперпараметры данного модуля.

Последний шаг – ранжирование сущностей. В данный момент основной признак ранжирования сущностей – количество просмотров на Wikipedia. Это позволяет решить ситуацию с указанием одинаковых названий сущностей, например Титаник 1953 года и Титаник 1997. Так как поиск может ошибаться, например найдя ближайшей не ту сущность в графе, дополнительное ранжирование помогает выбрать наиболее популярную сущность, которая очень схожа с искомой и часто это помогает выбрать верную сущность.

Тестирование алгоритма проводилось - из каждой коллекции выбиралось подмножество сущностей *M*, часть которого оставалась в таком же виде, а часть меняется – менялись местами слова в сущности (Сильвестр Сталлоне -> Сталлоне Сильвестр), создавались опечатки (Леонардо ДиКаприо -> Леанардо ДиКаприо).

Задача алгоритма – найти корректно все сущности подмножества $M$ в исходной коллекции. Метрика качества в такой задаче точность – сколько сущностей с валидационного датасета найдено в исходном датасете. Число в скобках – процент таких сущностей в валидационном датасете.

| | Неизменённые сущности (0.5) | Опечатка в сущности (0.25) | Замена порядка (0.25) | Общая точность |
|---|---|---|---|---|
| Фильмы | 1 | 0.74 | 0.83 | 0.915 |
| Кинодеятели | 0.99 | 0.42 | 0.485 | 0.781 |
| Исполнители | 1 | 0.77 | 0.805 | 0.916 |
| Музыкальные произведения | 0.99 | 0.68 | 0.69 | 0.875 |
| Страны | 0.984 | - | - | 0.984 |
| Жанры | 0.921 | - | - | 0.921 |
| Книги | 0.99 | 0.88 | 0.91 | 0.955 |
| Персонажи | 0.98 | 0.725 | 0.71 | 0.988 |

*Таблица 3. Метрики алгоритма нечеткого поиска*

## КЛАССИФИКАЦИЯ ПОДГРАФА

Важный этап в вопросно-ответной системе – локализация области ответа. В различных вопросно-ответных системах это происходит по-разному – в системах на основании коллекций документов, коллекция ранжируется от наиболее вероятного документа с ответом к наименее, в структурированных данных выделяется наиболее вероятная область данных с ответом. В этой работе автор локализует подграф, в котором вероятнее всего находится ответ на заданный вопрос. В главе про создание графа была представлена логика создания подграфов на основании запросов пользователя. Идея локализации области с ответом состоит в том, что новый вопрос пользователя попадет в одну из областей графа, которая была сформирована на этапе создания графа. Таким образом решается задача мультиклассовой классификации. Так как вопросно-ответная система предназначена для активного использования, необходимо разработать способ быстрого добавления новых *View* в систему для большего покрытия возможных типов вопросов.

Простая модель машинного обучения не подходит, так как такой алгоритм обновления будет подразумевать постоянное переобучение всей модели, что является долгим и трудозатратным процессом. Каждый раз для новой модели нужно будет подбирать новую архитектуру и другие параметры для максимального качества предсказания.

Поэтому в данной работе был применен другой подход – построение такого нелинейного преобразования, которое позволит отличать примеры из одного класса от другого, так что каждый раз при добавлении нового подграфа не будет происходить переобучения модели, а будет произведено нелинейное преобразование входного вектора в новое векторное пространство меньшей размерности и поиск в нем ближайшего соседа. Суть данного подхода – выучивание некоторой метрики, которое будет отличать одни классы от других на основании метрического подхода. Впервые такой подход был

использован компанией Facebook для классификации изображений, в этой же работе используется для классификации вопросов (Schroff, 2015).

В машинном обучении используются метрические методы, как способ измерить расстояние между объектами. Самая популярная – параметризируемая метрика Миньковского (10) на Евклидовом пространстве. Частные случаи этой метрики – Манхэттенская ($p = 1$), Евклидова ($p = 2$).

$$d(x_i, x_j) = \left(\sum |x_i - x_j|^p\right)^{\frac{1}{p}} \quad (10)$$

Также часто используется косинусная близость (11), которая позволяет определить расстояние между объектами, как угол между ними.

$$\cos(x, x') = \frac{x^T x'}{||x|| ||x'||} \quad (11)$$

Эти метрики используются в метрических алгоритмах, таких как KNN или K-means.

