

Институт Политехнический

Высшая школа Кибернетики и цифровых технологий

Направление 09.04.04 Программная инженерия

(шифр, наименование)

Профиль Проектирование и разработка систем искусственного интеллекта

ДОПУСТИТЬ К ЗАЩИТЕ

Руководитель ВШ КЦТ \_\_\_\_\_\_\_\_\_ Давыдов О. А.

подпись ФИО

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

дата

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ**

Тема Разработка программы "Базы Знаний ТОГУ" с использованием полнотекстового поиска с помощью лексем естественного языка

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_             Забавин А.С.

подпись ФИО

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

дата

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_            Вихтенко Э.М.

подпись ФИО

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

дата

Нормоконтролёр \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_             Резак Е. В.

подпись ФИО

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

дата

**Консультанты:**

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

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

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

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

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

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

дата

По \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

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

дата

Хабаровск – 2024 г.

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Тихоокеанский государственный университет»**

Институт Политехнический

Высшая школа Кибернетики и цифровых технологий

Направление 09.04.04 – Программная инженерия

(шифр, наименование)

Профиль Проектирование и разработка систем искусственного интеллекта

УТВЕРЖДАЮ

Руководитель ВШ КЦТ \_\_\_\_\_\_\_\_\_\_ Давыдов О.А.­.

подпись ФИО

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г.

**З А Д А Н И Е**

**на выпускную квалификационную работу**

Студенту Забавину Алексею Сергеевичу

1. Тема Разработка программы "Базы Знаний ТОГУ" с использованием полнотекстового поиска с помощью лексем естественного языка

Утверждена приказом по университету \_\_№ 020/246 \_ от \_25 марта\_\_ 2025 г.

2. Срок сдачи студентом выпускной квалификационной работы \_\_17 июня\_\_ 2025 г.

3. Исходные данные к работе \_

постановка задачи на разработку информационной базы, ее структуры и механизма полнотекстового поиска информации в ней с заданным качеством;

материалы (статьи, публикации в интернете) по теме исследования;

способы поиска на естественном языке, представление знаний для QA-системы, способы измерения качества ответов (ранжирования ответов);

алгоритмы полнотекстового поиска, семантической близости, синтаксического анализа;

материалы преддипломной практики.

4. Перечень подлежащих разработке в выпускной квалификационной работе вопросов

Изучение подходов к хранению информации, реализация базы данных предпочтительной структуры для QA-системы

Изучение и реализация алгоритмов полнотекстового поиска информации, подсчета значимости слова в базе знаний и в документе;

Изучение и реализация алгоритмов определения семантической близости слов в векторном пространстве языка;

Изучение и реализация алгоритмов синтаксического анализа предложения, выделения главных членов предложения передающих основную мысль запроса;

Изучение и реализация выявления «именованных сущностей» в тексте для повышения их значимости в пользовательском запросе;

Разработка программы, проведение экспериментов и исследование эффективности предложенного решения.

5. Перечень графической части (с точным указанием обязательных чертежей), либо раздаточного материала

Раздаточный материал к докладу

Руководитель работы Задание принял к исполнению студент

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

подпись ФИО подпись ФИО

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

дата дата

Реферат

Магистерская диссертация содержит 73 81 страницу текстового документа формата А4, включающего Ошибка: источник перекрёстной ссылки не найден 16 рисунков, Ошибка: источник перекрёстной ссылки не найден 5 таблиц, 33 использованных источников и 16 страницы приложения формата А4.

NLP, ембеддинг, [Word2vec](#__RefHeading___Toc913_2862954754), [PostgreSQL](#__RefHeading___Toc919_2862954754), NER, GloVe, RNN, BERT, векторное представление слов, контекстное окно, корпус текста, лексема.

Цель данного исследования состоит в разработке программного средства, использующего RNN искусственную нейронную сеть семейств Word2vec и GloVe для синтаксического и семантического анализа и оптимизации поисковых запросов.

Объектами исследования являются:

1. хранение информации для QA-системы в базе данных позволяющее решать задачу полнотекстового поиска в ней
2. частотный алгоритм ранжирования результатов поиска в коллекции документов;
3. методы лексического, синтаксического и семантического анализа текста.

Предметом работы является изучение качества поиска в различных вариациях — при простом поиске по вхождению текста, при индексировании на основе «частотной важности» слов в документе, а также используя оптимизации поискового запроса на основе семантической близости и синтаксической важности членов предложения поискового запроса.

Содержание

[Введение 6](#_Toc199194248)

[1 Анализ предметной области 10](#_Toc199194249)

[1.1 Основные понятия 10](#_Toc199194250)

[1.2 Задача обработки естественного языка в компьютерной лингвистике 10](#_Toc199194251)

[1.3 Полнотекстовый поиск 13](#_Toc199194252)

[1.4 Существующие решения полнотекстового поиска 13](#_Toc199194253)

[1.4.1 Реализация полнотекстового поиска в PostgreSQL 15](#_Toc199194254)

[1.5 Метрики релевантности. Оценка качества ранжирования в задаче поиска 16](#_Toc199194255)

[1.6 Существующие модели для NLP обработки текстов 20](#_Toc199194256)

[1.6.1 Рекуррентная нейросетевая языковая модель (RNNLM) 21](#_Toc199194257)

[1.6.2 Word2vec 22](#_Toc199194258)

[1.6.3 GloVe (Global Vectors) 25](#_Toc199194259)

[1.6.4 Transformers 26](#_Toc199194260)

[1.7 Описание концепции «словесных вложений» рассматриваемых моделей, Векторные представления слов 30](#_Toc199194261)

[1.7.1 Простые бинарные OHE ембенддинги 30](#_Toc199194262)

[1.7.2 Частотный эмбеддинг 31](#_Toc199194263)

[1.7.3 Контекстные ембеддинги. Скользящее окно. Word2vec 33](#_Toc199194264)

[1.8 Подведение итогов 37](#_Toc199194265)

[2 Описание разработанных алгоритмов 40](#_Toc199194266)

[2.1 Алгоритм синтаксического анализа запроса, выявление основной части запроса 40](#_Toc199194267)

[2.2 Алгоритм оптимизации по симантической близости и TF-IDF 42](#_Toc199194268)

[3 Описание разработанной программы 45](#_Toc199194269)

[3.1 Назначение программы и выбранный инструментарий 45](#_Toc199194270)

[3.2 Описание интерфейса программы 47](#_Toc199194271)

[3.3 Тестирование качества алгоритма оптимизации запроса 53](#_Toc199194272)

[3.4 Описание технической реализации программы 57](#_Toc199194273)

[Заключение 60](#_Toc199194274)

[Список использованных источников 61](#_Toc199194275)

[Приложение А 65](#_Toc199194276)

[Приложение Б 80](#_Toc199194277)

Введение

Полнотекстовый поиск (англ. Full text searching) — автоматизированный поиск документов, при котором поиск ведётся не по именам документов, а по их содержимому, всему или существенной части [1]. Это технический термин, обозначающий расширенный лингвистический текстовый запрос для базы данных или текстовых документов.

Цель данного исследования состоит в разработке программного средства, использующего RNN искусственную нейронную сеть семейств Word2vec и GloVe для синтаксического и семантического анализа и оптимизации поисковых запросов.

Объектами исследования являются:

1. хранение информации для QA-системы (Вопрос-Ответ) в базе данных позволяющее решать задачу полнотекстового поиска в ней
2. частотный алгоритм ранжирования результатов поиска в коллекции документов;
3. методы лексического, синтаксического и семантического анализа текста.

Предметом работы является изучение качества поиска в различных вариациях — при простом поиске по вхождению текста, при индексировании на основе «частотной важности» слов в документе, а также используя оптимизации поискового запроса на основе семантической близости и синтаксической важности членов предложения поискового запроса.

Актуальность продиктована необходимостью разработки автоматического ассистента службы поддержки ТОГУ, который бы с большей релевантностью предоставлял пользователю ответы на поставленные вопросы. Существующие сторонние разработки менее ориентированы на контекст работы и информационную специфику вуза и требуют больших усилий по интеграции.

В область науки об искусственном интеллекте входит такой раздел как «Инженерия знаний», которая отвечает за особенности построения экспертных систем и баз знаний для хранения информации. Этот раздел изучает средства извлечения, представления, структурирования и использования знаний до программной реализации компонентов системы. Сам термин был введен Альбертом Фейгенбаумом и Памелой МакКордак в 1983 как «Инженерия знаний — раздел (дисциплина) инженерии, направленный на внедрение знаний в компьютерные системы для решения сложных задач, обычно требующих богатого человеческого опыта». Как видно из этого определения ИЗ тесно соприкасается с разработкой программного обеспечения и используется во многих информационных исследованиях, например, таких, как исследования искусственного интеллекта, включая базы знаний, сбор данных, экспертные системы, системы поддержки принятия решений и географические информационные системы [2].

База знаний (БЗ) — база данных, содержащая правила вывода и информацию о человеческом опыте и знаниях в некоторой предметной области. В самообучающихся системах база знаний также содержит информацию, являющуюся результатом решения предыдущих задач [3].

Современные базы знаний работают совместно с системами поиска и извлечения информации. Для этого требуется некоторая модель классификации понятий и определённый формат представления знаний. Иерархический способ представления в базе знаний набора понятий и их связей называется онтологией.

Онтологию некоторой области знаний вместе со сведениями о свойствах конкретных объектов часто называют «базой знаний». Вместе с тем полноценные базы знаний [4] содержат в себе не только фактическую информацию, но и правила вывода, позволяющие делать автоматические умозаключения об уже имеющихся или вновь вводимых фактах и тем самым производить семантическую (осмысленную) обработку информации [5].

В данной магистерской работе большее внимание уделено возможности базы знаний иметь некоторую семантическую большую насыщенность чем обычная база данных за счет использования расширяющих функционал поиска технологий, и поэтому мы будем рассматривать в большей мере задачу информационного поиска чем механизм дообучения БЗ. В своей работе я исследую возможности существующих технологий и инструментария для построения расширенной системы хранения информации, организованной в виде QA (Вопрос-Ответ) справочника с расширенной системой полнотекстового поиска, а также использования NLP (Обработка Есстественного Языка) техник работы с текстами для повышения качества ранжирование релевантных результатов машиной полнотекстового поиска (Postgres Full-Text Search Engine)

Для достижения установленной цели были поставлены следующие задачи:

1. изучение на основе литературных данных существующих подходов к хранению информации QA-систем оценки их эффективности;
2. изучение на основе литературных данных существующих решений организации полнотекстового поиска и ранжирования результатов по частоте встречаемости слов
3. изучение на основе литературных данных существующих методов и моделей нейронных сетей для задачи определения семантической близости слов;
4. изучение на основе литературных данных существующих моделей для проведения синтаксического анализа предложений естественного языка;
5. изучение на основе литературных данных существующих моделей определения именованных сущностей (NER)
6. программная реализация базы данных с использованием статистической (Байесовской) индексации документов
7. программная реализация алгоритма, осуществляющего определение семантически близких лексем к лексемам в индексе базы знаний
8. программная реализация алгоритма синтаксического анализа для определения главных токенов пользовательского запроса и отбрасывания второстепенных
9. экспериментальная оценка эффективности применения вышеуказанных алгоритмов к задаче «информационного поиска», определения метрик релевантности ответа вопросу

Методология и методы исследования. В работе использована теория «словесных вложений» или эмбеддингов — что в свою очередь термины, следующие из математического подхода к представлению языка, векторному пространству языка. Так же в работе используются теория релевантности по определению качества ранжирования элементов для задачи информационного поиска. Все программы разработаны на языке программирования Python 3.11.

Исследование имеет большую теоретическую и практическую значимость.

Предложенный метод оптимизации пользовательского запроса улучшает семантическую емкость базы вопросов и ответов в тоже время позволяет не прибегать к тяжелым генеративным моделям при поиске заранее известной информации. Существующий ранее механизм поиска по документам в ИС ТОГУ не обеспечивает современный уровень качества работы пользователя с программой на естественном языке. Разработанный и исследованный на практике метод оптимизации поискового запроса помимо улучшения среднего ранка ответов, так же в будущих разработках может быть использован для получения качественного RAG контекста для современных генеративных LLM моделей. Метод использует обученные RNN модели семейства Word2Vec для синтаксического и семантического анализа токенов запроса.

Основные положения, выносимые на защиту:

1. Использование полнотекстового поиска позволяет эффективнее индексировать и искать информацию в документах текстового типа,
2. Настройка PostgreSQL tsvector позволяет индексировать как отдельные колонки, так и группы колонок реляционной базы с разного весомо ранжирования;
3. Применение синтаксического и семантического анализа пользовательского запроса позволило улучшить поисковую выдачу путем оптимизации исходного запроса;
4. Разработанный метод оптимизации запроса позволяет поднять качество релевантных ответов в два раза перед полнотекстовым поиском без модификаций;
5. Применение небольших Word2Vec моделей оправдано для создания QA системы базы знаний;

# Анализ предметной области

## Основные понятия

Для качественного решения задачи информационного поиска стоит начать анализ предметной области с такого понятия как «Язык». Собственно, сущность на котором кодируется базовые понятия окружающего мира, их взаимосвязи и представления знаний. Язык — сложная знаковая система, естественно или искусственно созданная и соотносящая понятийное содержание, и типовое звучание (написание). Термин «язык», понимаемый в широком смысле, может применяться к произвольным знаковым системам, хотя чаще он используется для более узких классов знаковых систем. Языки изучает лингвистика (языкознание). Знаковые системы — предмет изучения семиотики. Влияние структуры языка на человеческое мышление и поведение изучается психолингвистикой. Философия языка занимается общечеловеческими теоретическими проблемами, связанными с языком и с понятием языка [6]. Исходя из всех этих особенностей языка как предмета взаимодействия с пользователем в задаче информационного поиска и появляются дальнейшие техники анализа элементов языка, реализуемые в программе.

## Задача обработки естественного языка в компьютерной лингвистике

Компьютерная лингвистика (также: математическая или вычислительная лингвистика, англ. computational linguistics) — научное направление в области математического и компьютерного моделирования интеллектуальных процессов у человека и животных при создании систем искусственного интеллекта, которое ставит своей целью использование математических моделей для описания естественных языков. Компьютерная лингвистика частично пересекается с обработкой естественных языков. В последней акцент делается не на абстрактные модели, а на прикладные методы описания и обработки языка для компьютерных систем. Полем деятельности компьютерных лингвистов является разработка алгоритмов и прикладных программ для обработки языковой информации.

Математическая лингвистика является ветвью науки искусственного интеллекта. Её история началась в Соединённых Штатах Америки в 1950-х годах. В апреле 1959 года в Ленинграде состоялось I Всесоюзное совещание по математической лингвистике, созванное Ленинградским университетом и комитетом прикладной лингвистики. Главным организатором Совещания был Н. Д. Андреев. В Совещании приняли участие ряд видных математиков, в частности, С. Л. Соболев, Л. В. Канторович (впоследствии — Нобелевский лауреат) и А. А. Марков (последние двое выступали в прениях). В. Ю. Розенцвейг выступил в день открытия Совещания с программным докладом «Общая лингвистическая теория перевода и математическая лингвистика» [7].

Направления компьютерной лингвистики:

* Обработка естественного языка (англ. NLP - natural language processing). Уровни обработки и анализа текста: синтаксический, морфологический, семантический.
* Корпусная лингвистика, создание и использование электронных корпусов текстов.
* Создание электронных словарей, тезаурусов, онтологий.
* Автоматический перевод текстов.
* Автоматическое извлечение фактов из текста (извлечение информации) (англ. fact extraction, text mining).
* Автореферирование (англ. automatic text summarization).
* Построение систем управления знаниями. Экспертные системы.
* Создание вопросно-ответных систем (англ. question answering systems).
* Оптическое распознавание символов (англ. OCR).
* Автоматическое распознавание речи (англ. ASR).
* Автоматический синтез речи.

Как видно из направлений компьютерной лингвистики, для построения поисковой машины для знаний в широком смысле (текстовая, графическая, видео, аудио и т. д. Информация) в современном мире используются множество подходов и их комбинации: например, «Поиск по картинке» в поисковых системах Google, Yandex, суммаризация «Краткий пересказ» текстов или даже видео.

Остановимся на теории обработки естественного языка применительно к текстовой информации как часть актуальной разработанной программы. Обработка естественного языка изучает проблемы компьютерного анализа и синтеза текстов на естественных языках. Применительно к искусственному интеллекту анализ означает понимание языка, а синтез — генерацию грамотного текста. Понимание естественного языка иногда считают AI-полной задачей, потому как распознавание живого языка требует огромных знаний системы об окружающем мире и возможности с ним взаимодействовать. Само определение смысла слова «понимать» — одна из главных задач искусственного интеллекта. Качество понимания зависит от множества факторов: от языка, от национальной культуры, от самого собеседника и т. д.

Сложности в анализе естественного языка

Сложности с раскрытием анафор (распознаванием, что имеется в виду при использовании местоимений): предложения «Мы отдали бананы обезьянам, потому что они были голодные» и «Мы отдали бананы обезьянам, потому что они были перезрелые» похожи по синтаксической структуре. В одном из них местоимение они относится к обезьянам, а в другом — к бананам. Правильное понимание зависит от знаний компьютера, какими могут быть бананы и обезьяны.

Свободный порядок слов может привести к совершенно иному толкованию фразы: «Бытие определяет сознание» — что определяет что?

В русском языке свободный порядок компенсируется развитой морфологией, служебными словами и знаками препинания, но в большинстве случаев для компьютера это представляет дополнительную проблему.

В речи могут встретиться неологизмы, например, глагол «Загугли» — то есть поищи информации в Google. Система должна уметь отличать такие случаи от опечаток и правильно их понимать.

Правильное понимание омонимов — ещё одна проблема. При распознавании речи, помимо прочих, возникает проблема фонетических омонимов. Во фразе «Серый волк в глухом лесу встретил рыжую лису» выделенные слова слышатся одинаково, и без знания, кто глухой, а кто рыжий, не обойтись (кроме того, что лиса может быть рыжей, а лес — глухим, лес также может быть рыжим (характеристика, в данном случае обозначающая преобладающий цвет листвы в лесу), в то время как лиса может быть глухой, что порождает дополнительную проблему, вытекающую из предыдущей, хотя и отчасти компенсируется морфологией — у прилагательных в данном предложении род явно разный).

## Полнотекстовый поиск

Полнотекстовый поиск (англ. Full text searching) — автоматизированный поиск документов, при котором поиск ведётся не по именам документов, а по их содержимому, всему или существенной части [8]. Это технический термин, обозначающий расширенный лингвистический текстовый запрос для базы данных или текстовых документов. Поисковая система проверяет все слова, хранящиеся в документе, пытаясь соответствовать определенным критериям поиска, заданным пользователем [9].

Первые версии программ полнотекстового поиска предполагали сканирование всего содержимого всех документов в поиске заданного слова или фразы. При использовании такой технологии поиск занимал очень много времени (в зависимости от размера базы), а в интернете был бы невыполним. Современные алгоритмы заранее формируют для поиска так называемый полнотекстовый индекс — словарь, в котором перечислены все слова и указано, в каких местах они встречаются. При наличии такого индекса достаточно осуществить поиск нужных слов в нём и тогда сразу же будет получен список документов, в которых они встречаются.

## Существующие решения полнотекстового поиска

Существует множество решений для организации полнотекстового поиска. Большинство основывается на морфологии (нормализация слов и использование словаря лексем для составления индексов документов). Некоторые в дополнение к этому используют и концепции словесных вложений (embeddings) описаныне ранее для организации семантического поиска в векторном пространстве LLM модели работающей совместно с поисковой машиной (ElasticSearch). Вот некоторые из них:

LunrJS - это библиотека JavaScript, предназначенная для работы с браузером и сервером. Он не требует каких-либо внешних зависимостей или каких-либо дополнительных услуг. Он поставляется с процессорами на нескольких языках, которые можно настраивать в соответствии с потребностями пользователя. Это легкая альтернативная библиотека для Apache Solr. Lunr поддерживает нестандартные 14 языков и предлагает нечеткие термины.

Apache Solr - будучи платформой корпоративного уровня, Solr оснащен такими функциями, как запросы балансировки нагрузки, автоматизированные функции, централизованная конфигурация, распределенная мгновенная индексация и готовая к масштабированию инфраструктура. Solr используется несколькими крупными игроками, такими как DuckDuckGo, AT&T, Instagram, eBey, Comcast, Magento eCommerce, Adobe, Netflix, Internet Archive и другими.

Sphinx - это полнотекстовый сервер поисковых систем, написанный на C++ для достижения наилучшей производительности. Он работает бесперебойно в Windows, Linux, macOS. Он индексирует все данные в базе данных SQL или NoSQL. Он включает в себя индексацию базы данных SQL/NoSQL, поиск нетекстовых атрибутов, полнотекстовую индексацию в реальном времени и поддерживает распределенный поиск.

Manticore Search - это многоязычный полнотекстовый поиск с поддержкой наборов больших данных и потоковой передачи данных в реальном времени. Это лучший проект в этом списке, который предлагает уникальные функции, такие как геопоиск, репликации, алгоритмы ранжирования поиска, индексация в реальном времени и встроенная поддержка JSON. Поиск Manticore обеспечивает поддержку индексирования MySQL, PostgreSQL и плоских файлов, таких как CSV, TSV, а также файлов разметки.

Apache Lucene - это полнофункциональная библиотека текстовых поисковых систем. Он легко масштабируется с индексированием текста в реальном времени и низкими требованиями к оборудованию. Его функции включают в себя: ранжирование поиска (в пользу лучших результатов), десятки типов поисковых запросов, поиск по полю, несколько стратегий индексирования, несколько моделей ранжирования и настраиваемые системы хранения данных.

ElasticSearch - это популярный полнотекстовый поиск корпоративного уровня с открытым исходным кодом. Он имеет REST-API и поддерживает индексацию и масштабирование в реальном времени.

Кроме того, возможности полнотекстового поиска реализованы и в СУБД PostgreSQL. Учитывая доступность базы в России, ее популярность и наличие в ТОГУ инфраструктуры для нее я остановился на использовании именно ее для реализации полнотекстового поиска в базе знаний. Кроме того, использование PostgreSQL в качестве и базы данных и движка FTS имеет преимущество в накладных расходах места на диске (популярные движки FTS ориентируются на реплицирование существующих БД), и соответственно задержки в репликации могут привести что выдача может содержать не актуальные данные.

### Реализация полнотекстового поиска в PostgreSQL

В двух словах, полнотекстовый поиск реализуется за счёт индексации слов, содержащихся в документе, и связывания этих проиндексированных слов со ссылками на документ. Поиск по запросу с поддержкой логических операций, используя операторы and, or, not и скобки, впоследствии может сопоставляться с индексом для определения документа, содержащего слова из этого запроса. Очевидно, что индексирование каждого отдельного слова в документе приведёт к образованию очень большого индекса [1].

Для полнотекстового поиска в PostgreSQL предусмотрено несколько специальных типов. Тип tsvector представляет собой что-то вроде нормализованной строки (Лексема), по которой будет производиться поиск. Под нормализацией понимается выкидывание стоп-слов, таких, как предлоги, обрезание окончаний слов (Стемминг), и так далее [10]. Успешно нормализованное слово называется лексемой. Лексемой в лингвистике называется абстрактная единица морфологического анализа, которая соответствует набору форм, принимаемых одним словом [11].

Нормализация и исключение стоп-слов не только улучшает качество поиска, но и уменьшает размер представления документа в формате tsvector, и, как следствие, увеличивает быстродействие. Нормализация не всегда имеет лингвистический смысл, обычно она зависит от требований приложения [12].

Несколько примеров нормализации:

* Лингвистическая нормализация — словари Ispell пытаются свести слова на входе к нормализованной форме, а стеммеры убирают окончания слов
* Адреса URL могут быть канонизированы, чтобы, например, следующие адреса считались одинаковыми: http://www.pgsql.ru/db/mw/index.html http://www.pgsql.ru/db/mw/ http://www.pgsql.ru/db/../db/mw/index.html
* Названия цветов могут быть заменены их шестнадцатеричными значениями, например, red, green, blue, magenta -> FF0000, 00FF00, 0000FF, FF00FF
* При индексировании чисел можно отбросить цифры в дробной части для сокращения множества всевозможных чисел, чтобы, например, 3.14159265359, 3.1415926 и 3.14 стали одинаковыми после нормализации, при которой после точки останутся только две цифры.

## Метрики релевантности. Оценка качества ранжирования в задаче поиска

Есть специальное направление в машинном обучении, которое занимается изучением алгоритмов ранжирования способных самообучаться — обучение ранжированию (learning to rank). Чтобы выбрать из всего многообразия алгоритмов и подходов наилучший, необходимо уметь оценивать их качество количественно.

Ранжирование — задача сортировки набора элементов из соображения их релевантности. Чаще всего релевантность понимается по отношению к некому объекту. В задаче информационного поиска объект — это запрос, элементы — всевозможные документы (ссылки на них), а релевантность — соответствие документа запросу, а в задаче, например, рекомендаций же объект — это пользователь, элементы — тот или иной рекомендуемый контент (товары, видео, музыка), а релевантность — вероятность того, что пользователь воспользуется (купит/лайкнет/просмотрит) данным контентом.

Формально, рассмотрим *N* объектов и *M* элементов . Реузальтат работы алгоритма ранжирования элементов *E* для объекта — это отображение , которое сопоставляет каждому элементу вес , характеризующей степень релевантности элемента объекту (чем больше вес, тем релевантнее объект). При этом, набор весов задает перестановку на наборе элементов элементов *E* (считаем, что множество элементов упорядоченное) исходя из их сортировки по убыванию веса .

Чтобы оценить качество ранжирования, необходимо иметь некоторый «эталон», с которым можно было бы сравнить результаты алгоритма. Рассмотрим — эталонную функцию релевантности, характеризующую «настоящую» релевантность элементов для данного объекта ( — элемент идеально подходит, — полностью нерелевантен), а так же соответствующую ей перестановку (по убыванию ).

Существует два основных способа получения :

1. На основе исторических данных. Например, в случае рекомендаций контента, можно взять просмотры (лайки, покупки) пользователя и присвоить просмотренным весам соответствующих элементов 1 (), а всем остальным — 0.

2. На основе экспертной оценки. Например, в задаче поиска, для каждого запроса можно привлечь команду асессоров, которые вручную оценят релевантности документов запросу.

Стоит отметить, что когда принимает только экстремальные значения: 0 и 1, то перестановку обычно не рассматривают и учитывают лишь множество релевантных элементов, для которых .

Цель метрики качества ранжирования — определить, насколько полученные алгоритмом оценки релевантности и соответствующая им перестановка соответствуют истинным значениям релевантности . Рассмотрим основные метрики.

Mean average precision at K (map@K) — одна из наиболее часто используемых метрик качества ранжирования. Чтобы разобраться в том, как она работает начнем с «основ».

Замечание: "precision" метрики используется в бинарных задачах, где inline\_formula принимает только два значения: 0 и 1.

Precision at K (p@K) — точность на K элементах — базовая метрика качества ранжирования для одного объекта. Допустим, наш алгоритм ранжирования выдал оценки релевантности для каждого элемента . Отобрав среди них первые *K ≤ M* элементов с наибольшим можно посчитать долю релевантных. Именно это и делает precision at K:

|  |  |
| --- | --- |
|  | (1) |

Average precision at K

Precision at K — метрика простая для понимания и реализации, но имеет важный недостаток — она не учитывает порядок элементов в «топе». Так, если из десяти элементов мы угадали только один, то не важно на каком месте он был: на первом, или на последнем, — в любом случае *p@10=0.1*. При этом очевидно, что первый вариант гораздо лучше.

Этот недостаток нивелирует метрика ранжирования average precision at K (ap@K), которая равна сумме p@k по индексам k от 1 до K только для релевантных элементов, деленному на K:

|  |  |
| --- | --- |
|  | (2) |

Так, если из трех элементов мы релевантным оказался только находящийся на последнем месте, то , если угадали лишь тот, что был на первом месте, то , а если угаданы были все, то .

Теперь можно посчитать map@K.

Mean average precision at K (map@K) — одна из наиболее часто используемых метрик качества ранжирования. В p@K и ap@K качество ранжирования оценивается для отдельно взятого объекта (пользователя, поискового запроса). На практике объектов множество: мы имеем дело с сотнями тысяч пользователей, миллионами поисковых запросов и т.д. Идея map@K заключается в том, чтобы посчитать ap@K для каждого объекта и усреднить [13]:

|  |  |
| --- | --- |
|  | (3) |

Так же существуют еще метрики на основе понятия "совокупной прибыли" (Normalized discounted cumulative gain (nDCG)), однако она применяется, когда релевантность элементов задается не дискретными 0 и 1 а плавными значениями.

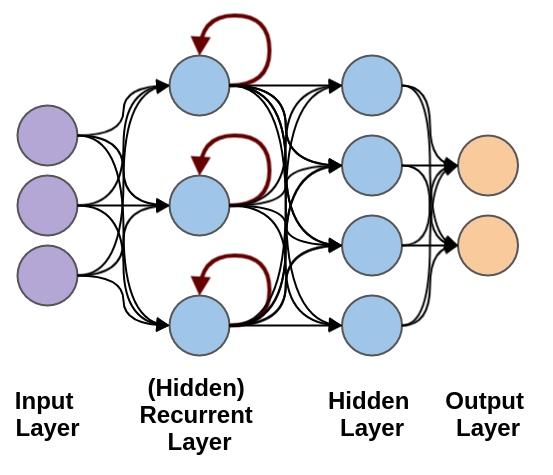
Отдельно стоит выделить метрики качества ранжирования, основанные на одном из коэффициентов ранговой корреляции. В статистике, ранговый коэффициент корреляции — это коэффициент корреляции, который учитывает не сами значения, а лишь их ранг (порядок). Два наиболее распространенных ранговых коэффициента корреляции: коэффициенты Спирмена (является ничем иным как корреляции Пирсона, посчитанной на значениях рангов) и Кендэлла (основан на подсчете согласованных (и несогласованных) пар у перестановок).

Для рекомендательных систем где есть обратная связь с пользователем, просматривающим рекомендации обычно используют метрики, основанные на каскадной модели поведения. Подобные модели поведения пользователя, где изучение предложенных ему элементов происходит последовательно и вероятность просмотра элемента зависит от релевантности предыдущих, называются каскадными. Примеры таких метрик Expected reciprocal rank (ERR) и разработанная Яндексом PFound.

Однако для рассматриваемой задачи создания базы знаний где релевантность определяется дескретно - наиболее применимы метрики, описанные в начале раздела основанные на точности на k-элементах (map@K).

## Существующие модели для NLP обработки текстов

Свои истоки архитектуры моделей для обработки естественного языка ведут из рекуррентных нейронных сетей (RNN). Рекуррентные нейронные сети (RNN) — это нейронные сети с направленными связями между элементами. Выход нейрона может снова подаваться на вход. Такая структура позволяет иметь подобие «памяти» и обрабатывать последовательности данных, например, тексты естественного языка.