Чаще всего в машинном обучения для измерения расстояния между двумя объектами используется евклидова метрика. Основная проблема этой метрики - признаки в данных могут иметь разный масштаб и подсчет расстояния между двумя векторами должен происходит без учета этого масштаба. Метрика Малаханобиса (12) позволяет учесть этот масштаб.

Метрика Малаханобиса:

- Учитывает тот факт, что дисперсия по разным направлениям разная.
- Учитывает ковариацию между переменными.
- Стандартизирует некоррелированные значения.

$$d^2(x_i, x_j) = (x_i - x_j)^T M^{-1} (x_i - x_j) \quad (12)$$

M - ковариационная матрица (положительно определенная).

Метрика Малаханобиса эквивалентна Евклидовой метрике после линейного преобразования (13).

$$M^{-1} = P^T P$$
$$d^2(x_i, x_j) = (x_i - x_j)^T P^T P (x_i - x_j) = (Px_i - Px_j)^T (Px_i - Px_j) \quad (13)$$

Также можно использовать нелинейные преобразования в метрике Малаханобиса (kernel trick) (14).

$$d_\phi(x, x') = ||\phi(x) - \phi(x')|| \quad (14)$$

В случае диагональной матрицы, получается стандартизированная евклидова метрика, потому в таком случае не учитывается ковариация объектов.

Логику метрики Малаханобиса возможно использовать в задаче классификации. Рассмотрим подход Metric Learning и функцию ошибки для выучивания такой метрики.

На практике задача Metric Learning ставится так – максимизация межклассового расстояния, при сохранение внутриклассового расстояния меньше определенного порога α (15).

$$\max \sum_{(x_i, x_j) \in D} d_M(x_i, x_j), \sum_{(x_i, x_j) \in S} d_M(x_i, x_j) \leq \alpha \quad (15)$$

Оптимизация задачи происходит пересчетом матрицы M на каждом шаге до сходимости (Xing, 2003). Но при больших размерах матрицы M и при большом количестве ограничений на задачу оптимизации, находить матрицу M становиться невозможно ввиду высокой вычислительной сложности.

На помощь приходит онлайн-обучение, где на вход алгоритма приходит 1 семпл данных (Anchor(A), Positive(P), Negative(N)) и на основании функции потерь, например triplet loss, вычисляется ошибка (17). Ошибка показывает – насколько расстояние от A до P меньше, чем расстояние от A до N.

$$F(A, P, N) = \max((f(A) - f(P)) - (f(A) - f(N)) + \alpha, 0) \quad (17)$$

Выучивается такое отношение между объектами, что объекты из одного класса располагаются рядом в некотором пространстве, а объекты из разных классов располагаются на бóльшем расстоянии друг от друга.

Metric Learning возможно использовать для решения практических задач в области машинного обучения. Например, классификация объектов.

В качестве нейронной сети для решения такой задачи обычно используется DSSM нейронная сеть с тремя базовыми нейронными сетями (Huang, 2013). Нейронная сеть в DSSM архитектуре представляет собой 2-3 слойную нейронную сеть с нелинейными функциями активации. Веса нейронной сети у всей DSSM сети одинаковые. Такой подход позволяет одновременно подать в нейронную сеть N векторов в N сетей и посчитать функцию потерь относительно этих N векторов, а не только относительно одного вектора и целевого значения.

Алгоритм решения задачи классификации с использованием metric learning подхода состоит из преобразования исходных объектов в нужное пространство и последующая классификация алгоритмами машинного обучения, например KNN.

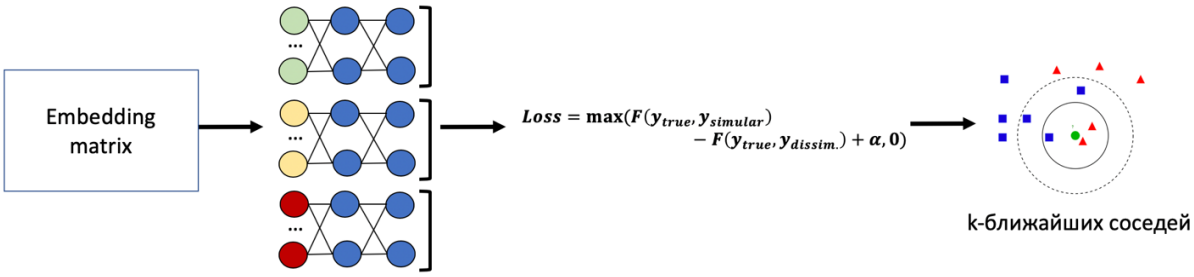


*Рисунок 13. Алгоритм классификации подграфа*

Как было указано выше можно использовать нелинейные преобразования при подсчете метрик и вычислять итеративно оптимальные матрицы преобразования. Также есть другие метрики, позволяющие нужным образом расположить объекты в пространстве.

Если использовать такой алгоритм для, например классификаций обращений, то можно наблюдать вот такую картину. Слева входные вектора в DSSM сеть обращений пользователя, справа после нелинейного

преобразования путём выучивания метрики. Видно, что на правой картинке, объекты из разных классов находятся на определенном расстоянии друг от друга.

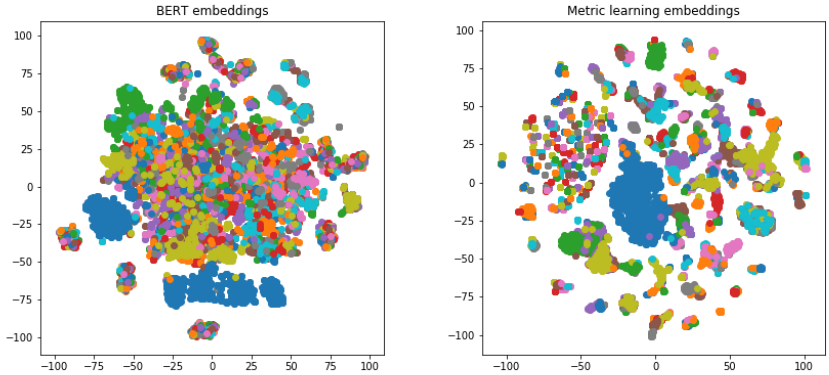


*Рисунок 14. Пример работы Metric Learning*

Есть несколько преимуществ данного подхода:

- На каждый класс можно иметь небольшое количество семплов и при этом отделять один класс от другого.
- При добавлении нового класса, нет необходимости переучивать всю модель.
- Так как для обучения такой большой мульти-классовой модели необходима бы была очень глубокая нейронная сеть, то высока вероятность что сеть переобучилась бы на текущих данных и не смогла бы дать хорошее качество предсказания на валидационной выборке.

Данный подход был применен в задаче определения подграфа в графе знаний. В качестве независимой переменной используются маскированные вопросы на естественном языке, а в качестве зависимой переменной – подграф, который содержит вопрос на естественном языке.

Для построения модели, которая сможет правильно относить вопрос в нужный подграф необходим датасет различных формулировок вопроса. Возможные формулировки вопроса собираются на краудсорс платформе, где пользователям предлагается задать вопрос к определённому графу знаний, при этом все ключевые для SPARQL запроса именованные сущности,

указанные в вопросе, заменяются соответствующей маской – FILM (для названия фильма), NAME (имя человека) и тд.

Результат тренировки нейронной сети – нелинейное преобразование, которое объекты из одного класса располагает близко в пространстве, а из разных далеко друг от друга. Выучиваемое нелинейное преобразование является нейронной сетью с 2 скрытыми слоями с нелинейной функцией активации ReLU (Rectified Linear Unit) (18).

$$f(x) = max\ (0, x) \tag{18}$$

Нейронная сеть состоит из 2 скрытых слоя размером 256 и 126.