Рисунок 1. Схема рекуррентной нейронной сети с двумя скрытыми слоями

Оригинальная вероятностная языковая модель нейронной сети прямого распространения (NNLM) состоит из входного, проекционного, скрытого и выходного слоёв. На входном слое N предыдущих слов кодируются с помощью кодирования 1-из-V, где V — размер словаря. Архитектура NNLM усложняется для вычислений между проекционным и скрытым слоями. При распространённом выборе N = 10 размер проецируемого слоя (P) может составлять от 500 до 2000 (в зависимости от количества измерений), а размер скрытого слоя H обычно составляет от 500 до 1000 единиц [14].

### Рекуррентная нейросетевая языковая модель (RNNLM)

Возникшая в 2001 г. идея привела к рождению одной из первых embedding-моделей NLP. Модель принимает на вход векторные представления n предыдущих слов и может «понимать» семантику предложения. Обучение модели базируется на алгоритме непрерывного мешка слов. Контекстные (соседние) слова подаются на вход нейронной сети, которая предсказывает центральное слово. Сжатые векторы объединяются, передаются в скрытый слой, где срабатывает softmax функция активации, определяющая, какие сигналы пройдут дальше. Языковая модель на основе рекуррентной нейронной сети (RNNLM) преодолевает определенные ограничения прямой связи NNLM, не имеет слоя проекции [14].

Готовые модели

У Google есть предварительно обученные open-source модели для большинства языков (английская версия). Модель использует три скрытых слоя нейронной сети прямого распространения, обучена на корпусе English Google News 200B и генерирует 128-мерный эмбеддинг.

Преимущества

* Простота. Модель быстро обучается и генерирует эмбеддинги, что достаточно для большинства простых приложений.
* Предварительно обученные версии доступны на многих языках.

Недостатки

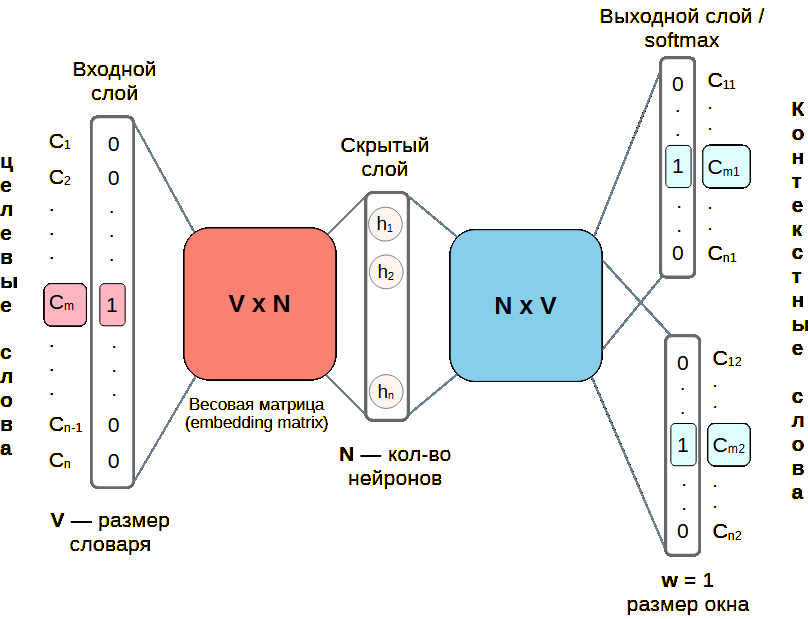
* Не учитывает долгосрочные зависимости.
* Простота ограничивает возможности использования.
* Новые embeddings модели намного мощнее.

### Word2vec

В 2013 году Томас Миколов (Tomas Mikolov) из Google предложил более эффективную модель обучения векторных представлений слов – Word2vec. Метод основывался на предположении, что слова, которые часто находятся в одинаковых контекстах, имеют схожие значения. Это остается все также RNN модель. Изменения были просты – устранение скрытого слоя и аппроксимация (упрощение) цели при обучении – но стали поворотной точкой в развитии языковых моделей NLP.

Вместо алгоритма непрерывного мешка слов модель Word2Vec использует Skip-gram (словосочетание с пропуском). Цель этой модели прямо противоположная предыдущей модели – предсказать окружающие слова на основе центрального.

Архитектура модели представлена на рисунке 2. V - количество слов в словаре после обучения, каждое слово в словаре описывается как вектор с однократным кодированием (двоичный вектор, в котором только позиция соответствующего слова имеет значение 1), N - количество нейронов (размерность векторного пространства слов). Весовая матрица VxN хранит обученный вектор и модель предсказываются векторы, которые соответствуют словам близким по контексту входному — то есть при обучении находившихся слева и с права в тексте (окно w=1).

Рисунок 2. архитектура Word2vec ИНС (skip-gram), 1 скрытый слой, окно = 1

Чтобы сделать обучение эффективнее, используется негативное семплирование (Negative Sampling): модели предоставляются слова, которые не являются контекстными соседями. Многие слова в текстах не встречаются вместе, поэтому модель выполняет много лишних вычислений. Подсчёт softmax — вычислительно дорогая операция. Подход Negative Sampling позволяет максимизировать вероятность встречи нужного слова в контексте, который является для него типичным, и минимизировать – в редком/нетипичном контексте.

Word2Vec сильно зависит от эффективности нормализации softmax. Функция softmax берет вектор ненормализованных вероятностей, созданный нейронной сетью, и применяет экспоненциальное преобразование к каждому элементу. Это преобразование гарантирует, что все результирующие значения будут положительными и что сумма всех значений будет равна 1. Нормализованные вероятности, создаваемые softmax, могут быть непосредственно интерпретированы как вероятность того, что каждое слово будет следующим словом в предложении [14].

С помощью Word2Vec удалось значительно повысить точность при гораздо меньших вычислительных затратах (обучение высококачественных векторов слов на наборе данных из 1,6 миллиарда слов заняло менее суток). Как показано в таблице ниже, Word2Vec превзошёл NNLM и RNNLM по точности. Это объясняется его способностью улавливать более тонкие семантические и синтаксические связи между словами по сравнению с NNLM и RNNLM [14]. На таблице 1 показано сравнение семантической точности определенной по тесту на родственность MSR – широко используемого эталонного теста для оценки семантических связей между словами.

Таблица 1 – Сравнение архитектур с использованием моделей, обученных на одних и тех же данных, с 640-мерными векторами слов

|  |  |  |  |
| --- | --- | --- | --- |
| Архитектура модели | Набор тестов на семантико-синтаксическую взаимосвязь слов | | Связанность слов MSR  (тестовый набор[20]) |
| Семантическая точность, % | Синтаксическая точность,% |
| RNNLM | 9 | 36 | 35 |
| NNLM | 23 | 53 | 47 |
| CBOW | 24 | 64 | 61 |
| Skip-gram | 55 | 59 | 56 |

Готовые модели

Предварительно обученная модель доступна в интернете. В Python-проект её можно импортировать с помощью библиотеки gensim.

Преимущества

* Простая архитектура: feed-forward, 1 вход, 1 скрытый слой, 1 выход.
* Модель быстро обучается и генерирует эмбеддинги (даже на предоставляемые слова).
* Эмбеддинги наделены смыслом, спорные моменты поддаются расшифровке.
* Методология может быть распространена на множество других областей/проблем (например, Lda2vec).

Недостатки

* Обучение на уровне слов: нет информации о предложении или контексте, в котором используется слово.
* Совместная встречаемость игнорируется. Модель не учитывает то, что слово может иметь различное значение в зависимости от контекста использования.
* Не очень хорошо обрабатывает неизвестные и редкие слова.

### GloVe (Global Vectors)

GloVe тесно ассоциируется с Word2Vec: алгоритмы появились примерно в одно и то же время и опираются на интерпретируемость векторов слов. Модель GloVe пытается решить проблему эффективного использования статистики совпадений. GloVe минимизирует разницу между произведением векторов слов и логарифмом вероятности их совместного появления с помощью стохастического градиентного спуска.

Готовые модели

Эмбеддинги GloVe легко доступны на веб-сайте Стэнфордского университета.

Преимущества

* Простая архитектура без нейронной сети.
* Модель быстрая, и этого может быть достаточно для простых приложений.
* GloVe улучшает Word2Vec. Она добавляет частоту встречаемости слов и опережает Word2Vec на большинстве бенчмарков.
* Осмысленные эмбеддинги.

Недостатки

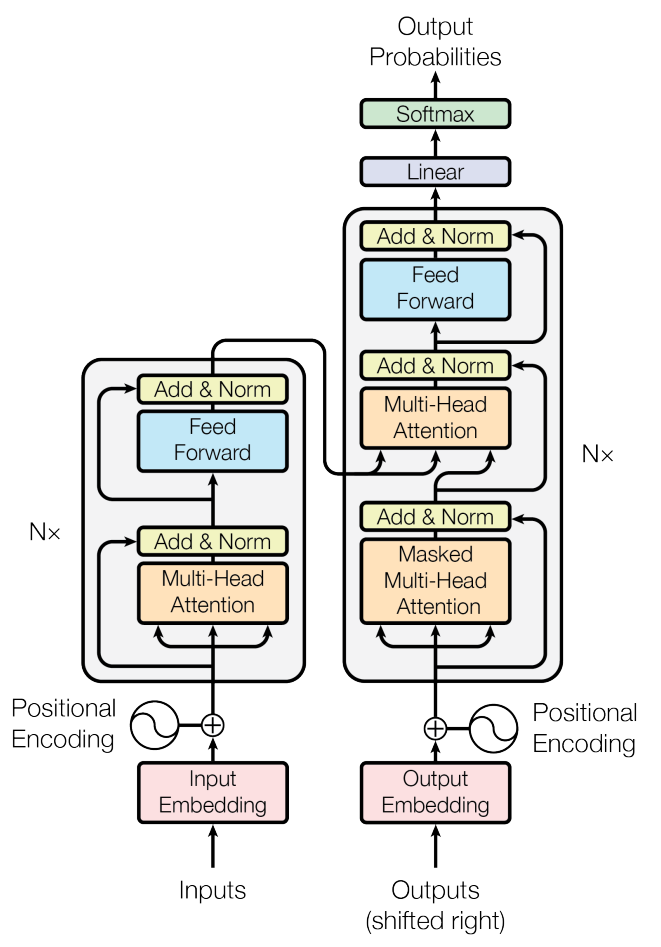
* Хотя матрица совместной встречаемости предоставляет глобальную информацию, GloVe остаётся обученной на уровне слов и даёт немного данных о предложении и контексте, в котором слово используется.
* Плохо обрабатывает неизвестные и редкие слова.

### Transformers

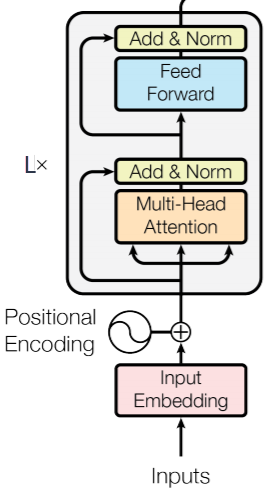
Более продвинутые языковые модели с контекстами порядка тысяч токенов и для решения широкого спектра NLP-задач строятся обычно на архитектуре Transformers. Популярность трансформеров взлетела до небес в связи с появлением больших языковых моделей вроде ChatGPT, GPT-4 и LLama. Эти модели созданы на основе трансформерной архитектуры и демонстрируют отличную производительность в понимании и синтезе естественных языков.

Главное преимущество трансформеров заключается в их способности обрабатывать длительные зависимости в последовательностях. Кроме того, они очень производительны, могут обрабатывать последовательности параллельно. Пожалуй, самый важный механизм в трансформенной архитектуре — это внимание. Он позволяет нейросети понять, какая часть входной последовательности наиболее релевантна задаче. Механизм внимания определяет для каждого токена последовательности, какие другие токены необходимы для его понимания в данном контексте [15].

Чтобы избавится от недостатков использования RNN сетей для обработки токенов, а именно требования наличия очень большого скрытого состояния для того чтобы модель "вспоминала" контекст большого числа шагов назад, а также сложность распараллеливания RNN из-за линейности вычисления состояния RNN-слоя и были спроектированы трансформеры. Они основываются на механизме самовнимания который позволяет проще модели "вспоминать" то что было ранее при обучении.

Рисунок 3. Схема ИНС архитектуры Transformer

Здесь, на рисунке 3 - энкодер преобразует входную последовательность символьных представлений (x1, ..., xn) в последовательность непрерывных представлений z = (z1, ..., zn). При заданном значении z декодер затем генерирует выходную последовательность (y1, ..., ym) символов по одному элементу за раз. На каждом шаге модель выполняется авторегрессия, используя ранее сгенерированные символы в качестве дополнительных входных данных при генерации следующего [16].

Рисунок 4. Схема энкодера архитектуры Transformer

На рисунке 4 представлено устройство энкодера. Он по очереди применяет к исходной последовательности N блоков. Каждый блок выдаёт последовательность такой же длины. В нём есть два важных слоя, multi-head attention и feed-forward. После каждого из них к выходу прибавляется вход (это стандартный подход под названием residual connection) и затем активации проходят через слой layer normalization: на рисунке эта часть обозначена как «Add & Norm». У декодера схема похожая, но внутри каждого из N блоков два слоя multi-head attention, в одном из которых используются выходы энкодера.

Первая часть transformer-блока — это слой self-attention. От обычного внимания его отличает то, что выходом являются новые представления для элементов той же последовательности, что мы подали на вход, причем каждый элемент этой последовательности напрямую взаимодействует с каждым.

Вторая часть трансформерного блока называется feed-forward network (FFN) и представляет собой два обычных полносвязных слоя, применяемых независимо к каждому элементу входной последовательности [17].

Существует много разных трансформерных архитектур, и большинство можно разделить на три типа.

Энкодеры

Модели-энкодеры синтезируют контекстуальные эмбеддинги, которые можно использовать в последующих задачах вроде классификации или распознавания именованных сущностей, поскольку механизм внимания может обрабатывать всю входящую последовательность. Самое популярное семейство чистых трансформеров-энкодеров — это BERT и его разновидности.

Декодеры

Этот тип архитектур почти идентичен предыдущему, главное отличие в том, что декодеры используют маскированный (или причинный) слой self-attention, поэтому механизм внимания может принимать только текущий и предыдущие элементы входной последовательности. То есть контекстуальные эмбеддинги учитывают только предыдущий контекст. К популярным моделям-декодерам относится семейство GPT.

Энкодеры-декодеры

Изначально трансформеры были представлены как архитектура для машинного перевода и использовали и энкодеры, и декодеры. С помощью энкодеров создается промежуточное представление, прежде чем с помощью декодера переводить в желаемый формат. Хотя энкодеры-декодеры сегодня менее распространены, архитектуры вроде T5 показывают, что задачи вроде ответов на вопросы, подведения итогов и классификации можно представить в виде преобразование последовательности в последовательность и решить с помощью описанного подхода.

## Описание концепции «словесных вложений» рассматриваемых моделей, Векторные представления слов

Крупными шагами в достижении современного уровня обработки текста и речи стали методы получения векторных представлений слов на основе обучения (Word2vec, FastText и т. д.) описаные в решениях выше а так-же появление рекуррентных архитектур ИНС (RNN, LSTM, GRU), появление механизма внимания Богданова (Bahdanau Attention Mechanism [18]) и, наконец, появление архитектуры трансформер [16] с применением механизма многопоточного самовнимания (multi-head self-attention) и большого количества производных моделей на её основе (BART, BERT, GPT, T5 и тд.). Но все это опирается на базовую идею о представлении какого-либо языка как векторного пространства, а слов в нем как векторов, которые в совокупности выражают семантическую (смысловую) связь между понятиями языка. Рассмотрим этот подход подробнее.

Чтобы компьютерная программа могла воспринимать слово нужно представить его в читаемом для машины виде. Рассмотрим различные стратегии для достижения этого.

В русскоязычной литературе эмбеддингами в общем случае обычно называют такие числовые векторы, которые получены из слов или других языковых сущностей.

### Простые бинарные OHE ембенддинги

В самой примитивной форме эмбеддинги слов получают простой нумерацией слов в некотором достаточно обширном словаре и установкой значения единицы в длинном векторе размерности, равной числу слов в словаре. Например, возьмем Толковый словарь Ушакова и пронумеруем все слова с первого до последнего. Так слово «абака» преобразуется в число 5, «абажур» - в 7 и так далее. Всего слов в словаре 85 289 слов. Эмбеддинг слова «абака» будет иметь 85288 нулей на всех позициях, кроме пятой, где будет стоять 1, а слово «абажур» - соответственно будет иметь нули на всех 85288 позициях кроме седьмой, где будет единица. Этот метод построения эмбеддингов называют унитарным кодированием, а в современной англоязычной литературе – one-hot encoding, а вектор полученный таким образом бинарным OHE-вектором.

Любому предложению на русском языке можно попытаться поставить в соответствие последовательность – более правильно с математической точки сказать - кортеж таких 85289-мерных векторов. И тогда действия над словами могут быть преобразованы в действия над этими числовыми векторами, что собственно и свойственно компьютеру [19].

Идея очень продуктивна в своей простоте — натуральный ряд бесконечен и можно перенумеровать все слова, не опасаясь проблем. Но у этой идеи есть и существенный недостаток: слова в словаре следуют в алфавитном порядке, и при добавлении слова нужно перенумеровывать заново большую часть слов. Но даже это не является настолько важным, а важно то, буквенное написание слова никак не связано с его смыслом. Например, слова “петух”, “курица” и “цыпленок” имеют очень мало общего между собой и стоят в словаре далеко друг от друга, хотя очевидно обозначают самца, самку и детеныша одного вида птицы. То есть мы можем выделить два вида близости слов: лексический (написание) и семантический (смысл). Как мы видим на примере с курицей, эти близости не обязательно совпадают. Можно для наглядности привести обратный пример лексически близких, но семантически далеких слов — "зола" и "золото".

### Частотный эмбеддинг

Как мы видим, чтобы получить возможность представить семантическую близость ембеддинг должен иметь возможность сохранять какую-то информацию о ней. Посмотрим в упомянутый Толковый словарь Ушакова – вы не найдете там такого популярного слова как «компьютер» и соответственно не можете получить вектор и никак проанализировать слово. Существенно снизить вероятность такой проблемы, можно не используя специальный словарь для составления вектора слова, а нумеруя слова в произвольном обширном наборе текстов, например, в Википедии, Большой российской энциклопедии. Для этих целей сегодня создаются специальные наборы, называемые корпусами текстов. Однако, использование представительных корпусов само по себе не помогает извлечь какую-либо пользу из превращения конкретного текста в кортеж чисел. Ведь любой текст на естественном языке представляет собой не только набор слов, но и несет некоторую семантику, смысл.

Задача научить компьютерную систему как-либо понимать смысл текста, извлекать из него семантическую информацию, используя примитивный эмбеддинг (one-hot encoding), является неразрешимой. Иными словами, системе нужна дополнительная информация, не только простые значения. Поэтому следующий шаг в обработке текстов был сделан путем учёта того факта, насколько часто каждое слово языка (термин) встречается в корпусе и насколько важно его появление в конкретном тексте. Так возник частотный эмбеддинг, в котором каждому слову в позицию, соответствующую его номеру, ставится в соответствие число - частота слова, а точнее, скорректированное значение частоты: формулу TF-IDF. Эта аббревиатура означает "term frequency — inverse document frequency".

|  |  |
| --- | --- |
|  | (4) |

где TF — это частота слова в тексте . IDF — существенно более интересная вещь: это логарифм обратной частоты распространенности слова в корпусе (коллекции текстов).

Распространенностью называется отношение числа текстов, в которых встретилось искомое слово, к общему числу текстов в корпусе. TF-IDF еще называют метрикой качества, важности слова в корпусе текста. С помощью TF-IDF тексты можно сравнивать, и делать это можно с меньшей опаской, чем при использовании обычных частот. Благодаря данному показателю можно снизить весомость наиболее широко используемых слов (предлогов, союзов, общих терминов и понятий). Для каждого термина в рамках определённого корпуса текстов предусматривается лишь одно единственное значение частоты слова. Показатель обратной частоты будет выше, если определённое слово с большой частотой используется в конкретном тексте, но редко - в других документах.

Используя эмбеддинги в виде таких векторов, удалось впервые осуществить автоматический семантический анализ текстов, определяя имеющиеся в корпусе текстов темы и классифицировать тексты по основным темам.

Существует несколько успешно применяемых алгоритмов такого анализа: латентный семантический анализ, латентное размещение Дирихле и тематические модели Biterm для коротких текстов. Использование таких моделей, например, позволило сортировать гигантские потоки электронных писем по тематике и направлять их согласно предписанным правилам. На этом этапе внутри NLP начал формироваться мощный поток технологий, которые получили общую формулировку, как «понимание естественного языка».

### Контекстные ембеддинги. Скользящее окно. Word2vec

Описанные выше подходы были (и остаются) хороши для времен (или областей), где количество текстов мало и словарь ограничен, хотя, как мы видим, там тоже есть свои сложности. Но с приходом в нашу жизнь интернета все стало одновременно и сложнее, и проще: в доступе появилось великое множество текстов, и эти тексты с изменяющимся и расширяющимся словарем. С этим надо было что-то делать, а ранее известные модели не могли справиться с таким объемом текстов. Количество слов в английском языке очень грубо составляет миллион — матрица совместных встречаемостей только пар слов будет 106 x 106. Такая матрица даже сейчас не очень лезет в память компьютеров, а, скажем, 10 лет назад про такое можно было не мечтать. 2013 году тогда мало кому известный чешский аспирант Томаш Миколов предложил свой подход к word-embedding, который он назвал word2vec. Его подход основан на другой важной гипотезе, которую в науке принято называть гипотезой локальности — “слова, которые встречаются в одинаковых окружениях, имеют близкие значения”. Близость в данном случае понимается очень широко, как то, что рядом могут стоять только сочетающиеся слова [19]. Модель, предложенная Миколовым, очень проста, в отличии, например, от основанных на латентном распределении Дирихле (LDA) моделей, (и потому так хороша) — мы будем предсказывать вероятность слова по его окружению (контексту). То есть мы будем учить такие вектора слов, чтобы вероятность, присваиваемая моделью слову, была близка к вероятности встретить это слово в этом окружении в реальном тексте [20, 21].

|  |  |
| --- | --- |
|  | (5) |

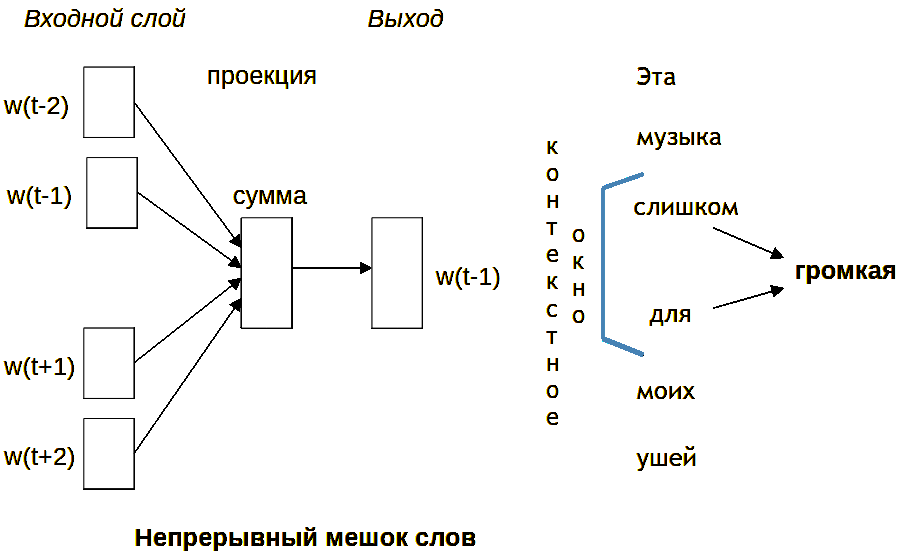
где — вектор целевого слова, — это некоторый вектор контекста, вычисленный (например, путем усреднения) из векторов окружающих нужное слово других слов. А — это функция, которая двум векторам сопоставляет одно число, например это может быть косинусное расстояние или коэффициент Отиаи.

Приведенная формула называется softmax, то есть “мягкий максимум”, мягкий — в смысле дифференцируемый. Это нужно для того, чтобы модель могла обучиться с помощью backpropagation, то есть процесса «обратного распространения ошибки».

Процесс тренировки устроен следующим образом: мы берем последовательно (2k+1) слов, слово в центре является тем словом, которое должно быть предсказано. А окружающие слова являются контекстом длины по k с каждой стороны — это и называется «контекстным окном». Каждому слову в нашей модели сопоставлен уникальный вектор, который мы меняем в процессе обучения нашей модели.

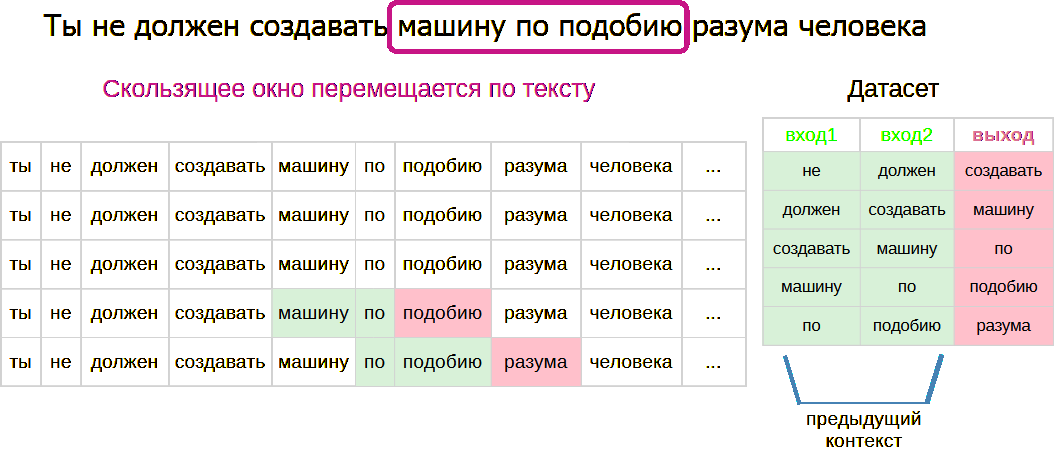
В целом, этот подход называется CBOW — continuous bag of words, continuous потому, что мы предоставляем модели последовательно наборы слов из текста, a BoW потому что порядок слов в контексте не важен.

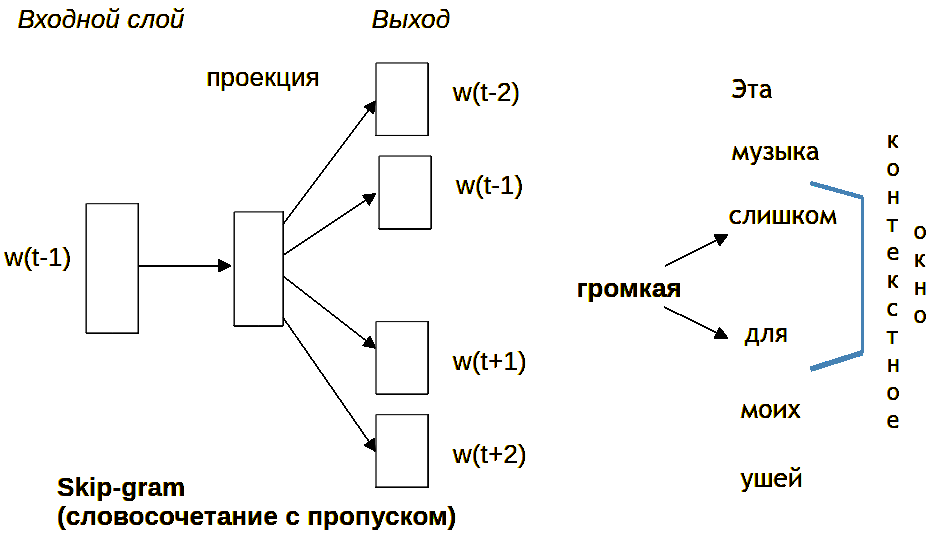
Также Миколовым сразу был предложен другой подход — прямо противоположный CBOW, который он назвал skip-gram, то есть “словосочетание с пропуском”. Мы пытаемся из данного нам слова угадать его контекст (точнее вектор контекста). В остальном модель не претерпевает изменений.

Рисунок 5. Непрерывный набор слов (Continuous Bag-of-Words, Word2Vec)

Для CBOW архитектуры объективная (целевая) функция, которая используется для предсказания целевого слова использует логарифмическую сумму вероятностей окружающий искомое n-слов слева и справа [20]:

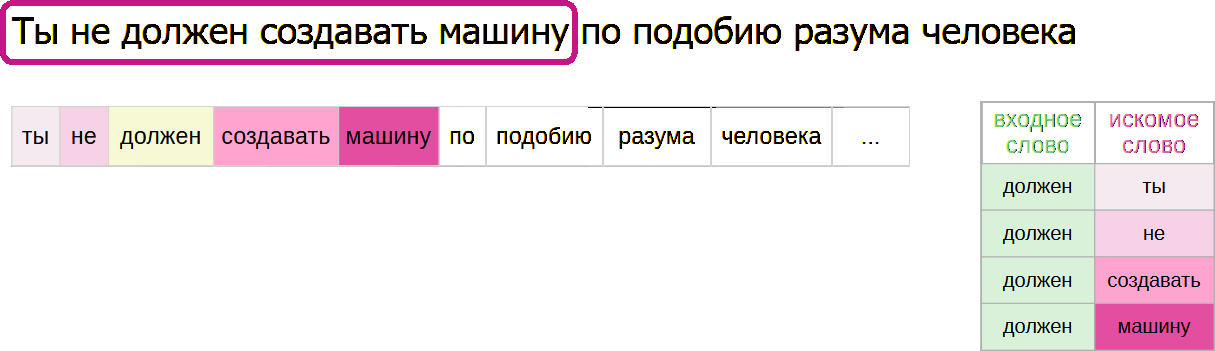
|  |  |
| --- | --- |
|  | (6) |

Рисунок 6. Принцип составления датасета для получения CBOW-модели Word2Vec для корпуса текста(рамка — контекстное окно)

Рисунок 7. Словосочетание с пропуском (Skip-gram, Word2Vec)

В тоже время для Skip-gram модели объективная функция трансформируется в сумму логарифмической вероятности окружающих n-искомых слов вокруг целевого слова [20]:

|  |  |
| --- | --- |
|  | (7) |

Рисунок 8. Принцип составления датасета для получения Skip-gram-модели Word2Vec для корпуса текста (рамка — контекстное окно)

С помощью упомянутых выше моделей Word2vec и аналогов GloVe возможно относительно дешево вычислительно (относительно тяжелых моделей типа BERT и NNLM моделей и других основанных на латентном семантическом анализе (LSA) и латентном распределении Дирихле (LDA)) находить семантическое сходство, выявлять словосочетания в тексте (парафразы), подходящие по контексту слова и другие операции имея предобученые на подходящем корпусе заданной тематике текста ембеддинги.