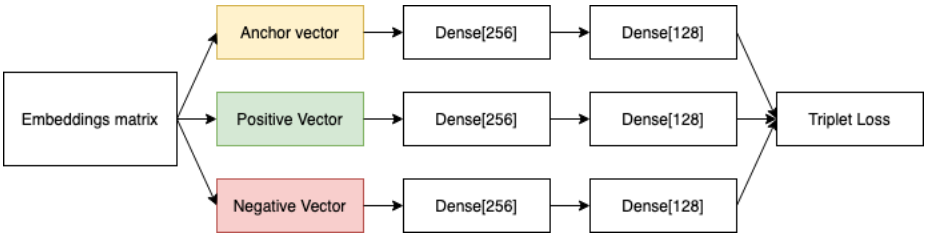


*Рисунок 15. Архитектура DSSM сети*

Сеть обучалась с помощью оптимизатора Adam, с шагом обучения 1e-3 и базовыми гиперпараметрами оптимизатора. Обучение сети происходило на GPU.

Во время обучения измерялось значение ошибки на тестовой и валидационной выборке, а также рассчитывалась метрика полноты относительно данной задачи на валидационном датасете – $\mathrm{D(A,P)} < \mathrm{D(A,N)}$, D – евклидова метрика расстояния.

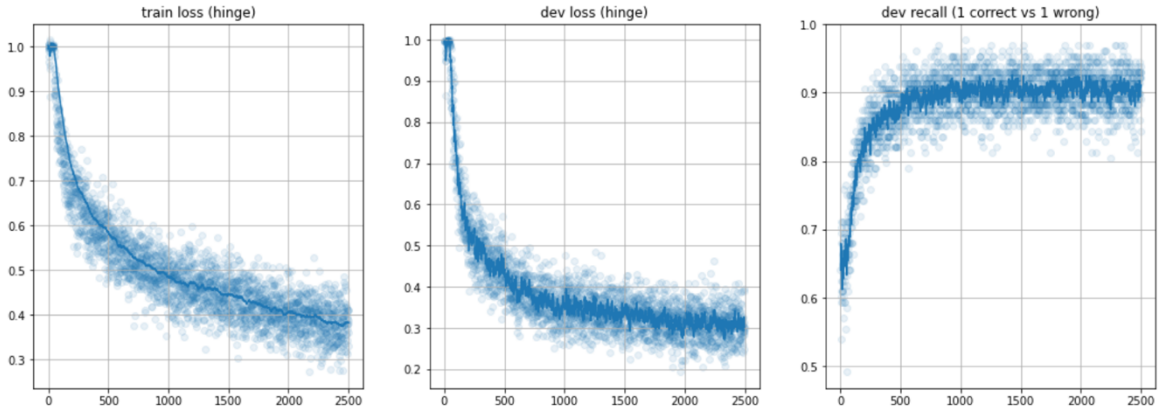


*Рисунок 16. Процесс обучения DSSM модели*

# ОПРЕДЕЛЕНИЕ ПУТИ В ГРАФЕ И МЕХАНИЗМ РАЗБОРА ПРЕДЛОЖЕНИЯ

Локализация области в графе, в которой содержится ответ – это первый шаг к определению SPARQL шаблона для вопроса на естественном языке. Следующий шаг – определение SPARQL шаблона.

SPARQL шаблон – это путь в графе знаний, который позволяет на основании данных указанных в вопросе прийти к сущности которая является ответом на вопрос. Определение подграфа позволяет точно определить набор возможных SPARQL запросов, которые могут привести к ответу. Так как при добавлении нового домена собирается достаточно маскированных формулировок вопроса, этот датасет можно использовать как базу знаний маскированных вопросов, для которых уже известно какому SPARQL запросу они уже соответствуют.

Процесс поиска базового SPARQL шаблона выглядит так:

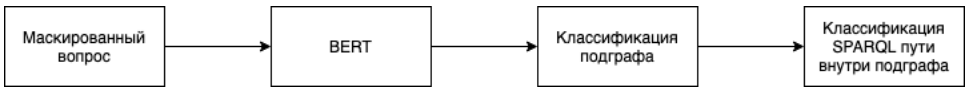


*Рисунок 17. Алгоритм поиска SPARQL запроса*

Рассмотрим последний шаг алгоритма. Есть база знаний вопросов и соответствующий им SPARQL запрос. Внутри каждого подграфа есть N SPARQL запросов. Входной запрос классифицируется в подграф, далее определяется ближайший вопрос внутри данного подграфа, к которому уже известен SPARQL запрос. В качестве метрики близости используется евклидова дистанция (10).