## Подведение итогов

В программе требуется работать с довольно независимым коротким пользовательским запросом, отчего контекст токенов не может быть большим. Кроме того, т.к. программа базы знаний призвана лишь оптимизировать пользовательский запрос для заданной поисковой машины, которая на данный работает по принципам лексемизации базы и вычисления статистической важности той или иной лексемы в наборе документов — для моих нужд достаточно моделей семейства Word2Vec и дистиллированных WordCNN моделей, далее разберем их подробнее.

Для русского языка наиболее известными моделями семейства Word2Vec являются: RusVectores и Navec. – RusVectores — набор семантических моделей для русского языка. Для обучения использовались: Национальный корпус русского языка и дамп русского сегмента Википедии; – Navec семантические модели из состава проекта Natasha. Разработчики проекта Natasha [22, 23] провели сравнительное тестирование моделей RusVectores и Navec (результат в таблице 2).

Таблица 2 – Сравнение в.с. моделей

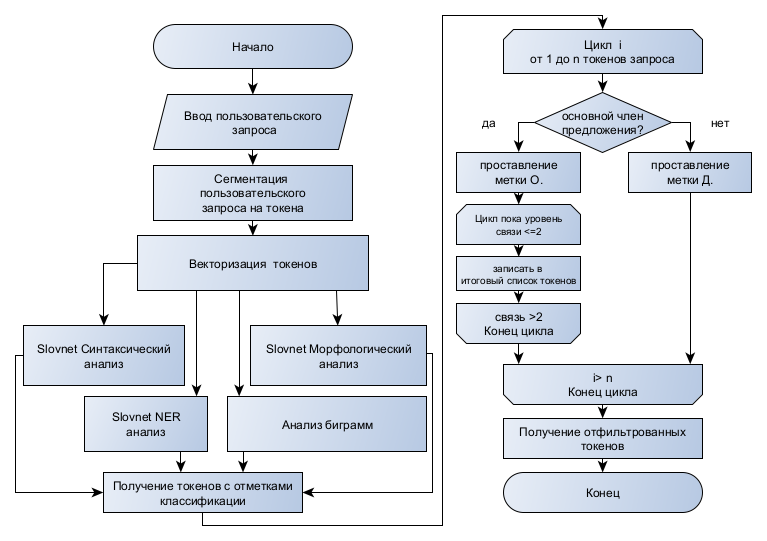
|  |  |  |  |
| --- | --- | --- | --- |
|  | Качество | Размер модели, МБ | Размер словаря, ×10 3 |
| Navec | 0.719 | 50.6 | 500 |
| RusVectores | 0.638–0.726 | 220.6–290.7 | 189–249 |

Всеже большое количества современных NLP задач решается с помощью трансформерных ИНС, которые в свою очередь сети обучения с учителем. При использовании принципа обучения с учителем получение положительного результата невозможно без достаточного объёма качественных (нормализованных, очищенных и размеченных) данных. В настоящее время доступно несколько достаточных по объёму корпусов текста на русском языке. Удобной «точкой доступа» является агрегатор ссылок на корпуса русского текста и датасеты для обучения моделей выделения именованных сущностей это Corus [24] из состава проекта Natasha. В настоящее время через Corus доступно более 30 корпусов и датасетов в том числе: OpenCorpora (открытый корпус), Omnia Russica (TaigaCommon Crawl, Wikipedia и Aranea), дамп русского сегмента Википедии. На сайте ruscorpora.ru доступен Национальный корпус русского языка. Используя данные датасеты проектом natasha была обучена Slovnet BERT NER — аналог DeepPavlov BERT NER. Далее дистилляция её через синтетическую разметку (Nerus) в WordCNN-CRF (Сверточная нейронная сеть с условным случайным полем (CRF) для классификации предложений) c квантованными эмбеддингами (Navec) и движок для инференса на NumPy. Подробнее: обучается тяжёлая модель c BERT-архитектурой на небольшом вручную аннотированном датасете. Размечается для нее корпус, получится большой грязный синтетический тренировочный датасет. Обучается на нём компактная примитивная модель. Этот процесс называется дистилляцией: тяжёлая модель — учитель, компактная — ученик. Считается, что BERT-архитектура избыточна для задачи NER, компактная модель несильно проиграет по качеству тяжёлой. В проекте Natasha библиотека Slovnet занимается подготовкой модели-учителя и модели ученика, обучением и инференсом современных моделей для русскоязычного NLP. В библиотеке Slovnet есть ещё две модели, обученные по этому рецепту: Slovnet Morph — морфологический теггер, Slovnet Syntax — синтаксический парсер. Slovnet Morph отстаёт от тяжёлой модели-учителя на 2 процентных пункта, Slovnet Syntax — на 5. У обеих моделей качество и производительность выше существующих решений для русского на новостных статьях [25].

# Описание разработанных алгоритмов

## Алгоритм синтаксического анализа запроса, выявление основной части запроса

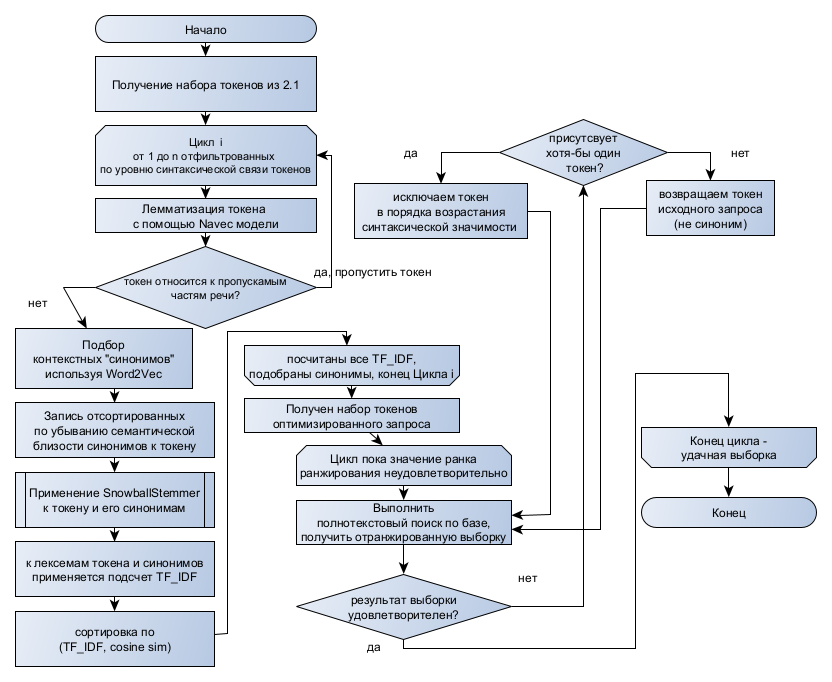
На рисунке 9 представлена блок-схема алгоритма 2.1

Рисунок 9. Алгоритм синтаксического анализа запроса, выявление основной части

* Начинается алгоритм с инициализации запроса в качестве документа, сегментированного на токены, в качестве разделителей используются знаки препинания и пробелы. Во время сегментации по устоявшимся стандартам NLP анализа токены менее 3 символов пропускаются — т. к. считается что их семантическая значимость низка и имеет повышеную омонимичность с другими морфемами слов что вносит изличную энтропию в анализ.
* После сегментации на токены — документ (запрос) векторизируется с помощью модели эмбеддингов Navec, задача модели представить токены (слова) как векторы векторного пространства, обученного на корпусе «navec\_news\_v1\_1B\_250K\_300d\_100q.tar», ранее были указаны особенности работы этой модели.
* После векторизации параллельно происходит применение к документу (с уже векторизоваными токенами) две предобученых модели проекта Slovnet: Морфологический анализатор, Синтаксический анализатор. Эти модели тоже обучены на размеченных корпусах текста новостных статей.
* Сегментированный на токены документ поискового запроса тестируется на предмет наличия «биграмм» или словосочетаний из двух токенов и именованых сущностей (NER) двумя моделями. Для выявления биграмм в рамках данной работы была обучена статистическая модель на корпусе текстов проекта «Leipzig Corpora Collection» [25] и «Corus» [23]. Для выявления именованых сущностей используется предобученная модель из проекта Slovnet [26]. В токены документа которые относятся к биграмам или к NER добавляется атрибут принадлежности токена.
* После применения моделей для анализа текста — токены помечаются дополнительными атрибутами: определяется часть речи токена, и каким членом предложения является токен, также отмечается относится ли токен к именам собственным или персональным именам (NER) — дело в том, что синтаксическая значимость именовыных сущностей выше чем нарицательных существительных, NER, позднее в агоритме оптимизации NER точно также, как и словосочетания исключаются из оптимизаций.
* Полученые от модели синтаксического анализа маркеры связи членов предложения позволяют выделить основные члены предложения первого уровня связи как подлежащее и сказуемое и т.д.. Если запрос представляет собой сложное предложение, то запрос сокращается до 2 уровня связи — тоесть основные и дополнительные члены предложения — подлежащее, сказуемое, одно определение или дополнение

## Алгоритм оптимизации по симантической близости и TF-IDF

На рисунке 10 представлена блок-схема алгоритма 2.2

Рисунок 10. Алгоритм оптимизации по семантической близости и TF-IDF

* Программа использует набор токенов полученный после алгоритма 2.1. Токены итерируются, исходя из результатов синтаксического анализа группируются в группы основных и дополнительных членов предложения
* Токены содержащие отметки о NER или что это часть биграммы — сохраняются независимо от части речи и предложения
* Отфильтрованные по уровню анализируемой связи членов предложения (3) все исходные токены фильтруются, оставля лишь основные + NER + биграммы. В зависимости от типов предложения (полные, неполные, односоставные, многосоставные, однородные и т.д.) определение условного уровня связи может различаться
* После фильтрации каждый токен лемматизируется с помощью семантической модели
* После лемматизации из набора токенов дополнительно убираются члены предложения: знаки препинания, предлоги, союзы, местоимения, не относящиеся к NER и биграммам.
* По лемматизированным токенам подбираются синонимы предобученной моделью Word2Vec, определяется индекс близости (косинусное расстояние). Синонимы сортируются по убыванию сходства и записыаются в токен.
* К токену и его синонимам применяем стеммер (обрезаем изменяемые части слова, оставляем лексему, такую же как хранится в базе данных полнотекстового поиска)
* После этого цикла начинается еще один уже по оставшимся токенам, для которых найдены и синонимы и применены стеммеры. Для каждого токена и синонима выполняется функция подсчета показателя TF\_IDF в базе полнотекстового поиска.
* Для каждого оставшегося токена и синонимов ранжируется ряд по двум ключам сортировки — с первым ключем TF\_IDF лексемы, и вторым ее синтаксическое сходство с исходным токеном. Если у синонима TF\_IDF выше чем исходного токена — то он времено подменяет исходный токен в оставшемся запросе. Для всех оставшихся токенов применяется данный алгоритм. После него остается оптимизирвоанный список лексем которые имеют наибольшую среднюю меру TF\_IDF нашей базы знаний.
* Полученый набор токенов/синонимов отправляется как полнотекстовый запрос базе данных. По результатам запроса мы получаем выборку и значение Ранка для каждого результата.
* Если результат неудовлетворителен (выборка не преодолела некий порог ранка) то пробуем исключить токен/синоним начиная с неосновных членов предложения, повторяем поиск до тех пор пока результат поиска не будет удовлетворителен. Если после последнего токена результат по-прежнему не удовлетворителен — алгоритм аналогично в обратном порядке заменяет «оптимизированный» токен исходным.

# Описание разработанной программы

## Назначение программы и выбранный инструментарий

Разработанная программа представляет собой демонстрационное приложение, способное добавлять ответы на набор заданных вопросов сохраняя их в базе данных на удаленном сервере. Пользователь может вводить запрос и получить отранжированные результаты ответов на него, версия программы для магистерской работы содержит визуальную демонстрацию алгоритмов, применяемых к пользовательскому запросу, а также подсчет метрик качества выдачи. Урезанная версия алгоритма полнотекстового поиска также реализована в рамках Telegram-бота.

Программа написана на языке Python 3.7 с использованием библиотеки PyQt, которая предоставляет инструментарий для построения графических приложений [27].

Выбор продиктован, во-первых, растущей с каждым годом популярностью данных продуктов, во-вторых, их кроссплатформенностью, что обеспечит возможность запуска спроектированного программного продукта на операцион–ных системах ОС GNU/Linux, Windows, macOS.

В работе использован комплекс модулей и оптимизированных эмбедингов и моделей русского языка проекта natasha обученных на больших корпусах новостных статей и художественной литературы. Для синтаксического и морфологического анализа используются модели семейства Slovnet, которые представляют собой дистилированные WordCNN-CRF (Сверточная нейронная сеть с условным случайным полем (CRF) для классификации предложений) версии большой Slovnet BERT. Для вычисления семантической близости и векторизации слов в работе используются Navec модель, обученная на корпусе порядка 12 млрд слов художественного или 1 млрд слов новостного текста. Для выявления словосочетаний в ходе работы на корпусах Лейпцигского университета [28] была обучена простая статистическая модель парафраз из библиотеки gensim [29].

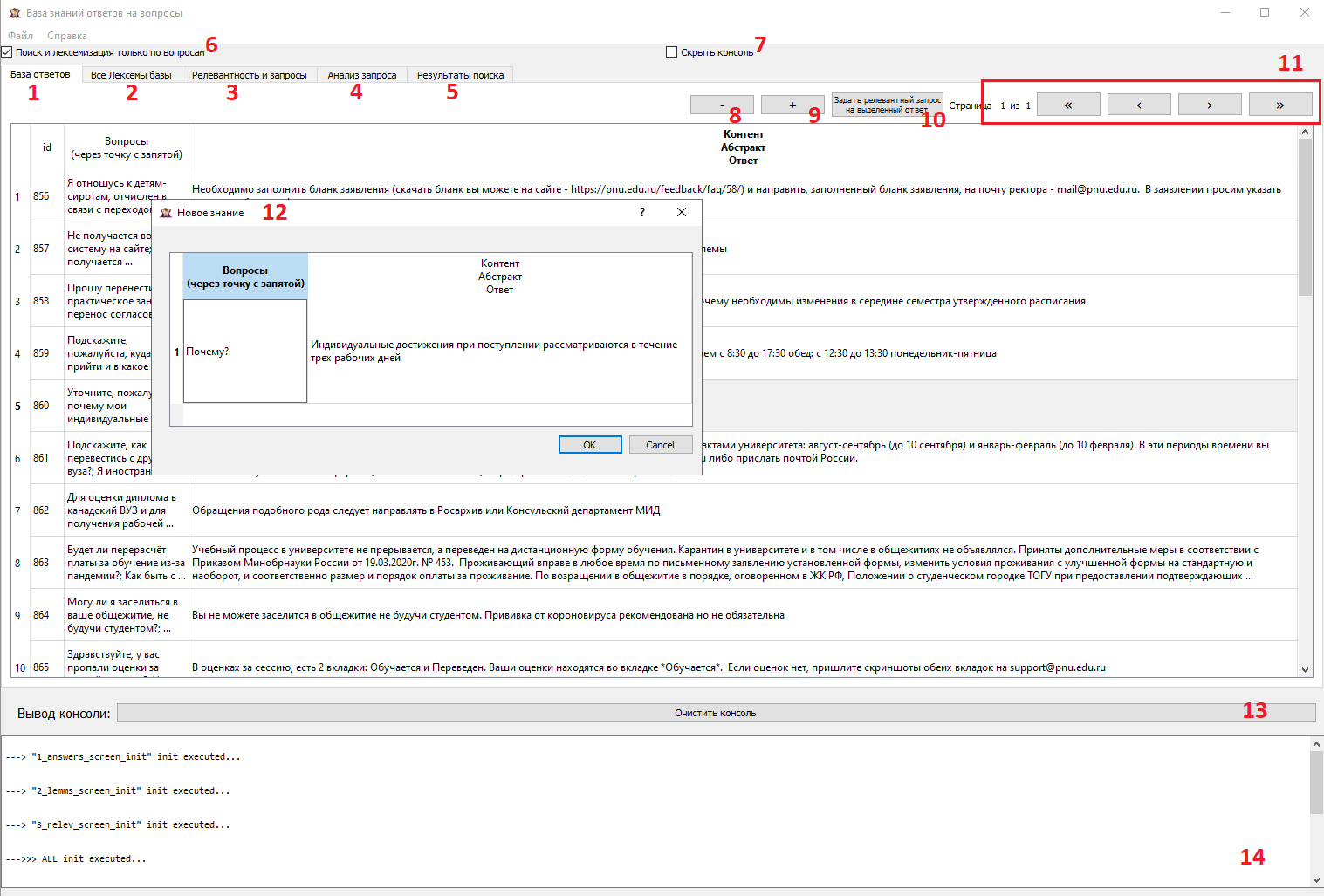
Цель конечной системы, в программе которая позволит пользователю получать Ответы на заданные в, например, чат-боте или на сайте, Вопросы подразумевает детерминированность и стабильность самого Ответа, что сложно на текущий момент обеспечивать генеративными моделями - в них всегда присутствует уровень энтропии в ответе исходя из вероятностной природы функции активаций ИНС лежащих в основе таких моделей.

Кроме того, если использовать систему Вопросов-Ответов как выполняющую функционал службы поддержки ТОГУ - удобно иметь возможность добавлять знания по мере существования система без дополнительных сложностей требуемых после этого ресурсозатратного "дообучения" модели. Конечно для этой задачи в "мире генеративных лингвистических моделей (LLM)" есть такие техники как RAG (Retrieval-Augmented Generation, генерация ответа с использованием результатов поиска). Большие языковые модели знают многое, но не всё. Так как обучение таких моделей занимает много времени, данные, использованные в последнем сеансе их обучения, могут оказаться достаточно старыми. И хотя LLM знакомы с общеизвестными фактами, сведения о которых имеются в интернете, они ничего не знают о ваших собственных данных. А это — часто именно те данные, которые нужны в приложении в данный момент времени.

До наступления эры LLM модели часто дополняли новыми данными, просто проводя их дообучение. Но теперь, когда используемые модели стали гораздо масштабнее, когда обучать их стали на гораздо больших объёмах данных, дообучение моделей подходит лишь для совсем немногих сценариев их использования. Дообучение особенно хорошо подходит для тех случаев, когда нужно сделать так, чтобы модель взаимодействовала бы с пользователем, используя стиль и тональность высказываний, отличающиеся от изначальных. Один из отличных примеров успешного применения дообучения — это когда компания OpenAI доработала свои старые модели GPT-3.5, превратив их в модели GPT-3.5-turbo (ChatGPT) [30]. Принимая во внимание все это и цели, и задачи разрабатываемой программы - я остановился на использовании ИНС модели только в части предварительной обработки запросов полнотекстового поиска по определенной конечной базе данных ответов.

## Описание интерфейса программы

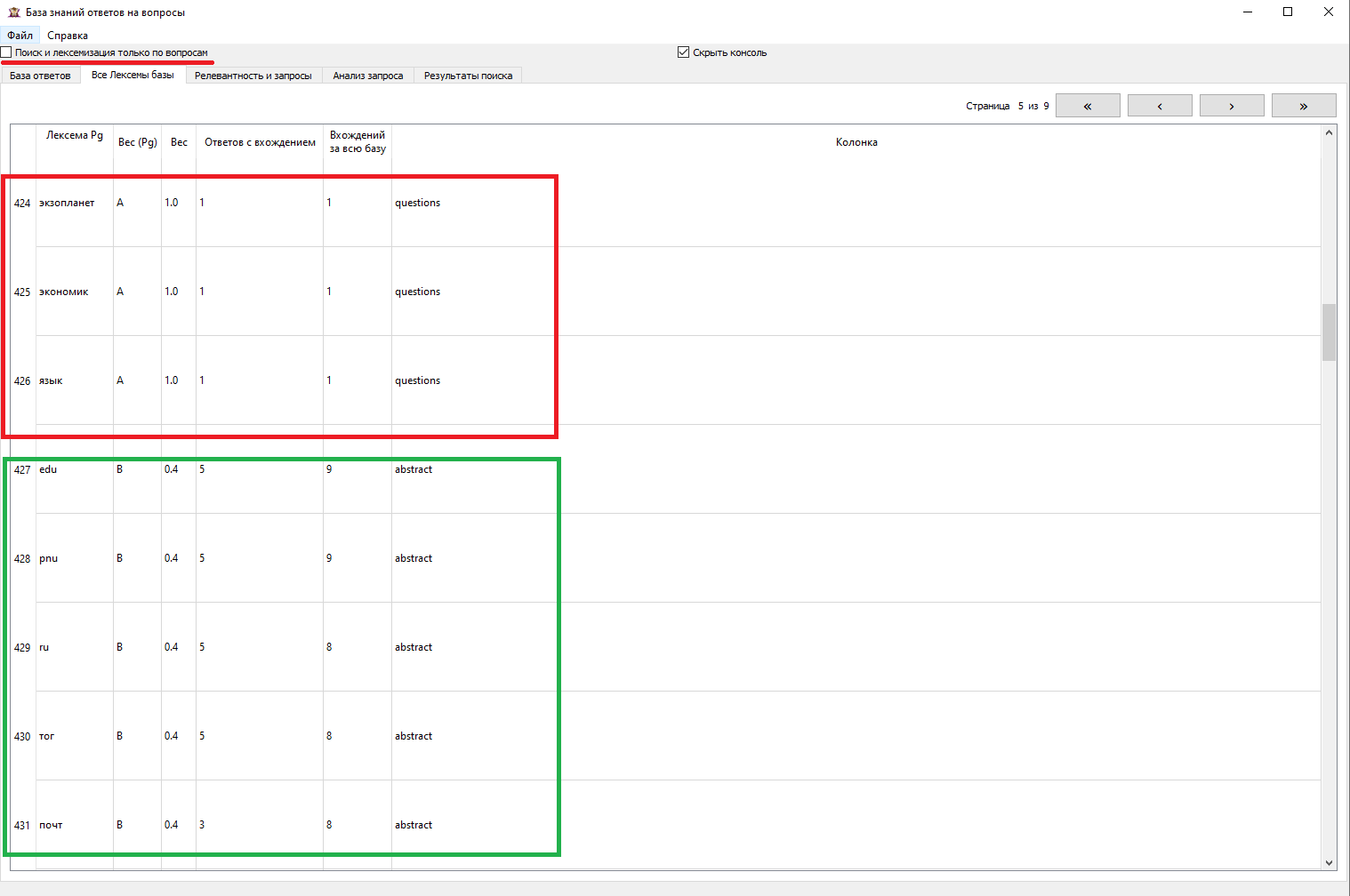
Интерфейс программы представлен на рисунке 11.

Рисунок 11. Начальный экран программы

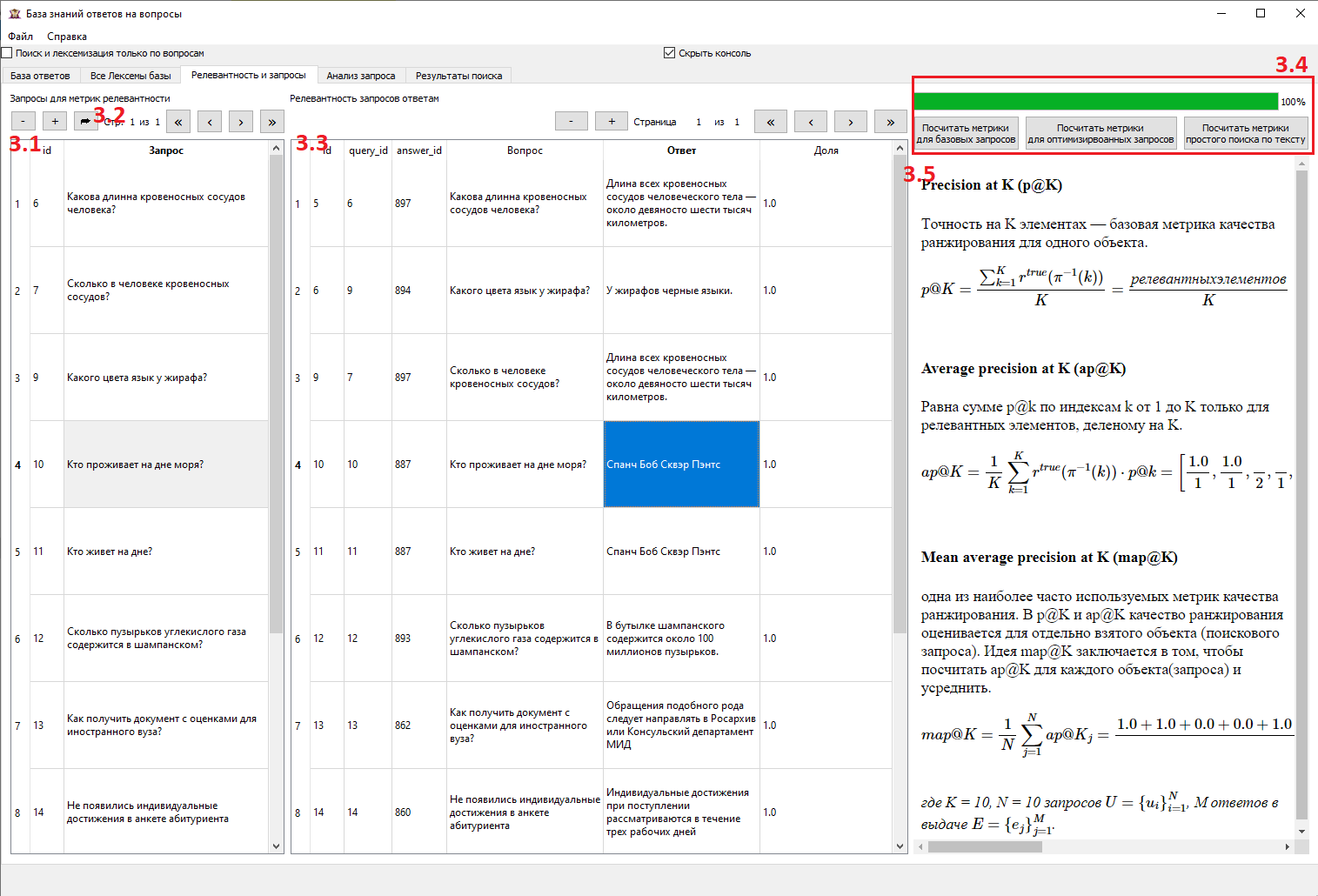
На рисунке 11 под номерами 1-5 вкладки представляющие различный функционал программы.

Первый экран на рисунке 11 позволяет добавлять или удалять записи в базу знаний, на скриншоте показано диалоговое окно добавления новой пары вопрос/ответ. Под номером 14 на рисунке 11 указано поле консольного вывода, которое можно очищать кнопкой 13 или скрыть полностью чекбоксом 7. Под номером 11 интерфейс пагинации, 8/10 соответственно кнопки удаления и добавления, под номером 10 ярлык для внесения запроса для метрик ранжирования сразу из этого экрана.

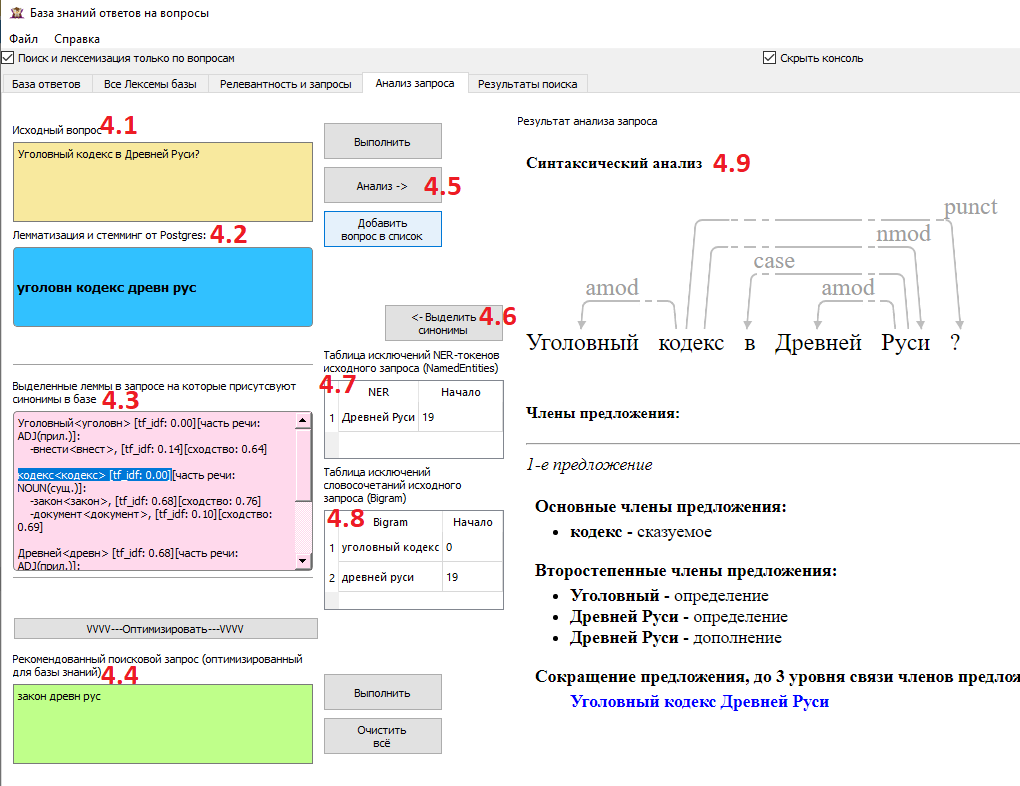
На рисунке 12 показан второй экран показывающие как во общем за всю базу знаний документы (Вопросы и Ответы) были разбиты на лексемы, в колонках указывается сколько лексема встречается в документе (непосредственно в ответе к которому принадлежит) а также за всю базу. Эти данные позволяют посчитать важность слова в базе знаний чтобы посчитать TF\_IDF в дальнейшем. Чем больше совокупная важность лексем, тем выше результат будет в выдаче полнотекстового поиска. Последняя колонка в данной таблице связана с чекбоксом 6 из описания выше — можно переключить режим поиска программы чтобы при поиске ипользовать только лексемы определенных колонок.

Рисунок 12. Экран лексем в базе знаний

На третьем экране программы представлен интерфейс задания исходных данных для подсчета метрик релевантности алгоритма простого поиска по тексту в базе, полнотекстового поиска по лексемам, оптимизированного полнотекстового поиска по лексемам. Под номером 3.1 экран добавления запросов для теста, в данном случае добавлено 10 запросов (K=10). Под пунктом 3.3 на рисунке 13 задание доли релевантности запроса определенному ответу, стоит обратить внимание что запросы слева не всегда точно соответствуют тексту самого вопроса, где то вопрос сокращен или перефразирован.