Алгоритм поиска ближайшего соседа – реализация алгоритма приближенного поиска в FAISS. Алгоритм поиска в FAISS был описан в главе с нечетким поиском сущностей графа.

SPARQL шаблон позволяет определить путь в графе, который приведет к нужному ответу. Но часто вопросы не просто требуют сущность в ответ на вопрос, а например количество таких сущностей, значение предиката

указанной в вопросе сущности или сортировка сущностей на основании деталей сущности. Примеры вопросов:

- Кто продюсер фильма Титаник? – Предикат вопроса – продюсер
- Сколько альбомов выпустил Эминем? – Уточнение вопроса – подсчет количества сущностей.
- Последний фильм Тарантино? – Сортировка фильмов по убыванию года производства

Разбор вопроса происходит уже на том уровне, когда есть информация что за SPARQL шаблон соответствует данному вопросу и в какой подграф соответствует данному запросу. Такая информация позволяет разобрать входной вопрос с помощью контекстно-свободных грамматик и регулярных выражений.

Также для каждого подграфа определены правила, которые указывают порядок поиска именованных сущностей вопроса. Например, если в подграф попал вопрос с двумя сущностями – FILM и CHAR (пример – *Кто сыграл Анри Фортена в фильме Отверженные?*), то необходимо сначала найти все FILM(Отверженные), а потом найти все смежные CHAR узлы графа, и найти нужную сущность CHAR (*Анри Фортен*) уже среди этих сущностей.

Инструмент для работы с контекстно-свободными грамматиками, который применяется в данной работе – Yargy. С помощью Yargy возможно провести разбор предложения и извлечь оттуда ключевые слова. Для каждого типа SPARQL есть свой обработчик на контекстно-свободных грамматиках (КСГ). Так как используемые правила для разбора предложения повторяются, в данной работе реализован базовый класс с основными правилами на КСГ, которые переиспользуются для новых SPARQL запросов.

# МЕТОДОЛОГИЯ ДОБАВЛЕНИЯ НОВОГО ДОМЕНА В ГРАФ ЗНАНИЙ

Один из результатов, представленных в этой работе – реализованный инструмент по созданию доменных вопросно-ответных систем. На основании него в системе было создано 2 вопросно-ответных системы по фильмам и музыки.

Рассмотрим методологию добавления нового домена в данную систему:

1. Сбор данных
    a. Сбор необходимых вопросов к домену на краудсорс платформе. Пример задания - *Представьте, что вы разговариваете с виртуальным ассистентом. Спросите его что-либо про кино.*
    b. Анализ вопросов:
        i. Выделение возможных типов вопросов к системе.
        ii. Выделение основных типов сущностей вопроса.
        iii. Выделение основных типов связей в вопросе.
    c. Сбор тренировочных датасетов для извлечения именованных сущностей, для классификации подграфов и для классификации домена.
2. Формирование графа или его обновление
    a. Создание SPARQL запросов для выделенных типов вопросов, которые извлекают подграфы из базового графа знаний. Результат исполненных SPARQL сохраняется в RDF файлы и загружается в графовую базу данных.
3. Обновление контента программы
    a. Данные, собранные на первом этапе, используются для переобучения моделей программы:
        i. Классификатор домена
        ii. Классификатор SPARQL шаблона
        iii. Обучение моделей NER
        iv. Классификатор подграфа

v. Доработка моделей на КСГ

b. Создание коллекций сущностей из сущностей графа, подбор гиперпараметров для высокого качества нечеткого поиска в них.

Представим данную методологию на схеме.