Рисунок 13. Экран расчета качества поисковой выдачи

В блоке 3.4 находится кнопки проведения трех тестов соответственно: тестирование стандартного полнотекстового поиска по базе знаний, тестирование оптимизированного полнотекстового поискового запроса и обычного вхождение строки (SQL оператор LIKE). При нажатии на данные кнопки происходит расчет показателей точности выдачи из раздела 1.5.

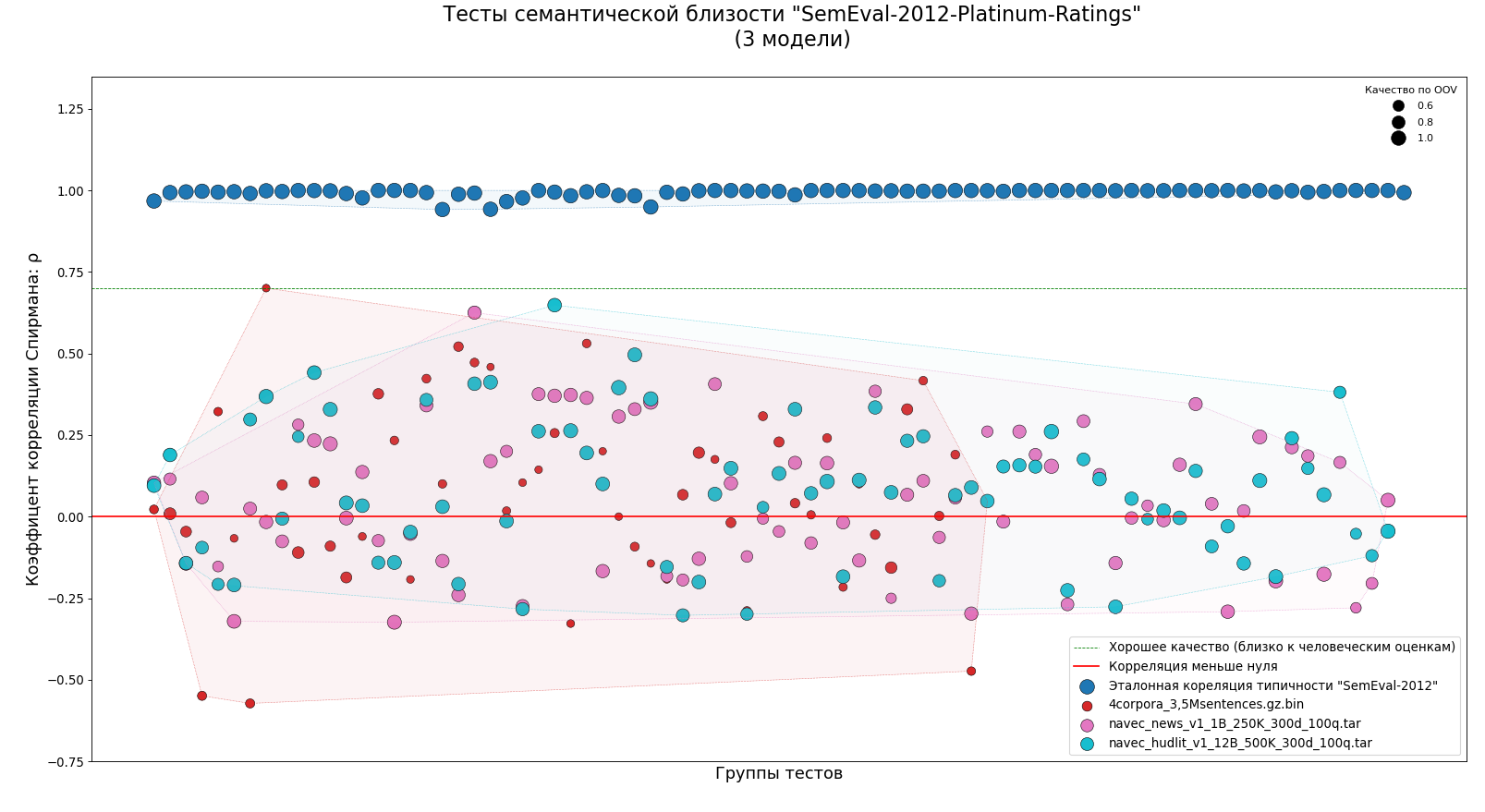
Рисунок 14. Экран анализа запроса

Экран 4 показанный на рисунке 14 показывает развернутый алгоритм по которому оптимизируется пользовательский запрос цель которого чтобы выдача пользователю была релевантной.

Под номером 4.1 изначальный пользовательский ввод, можно заметить, что в нем введен вопрос, не существующий в таком виде в базе знаний «Уголовный кодекс в Древней Руси?», в базе присутствуют знание лишь по вопросу в формулировке «Как назывался свод законов в Древней Руси?». После выполнения этого запроса «как есть» на экране 5 мы бы получили бы пустую выборку (5.1). Однако проведем анализ исходного запроса (кнопка 4.5) — после ее нажатия программа запускает в работу алгоритмы 2.1 и 2.2. После этого мы видим граф синтаксического анализа 4.9, таблицу Именованых Сущностей под номером 4.7 — в данном примере модель ner\_tagger успешно выделила топоним «Древняя Русь» - который без изменений (кроме стемминга) будет в итоговом запросе. В окне 4.3 результаты действия по нажатию кнопки «Выделить синонимы» - после этого ко всем токенам, оставшимся после синтаксического анализа, применяется модель поиска семантически близких слов.

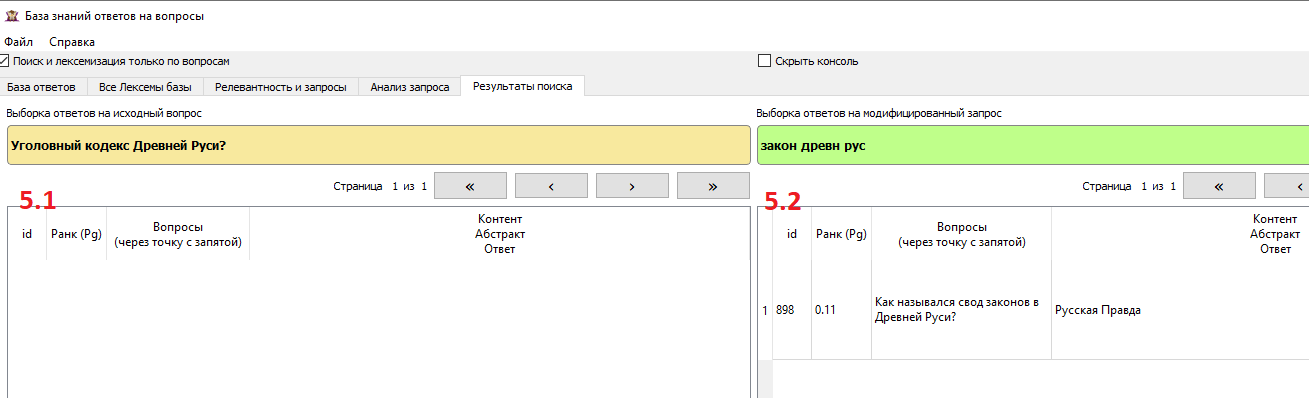
Работа по выделению синонимичных токенов в актуальной программе осуществляется моделью Word2Vec. В ходе данного исследования мной обучена модель 4corpora\_3,5Msentences.gz.bin на основе трех корпусов текста русского языка Лейпцигского университета [28] и архива статей Википедии за 2020 год проекта Corus [23]. В реализации использован пакет gensim. Обучение производилось со следующими значениями параметров параметрами: количество воркеров 8, размер вектора слова 300, максимальное количество токенов в словаре 500 000, контекстное окно 5, эпох 5. В алгоритме стохастического градиентного спуска скорость обучения α= 0.025.

Для оценки качества модели улавливает использован тест SemEval-2012 [14]. Тест представляет собой данные людьми (экспертами) оценки семантических соотношений для пар слов. Для меры качества модели используется корреляция Спирмана между экспертными оценками и полученными от обученной модели. Расстояние вычисляется как косинусное расстояние. Для сравнения использовались две модели проекта Natasha navec\_hudlit\_v112B\_500K\_300d\_100q и navec\_news\_v1\_1B\_250K\_300d\_100q [22]. Результаты расчетов представлены на рисунке 15.

Рисунок 15. Коэффициент Спирмана в тесте семантической близости для моделей по SemEval-2012

Как видно из пузырьковой диаграммы, лучше всего улавливает семантическую близость модель navec, обученная на корпусах из 12 миллиардов слов художественной литературы. Немаловажен размер точек на диаграмме. Он показывает, насколько часто при определении слов у модели случалась ошибка Out Of Vocabulary (OOV). Хотя обученная в процессе исследования модель 4corpora\_3,5Msentences иногда и давала результат порядка 0.75, что считается хорошим качеством, но в данном наборе слов удалось найти векторное представление слов в лучшем случае в 50% случаев. Результаты с OOV больше, чем 75%, отбрасывались, поэтому получили, что модель 4corpora\_3,5Msentences в конечных тестах вообще не смогла добиться значимых оценок семантической близости на данных тестовых наборах. Исходя из этого в актуальной программе была выбрана модель navec\_hudlit\_v112B\_500K\_300d\_100q.

Мы видим что модель нашла слову «кодекс», которое в исходном виде в лексемах не присутствует ([tf\_idf: 0.00]) синонимы «закон» ([tf\_idf: 0.68]) и «документ» ([tf\_idf: 0.10]), подменяем в исходном запросе слово «кодекс» «законом», оставляем только значимые лексемы («закон древн рус») и уже на этот вопрос находится релевантный ответ (5.2) на рисунке 16.

Рисунок 16. Экран результатов выборки

Таким образом видна практическая польза NLP анализа запросом перед отправкой в базу — такая предварительная техника во общем позволяет повысить точность выдачи в разных случаях до 3 раз.

## Тестирование качества алгоритма оптимизации запроса

Для оценки качества любого алгоритма ранжирования изначально эксперт в виде человека определяет целевые кейсы запросов, которым присваивается соответствуют ответы в системе базы знаний. Соответствие определяется численным показателем релевантности от 0 до 1, где 0 — не релевантный заданному ответу запрос и 1 — полностью релевантный ответ. Как было описано в разделе 1.5 в рамках данной работы мы ограничимся метриками, имеющими дела с двумя состояниями «релевантный» «не релевантный».

Было задано 10 запросов для теста. Вопрос строково идентичен вопросу в базе у двух запросов «Как назывался свод законов в Древней Руси?» и «Какого цвета язык у жирафа?». Убраны/изменены союзы, предлоги, окончания вопроса присутствующего в базе знаний у 2 запросов «Какова длинна кровеносных сосудов человека?» (убрано обобщение «всех»), «Сколько пузырьков углекислого газа содержит шампанское?» (изменены окончания и убран предлог «в»). Остальные 6 запросов перефразированы с использованием синонима или изменен порядок членов предложения.

В таблице 3 представлены результаты точности поиска на K=10 элементах (p@K) и средняя точность на K элементах (ap@K).

Таблица 3 – p@K и ap@K для поиска по вхождению строки, при K=10

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Запрос | Точное соответствие | Изменение | Перефразирование | p@K | ap@K |
| 1 | Какова длинна кровеносных сосудов человека? |  | + |  | 0.0 | 0.0 |
| 2 | Сколько в человеке кровеносных сосудов? |  |  | + | 0.0 | 0.0 |
| 3 | Какого цвета язык у жирафа? | + |  |  | 1.0 | 1.0 |
| 4 | Кто проживает на дне моря? |  |  | + | 0.0 | 0.0 |
| 5 | Кто живет на дне? |  |  | + | 0.0 | 0.0 |
| 6 | Сколько пузырьков углекислого газа содержит шампанское? |  | + |  | 0.0 | 0.0 |
| 7 | Как получить документ с оценками для иностранного вуза? |  |  | + | 0.0 | 0.0 |
| 8 | Не появились индивидуальные достижения в анкете абитуриента |  |  | + | 0.0 | 0.0 |
| 9 | Я преподаватель кафедры и не могу зайти на сайт и заполнить журнал преподавателя |  |  | + | 0.0 | 0.0 |
| 10 | Как назывался свод законов в Древней Руси? | + |  |  | 1.0 | 1.0 |

Средневзвешенная точность на K элементов для данного алгоритма поиска ответов по вхождению строки представлена на формуле 8:

|  |  |
| --- | --- |
|  | (8) |

Таблица 4 – p@K и ap@K для полнотекстового поиска Postgres, при K=10

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Запрос | Точное соответствие | Изменение | Перефразирование | p@K | ap@K |
| 1 | Какова длинна кровеносных сосудов человека? |  | + |  | 1.0 | 1.0 |
| 2 | Сколько в человеке кровеносных сосудов? |  |  | + | 0.0 | 0.0 |
| 3 | Какого цвета язык у жирафа? | + |  |  | 1.0 | 1.0 |
| 4 | Кто проживает на дне моря? |  |  | + | 0.0 | 0.0 |
| 5 | Кто живет на дне? |  |  | + | 0.0 | 0.0 |
| 6 | Сколько пузырьков углекислого газа содержит шампанское? |  | + |  | 1.0 | 1.0 |
| 7 | Как получить документ с оценками для иностранного вуза? |  |  | + | 0.0 | 0.0 |
| 8 | Не появились индивидуальные достижения в анкете абитуриента |  |  | + | 0.0 | 0.0 |
| 9 | Я преподаватель кафедры и не могу зайти на сайт и заполнить журнал преподавателя |  |  | + | 0.0 | 0.0 |
| 10 | Как назывался свод законов в Древней Руси? | + |  |  | 1.0 | 1.0 |

Средневзвешенная точность на K элементов для данного алгоритма полнотекстового поиска ответов представлена на формуле 9:

|  |  |
| --- | --- |
|  | (9) |

Таблица 5 – p@K и ap@K для полнотекстового поиска с NLP оптимизацией, при K=10

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Запрос | Точное соответствие | Изменение | Перефразирование | p@K | ap@K |
| 1 | Какова длинна кровеносных сосудов человека? |  | + |  | 1.0 | 1.0 |
| 2 | Сколько в человеке кровеносных сосудов? |  |  | + | 1.0 | 1.0 |
| 3 | Какого цвета язык у жирафа? | + |  |  | 1.0 | 1.0 |
| 4 | Кто проживает на дне моря? |  |  | + | 1.0 | 1.0 |
| 5 | Кто живет на дне? |  |  | + | 1.0 | 1.0 |
| 6 | Сколько пузырьков углекислого газа содержит шампанское? |  | + |  | 1.0 | 1.0 |
| 7 | Как получить документ с оценками для иностранного вуза? |  |  | + | 0.0 | 0.0 |
| 8 | Не появились индивидуальные достижения в анкете абитуриента |  |  | + | 1.0 | 1.0 |
| 9 | Я преподаватель кафедры и не могу зайти на сайт и заполнить журнал преподавателя |  |  | + | 0.33 | 0.61 |
| 10 | Как назывался свод законов в Древней Руси? | + |  |  | 1.0 | 1.0 |

Средневзвешенная точность на K элементов для данного алгоритма полнотекстового поиска ответов с NLP оптимизацией представлена на формуле 10:

|  |  |
| --- | --- |
|  | (10) |

Таким образом мы видим, что проведение дополнительного анализа пользовательского запроса повышает точность практически в два раза по сравнению с стандартным полнотекстовым поиском в базе знаний, и практически в 4 раза больше поиска не используя механизмы полнотекстового поиска Postgres.