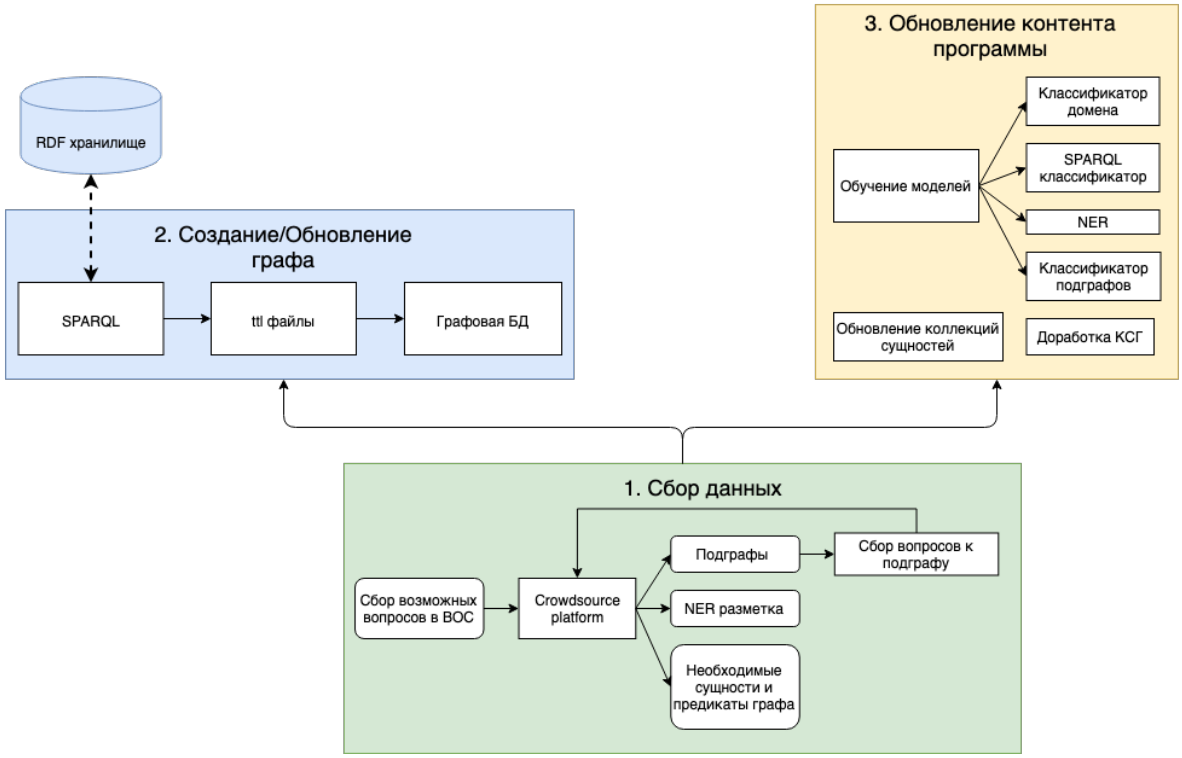


*Рисунок 18. Процесс добавления нового домена в систему*

## СРАВНЕНИЕ С ДРУГИМИ ВОПРОСНО ОТВЕТНЫМИ СИСТЕМАМИ

В данной части представлено сравнение с другими вопросно-ответными системами, построенными на графах знаний. В анализе рассмотрено 2 системы – DeepPavlov KBQA и AquaDR. DeepPavlov KBQA (Burtsev, 2018) – русскоязычная система, QAnswer (Diefenbach, 2017) – англоязычная. Обе системы строились на основании графа знаний WikiData. Для оценки использовались русскоязычные вопросы и их переведенные на английский версии. Оценивалось ответное качество на вопросы из двух доменов – кино и музыка. На краудсорс платформе было набрано 250 вопросов по каждому домену и переведено на русский язык. Было проверено, что все сущности присутствующие в вопросе, есть в графе знаний WikiData.

Так как все оцениваемые системы комплексные, еще оценивалось корректность определенного типа вопроса. Например, вопрос был про дату, и система дала неверный ответ на вопрос, но верно поняла тип вопроса и возвратила дату. Это связано с тем, что определение верного идентификатора узла зависит от самого графа знаний, от алгоритма выделения сущностей и от алгоритма нечеткого поиска. Так как эти элементы системы сугубо индивидуальны от системы к системе, стоит добавить еще один тип оценки системы, которая не включает эти компоненты.

| | Ответное качество | Качество определения типа вопроса |
|---|---|---|
| KGQA | 0.59 | 0.74 |
| Deep Pavlov KBQA | 0.15 | 0.35 |
| QAnswer | 0.20 | 0.38 |

*Таблица 4. Сравнение KGQA с другими вопросно-ответными системами на основании графов знаний*

Высокое ответное качество представленной в работе системы объясняется следующими факторами:

- Цель данной работы – разработать вопросно-ответную систему для различных доменных областей знаний, в то время как двух других –

решить задачу ODQA с помощью базы знаний WikiData. Ввиду этого, в представленной работе гораздо лучше проработаны возможные вопросы к исследуемым двум доменам.

- Общее достаточно высокое качество определения типа вопроса всех систем обуславливается тем, что много вопросов (около 40% вопросов валидационной выборки) были простыми вопросами, где не был известен субъект или объект триплета.
- Deep Pavlov KBQA и QAnswer не рассматривают возможную связность двух сущностей в вопросе. В KGQA системе есть функциональность, которая в процессе разбора вопроса позволяет узнать смежные узлы сущности, указанной в вопросе и осуществлять нечеткий поиск уже среди смежных узлов, а не в независимом друг от друга порядке.
- Тренировочные датасеты для данной работы собирались таким же образом, как и валидационный датасет для оценки системы. Возможно, этот факт тоже позволил KGQA системе ответить на большее количество заданных вопросов.

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной магистерской работы было выполнено следующее:

- Разработан инструмент по созданию вопросно-ответной системы на основании графов знаний для новых доменов
- Разработаны и внедрены вопросно ответные системы для доменов фильмов и музыки
- Реализован алгоритм быстрого добавления новых доменов в систему без переобучения модели
- Разработана методология по добавлению новых доменов в вопросно отвернут систему

Работа выполнена с учетом требований и ограничений, указанных в главе. В будущем планируется усовершенствования процесса добавления новых доменов в систему, пополнение системы новыми доменами и реализация диалогового менеджера для возможности уточнения вопроса.

# СПИСОК ЛИТЕРАТУРЫ


1. Adel Tahri, Okba Tibermacine (2013) DBPedia based factoid question answering system // International Journal of Web & Semantic Technology (IJWesT) Vol.4, No.3
2. Christina Unger, André Freitas, Philipp Cimiano (2014) An introduction to Question Answering over Linked Data // In Proceedings of the 2014 Reasoning Web Summer School
3. William W. Cohen, Pradeep Ravikumar, Stephen F. Fienberg (2003) A comparison of string distance metrics for Name-Matching Tasks // IIWeb. – 2003. – T. 2003. – C. 73-78.
4. Daniel Hernández, Aidan Hogan, Markus Krötzsch (2015) Reifying RDF: What Works Well With Wikidata, ISWC, USA, pp. 32-47
5. Daniel Jurafsky, James H.Martin (2018) Speech and Language Processing. 558 c.
6. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018) Bert: Pretraining of deep bidirectional transformers for language understanding, Available at: arXiv:1810.04805
7. Felix Conrads , Jens Lehmann , Muhammad Saleem , Mohamed Morsey, Axel-Cyrille Ngonga Ngomo (2017), Iguana: A Generic Framework for Benchmarking the Read-Write Performance of Triple Stores, Available at: https://svn.aksw.org/papers/2017/ISWC_Iguana/public.pdf
8. Florian Schroff, Dmitry Kalenichenko, James Philbin (2015), FaceNet: A Unified Embedding for Face Recognition and Clustering, Available at: arXiv:1503.03832
9. Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. Available at: arXiv:1603.01360.

10. Juan Luis Suárez, Salvador García, Francisco Herrera (2019) A Tutorial on Distance Metric Learning: Mathematical Foundations, Algorithms and Software, Available at: arXiv: 1812.05944
11. Khriyenko Oleksiy, (2019), RDF data Serialization and Storing, Available at: http://users.jyu.fi/~olkhriye/ties4520/lectures/Lecture02.pdf
12. Tommaso Soru, Edgard Marx, André Valdestilhas, Diego Esteves, Diego Moussallem, Gustavo Publio (2018) Neural Machine Translation for Query Construction and Composition, Available at: arXiv:1806.10478
13. Wei Shen, Jianyong Wang, Jiawei Han (2014) Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions, IEEE, Vol. 27, pp. 443-460
14. Yuqing Gao, Jisheng Liang, Benjamin Han, Mohamed Yakout, Ahmed Mohamed (2018) Building a Large-scale, Accurate and Fresh Knowledge Graph, KDD, UK, available at: https://kdd2018tutorialt39.azurewebsites.net/KDD%20Tutorial%20T39.pdf
15. Burtsev M. et al. DeepPavlov: Open-Source Library for Dialogue Systems //Proceedings of ACL 2018, System Demonstrations. – 2018. – С. 122-127.
16. Diefenbach D., Singh K., Maret P. Wdaqua-core0: A question answering component for the research community //Semantic Web Evaluation Challenge. – Springer, Cham, 2017. – С. 84-89.
17. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs //IEEE Transactions on Big Data. – 2019.
18. Chen D. et al. Reading wikipedia to answer open-domain questions //arXiv preprint arXiv:1704.00051. – 2017.
19. Soru T. et al. SPARQL as a Foreign Language //arXiv preprint arXiv:1708.07624. – 2017.
20. Hwang W. et al. A comprehensive exploration on wikisql with table-aware word contextualization //arXiv preprint arXiv:1902.01069. – 2019.
21. А. Л. Доброхотов. // Новая философская энциклопедия, 2003.

22. Decker S. et al. The semantic web: The roles of XML and RDF //IEEE Internet computing. – 2000. – Т. 4. – №. 5. – С. 63-73.
23. Vrandečić D., Krötzsch M. Wikidata: a free collaborative knowledgebase //Communications of the ACM. – 2014. – Т. 57. – №. 10. – С. 78-85.
24. Jena A. semantic web framework for Java. – 2007.
25. Erling O. Virtuoso, a Hybrid RDBMS/Graph Column Store //IEEE Data Eng. Bull. – 2012. – Т. 35. – №. 1. – С. 3-8.
26. SYSTAP L. BlazeGraph. – 2015.
27. Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, Angel Kirilov. KIM – a semantic platform for information extraction and retrieval. In Natural language engineering, Volume 10, Issue 3-4, pp. 375-392. Cambridge University Press. September 2004.
28. Auer S. et al. Dbpedia: A nucleus for a web of open data //The semantic web. – Springer, Berlin, Heidelberg, 2007. – С. 722-735.
29. Kingma D. P., Ba J. Adam: A method for stochastic optimization //arXiv preprint arXiv:1412.6980. – 2014.
30. Gers F. A., Schmidhuber J., Cummins F. Learning to forget: Continual prediction with LSTM. – 1999.
31. Chung J. et al. Empirical evaluation of gated recurrent neural networks on sequence modeling //arXiv preprint arXiv:1412.3555. – 2014.
32. Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. – 2017. – С. 5998-6008.
33. Maaten L., Hinton G. Visualizing data using t-SNE //Journal of machine learning research. – 2008. – Т. 9. – №. Nov. – С. 2579-2605.
34. Тестелец Я. Г. Введение в общий синтаксис. – Федеральное государственное бюджетное образовательное учреждение высшего образования Российский государственный гуманитарный университет, 2001.
35. Straka M., Hajic J., Straková J. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, pos

tagging and parsing //Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16). – 2016. – С. 4290-4297.

36. Korobov M. Morphological analyzer and generator for Russian and Ukrainian languages //International Conference on Analysis of Images, Social Networks and Texts. – Springer, Cham, 2015. – С. 320-332.
37. Lample G. et al. Neural architectures for named entity recognition //arXiv preprint arXiv:1603.01360. – 2016.
38. Xing E. P. et al. Distance metric learning with application to clustering with side-information //Advances in neural information processing systems. – 2003. – С. 521-528.
39. Huang P. S. et al. Learning deep structured semantic models for web search using clickthrough data //Proceedings of the 22nd ACM international conference on Information & Knowledge Management. – 2013. – С. 2333-2338.