## Описание технической реализации программы

Приложение состоит из файлов интерфейса ui, разработанных в Qt Designer, и непосредственно py-файлов, реализующих тот или иной функционал.

info\_service.ui – основной файл интерфейса, включает главное окно программы (см. рисунок 11).

info\_service\_del\_instance.ui — файл интерфейса диалогового окна для CRUD операции удаления.

info\_service\_new\_instance.ui — файл интерфейса диалогового окна для CRUD операции создания нового элемента.

main.py – файл, содержащий главный интерфейс программы. При старте программы дополнительные .py файлы логики, событий и инициализации из папок (actions, events, initializators) подгружаются к основному сеансу программы.

main.MainWindow – класс, реализующий главное окно. При старте в атрибутах класса инициализируется словарь ключ-значение для хранения потоко-безопасных данных. При старте программы создается три цикла обработки событий (Event Loop) - stdout\_write\_loop, program\_init\_loop, program\_actions\_loop и записываются в экземпляр данного класса.

stdout\_write\_loop — и \_forever\_run\_stdout\_write\_thread — не основной поток работы программы в который перенаправляется поток вывода любых обрабатываемых функций, который в последствии отображается в окне программы.

program\_actions\_loop — и \_forever\_run\_actions\_loop\_thread не основной поток (отличный от потока, в котором исполняется сам графический интерфейс) в который отправляются действия пользователя

program\_init\_loop — и \_program\_init\_thread поток, отрабатывающий один раз при страте программы, в этом потоке производятся первичная настройка и добавление событий к графическим элементам.

db\_base.py — в файле содержатся реквизиты доступа к СУБД PostgresSQL и модели данных QuestAnswerBase, QuestAnswerBaseRelevQuery. В модели данных QuestAnswerBase в качестве колонки-индекса полнотектового поиска заданы две колонки, представляющие тип данных TSVector: “search\_vector” и “q\_search\_vector”. Колонка search\_vector представляет собой вектор полнотекстового поиска, который лексимизирует колонки questions→abstract, т. е. большего веса при запросе полнотекстового поиска будет иметь лексемы вопросов, в меньшей мере лексемы самого ответа, q\_search\_vector же лексимизирует только текст вопроса(ов).

В подпапке actions осуществляется обработка действий, составляющих логику работы программы, описание основных:

actions\db\_list\_search\_entries.py — в этом файле описание функции полнотекстовго поиска. На вход подается пользовательский запрос, флаг о необходимости сортировки выдачи по ранку, и флаг поиска только среди колонки вопросов (без этого параметра поиск будет осуществлятся по tsvector который учитывает и абстракт ответа по весу второй категории важности после вопроса)

actions\query\_optimize\_query.py — в этом файле содержится функции ru\_stemmer — «Snowball» стеммер токенов [31], get\_has\_in\_postgres\_TF\_IDF — функция определяющая метрику TF\_IDF по формуле (4) определенного токена в рамках базы знаний [10], которые вызываются в ходе работы основной логики алгоритмов 2.1 и 2.2 (Рисунок 9, 10). Экземпляры morph\_vocab — словарь для нормализации токенов pymorphy2, segmenter — функция разбиения предложения на токены, emb — представление токена вектором Navec что требуется дальнейшими NLP моделями. Модели syntax\_parser, morph\_tagger, ner\_tagger, bigram\_reloaded — соответственно синтаксический анализатор, добавляющий дополнительную разметку синтаксической связи и членов предложения, морфологический анализатор размечает части речи, анализатор именованых сущностей помечают токен или токены которые относятся к именам собственным или нарицательным существительным, а также статистическая модель определения парафраз в тексте. Работа функции есть цепочка двух алгоритмов описанная в разделе 2.1 и 2.2. Результатом работы функции является строка, к которой применены алгоритмы оптимизации, кроме того в случае вызова из графического интерфейса — то заполняются соотвествующие поля в экранах программы (граф синтаксического анализа и сам запрос в зеленом поле).

Файлы с названием actions\metrics\_\* отвечают за расчет метрик качества представления результатов полнотекстового поиска:

actions\metrics\_getTopKrelevN.py — расчет топ-K элементов (требующихся для расчета MAP@K), функция поочередно применяет полнотектовый поиск с/без оптимизации к набору релевантных/нерелевантных запросов из тестирования, и возвращает сумму релевантных элементов в выдаче (исходя из отмеченной доли релевантности) функцией actions\metrics\_rel01\_by\_text.py.

actions\metrics\_getPrecisionAtK.py — это метрика простой точности на K результатах (p@k) описанная в формуле (1)

actions\metrics\_getAvgPrecisionOfKres.py — средняя точность на K результатах (ap@K) описанная в формуле (2).

actions\metrics\_getMeanAvgPrecisionAtK.py — средневзвещенная точность поиска на K элементах (map@K) описанная в формуле (3)

Заключение

Поставленные задачи были решены в полном объеме, а исследование имеет большую теоретическую и практическую значимость.

Предложенный метод оптимизации пользовательского запроса к базе знаний практически в два раза повышает точность выдачи. Поиск по строке не приспособлен для QA-системы на естественном языке т. к. имеет недостаточную точность. Стандартный механизм полнотекстового поиска PostgresSQL, основанный на лексемизации слов документа и составления индекса слов по уровню статистической значимости (важности) в чистом виде пригоден для QA системы, однако на его точность в значительной мере влияют шумы, то есть токены пользователя которые семантически не относятся к основным в запросе. Кроме того, стандартный полнотекстовый поиск не работает с контекстными синонимами в запросе, что в определенных случаях может понизить точность выборки. Предложенные алгоритмы используют для синтаксического и морфологического анализа WordCNN-CRF модели семейства Slovnet. Для вычисления семантической близости и векторизации слов в работе используются Navec модель, обученная на корпусе порядка 12 млрд слов художественного или 1 млрд слов новостного текста. Для выявления словосочетаний в ходе работы на корпусах Лейпцигского университета была обучена простая статистическая модель парафраз.

Предложенный и исследованный метод анализа пользовательского запроса в дальнейшем может быть использован для генерации RAG контекста к современным LLM для создания виртуального ассистента для студентов ТОГУ.

Список использованных источников

1. Sephton P. Полнотекстовый поиск в PostgreSQL // Текст: электронный // pgdocs.ru [Электронный ресурс]. URL: https://pgdocs.ru/docs/fullsearch.html. (дата обращения 25.05.2025)

2. Инженерия знаний [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Инженерия\_знаний. (дата обращения 25.05.2025)

3. Systems and software engineering —Vocabulary № ISO/IEC/IEEE 24765:2010(E).

4. Чистов Д. В. “Новые информационные технологии в образовании: применение технологий “1С” для развития компетенций цифровой экономики” // научно-практическая конференция «Новые информационные технологии в образовании»: Сборник — 2018.

5. База знаний [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/База\_знаний. (дата обращения 25.05.2025)

6. Язык [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Язык.

7. Компьютерная лингвистика [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Компьютерная\_лингвистика. (дата обращения 25.05.2025)

8. Полнотекстовый поиск // Текст: электронный // habr.com [Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/Полнотекстовый\_поиск.

9. Abbas H. “15 Free Full-Text Search Engine Solutions for developers” // Текст: электронный // medevel.com [Электронный ресурс]. URL: https://medevel.com/os-fulltext-search-solutions/. (дата обращения 25.05.2025)

10. Основы полнотекстового поиска в PostgreSQL // Текст: электронный // pgdocs.ru [Электронный ресурс]. URL: https://eax.me/postgresql-full-text-search/. (дата обращения 25.05.2025)

11. Учебник по обработке естественного языка или NLP // Текст: электронный // gorbachev.am [Электронный ресурс]. URL: https://www.gorbachev.am/files/library/nlp/NLP Tutorial.pdf. (дата обращения 25.05.2025)

12. Postgres Pro - Словари // Текст: электронный // postgrespro.ru [Электронный ресурс]. URL: https://postgrespro.ru/docs/postgresql/15/textsearch-dictionaries. (дата обращения 25.05.2025)

13. Метрики качества ранжирования // Текст: электронный // habr.com [Электронный ресурс]. URL: https://habr.com/ru/companies/econtenta/articles/303458/. (дата обращения 25.05.2025)

14. SemEval-2012: Semantic Evaluation Exercises. Task 2: Measuring Degrees of Relational Similarity [Электронный ресурс]. URL: https://sites.google.com/site/ semeval2012task2/home?authuser=0. (дата обращения 25.05.2025)

15. Объясняем простым языком, что такое трансформеры [Электронный ресурс]. URL: https://habr.com/ru/companies/mws/articles/770202/. (дата обращения 25.05.2025)

16. Ashish Vaswani, Noam Shazeer, Niki Parmar “Attention Is All You Need” // Текст: электронный // arXiv [Электронный ресурс]. URL: https://arxiv.org/pdf/1706.03762.pdf. (дата обращения 25.05.2025)

17. Рябинин М. “Учебник по машинному обучению - Трансформеры” [Электронный ресурс]. URL: https://education.yandex.ru/handbook/ml/article/transformery. (дата обращения 25.05.2025)

18. Bahdanau D. “Neural Machine Translation by Jointly Learning to Align and Translate” // Текст: электронный // arXiv [Электронный ресурс]. URL: https://arxiv.org/pdf/1409.0473.pdf. (дата обращения 25.05.2025)

19. Крылов В. “Что такое эмбеддинги и как они помогают искусственному интеллекту понять мир людей” // Текст : непосредственный // Наука и Жизнь. 2023. № 9 (399). C. 10–11.

20. Tomas Mikolov Kai Chen G. C. “Efficient Estimation of Word Representations in Vector Space” // Текст: электронный // arXiv [Электронный ресурс]. URL: https://arxiv.org/abs/1301.3781. (дата обращения 25.05.2025)

21. Mikolov T. “Distributed Representations of Words and Phrases and their Compositionality” // Текст: электронный // arXiv [Электронный ресурс]. URL: https://arxiv.org/pdf/1310.4546.pdf. (дата обращения 25.05.2025)

22. Кукушкин А. “Navec — компактные эмбеддинги для русского языка” // Текст: электронный // natasha.github.io [Электронный ресурс]. URL: https://natasha.github.io/navec/. (дата обращения 25.05.2025)

23. Кукушкин А. “Corus - набор корпусов текста русского языка” // Текст: электронный // natasha.github.io [Электронный ресурс]. URL: https://github.com/natasha/corus. (дата обращения 25.05.2025)

24. Гладышев В. В. “Средства машинной обработки русского языка” // Молодой ученый. 2022. № 4 (399). C. 7–9.

25. Кукушкин А. “Проект Natasha. Набор качественных открытых инструментов для обработки естественного русского языка (NLP)” [Электронный ресурс]. URL: https://habr.com/ru/articles/516098/. (дата обращения 25.05.2025)

26. Кукушкин А. “SlovNet - библиотека Python для моделирования NLP на основе глубокого обучения для русского языка” // Текст: электронный // natasha.github.io [Электронный ресурс]. URL: https://github.com/natasha/slovnet. (дата обращения 25.05.2025)

27. Прохоренок Н. А., Дронов В. А. Python 3 и PyQt 5. Разработка приложений / Прохоренок Н. А., Дронов В. А., 2-е изд., перераб. и доп-е изд., СПб : БХВ-Петербург, 2018.

28. Leipzig Corpora Collection // Текст: электронный // wortschatz.uni-leipzig.de. (дата обращения 25.05.2025)

29. Gensim is Python library // Текст: электронный // radimrehurek.com [Электронный ресурс]. URL: https://radimrehurek.com/gensim/. (дата обращения 25.05.2025)

30. Добавление собственных данных в LLM с помощью RAG // Текст: электронный // habr.com [Электронный ресурс]. URL: https://habr.com/ru/companies/wunderfund/articles/779748/. (дата обращения 25.05.2025)

31. Портер М. Russian stemming algorithm // Текст: электронный // snowball.tartarus.org [Электронный ресурс]. URL: http://snowball.tartarus.org/algorithms/russian/stemmer.html. (дата обращения 25.05.2025)

32. Каряева М. С., Браславский П. И., Соколов В. А. Векторное представление слов с семантическими отношениями: экспериментальные наблюдения // Моделирование и анализ информационных систем. – 2018. – Т. 25, № 6(78). – С. 726-733. – DOI 10.18255/1818-1015-2018-6-726-733.

33. Булыга Ф. С., Курейчик В. М. Сравнительный анализ методов векторизации текстовых данных большой размерности // Известия ЮФУ. Технические науки. – 2023. – № 2(232). – С. 212-226. – DOI 10.18522/2311-3103-2023-2-212-226.

Приложение А

(обязательное)

#!/usr/bin/env python

import threading

import asyncio

import time

import os

import sys

import glob

import logging

import pathlib

from importlib import import\_module

from inspect import isfunction, iscoroutinefunction

from PyQt5 import QtWebEngineWidgets

from PyQt5 import QtCore, QtGui, QtWidgets

from PyQt5 import uic # , pyrcc

from PyQt5.QtWebEngineWidgets import QWebEngineView # working with PyQt5 5.13.1

if getattr(sys, 'frozen', False) and hasattr(sys, '\_MEIPASS'):

os.environ["PYMORPHY2\_DICT\_PATH"] = str(pathlib.Path(sys.\_MEIPASS).joinpath('pymorphy2\_dicts\_ru/data'))

MAIN\_PACKAGE\_DIR = os.path.abspath(os.path.join(os.path.split(str(\_\_file\_\_))[0]))

PACKAGE\_NAME = os.path.basename(MAIN\_PACKAGE\_DIR)

sys.path.append(MAIN\_PACKAGE\_DIR)

sys.path.append(os.path.abspath(os.path.join(os.path.split(str(\_\_file\_\_))[0], '..')))

# Импорт корневых модулей программы

import info\_service.db\_base

import info\_service.db\_utils

# / Импорт корневых модулей программы

\_LOG\_LAST\_TIMESTAMP = 0

\_LOG\_SAFE\_QT\_BUFFER = ''

\_LOG\_SAFE\_QT\_BUFFER\_INFIELD = ''

\_clearing\_field\_flag = False

\_all\_inits\_runned = False

# Импорт динамических провайдеров, и их доимпортирование специально для pyInstaller

from info\_service.actions import DYN\_FUNC\_PROVIDERS as ACTION\_PROVIDERS

if not len(ACTION\_PROVIDERS):

\_modules\_actions = glob.glob(os.path.join(MAIN\_PACKAGE\_DIR, 'actions', "\*.py"))

\_modules\_actions = [os.path.basename(x)[:-3] for x in \_modules\_actions if os.path.isfile(x)]

\_modules\_actions = [x for x in \_modules\_actions if not x.startswith('\_')]

for \_module\_name in \_modules\_actions:

\_modul = import\_module(f'{PACKAGE\_NAME}.actions.{\_module\_name}')

\_provider = getattr(\_modul, str('main'), None)

if not \_provider:

\_provider = getattr(\_modul, str(\_module\_name), None)

if isfunction(\_provider):

ACTION\_PROVIDERS[\_module\_name] = \_provider

setattr(sys.modules[f'{PACKAGE\_NAME}.actions'], \_module\_name, \_provider)

from info\_service.initializators import DYN\_FUNC\_PROVIDERS as INIT\_PROVIDERS

if not len(INIT\_PROVIDERS):

\_modules\_initializators = glob.glob(os.path.join(MAIN\_PACKAGE\_DIR, 'initializators', "\*.py"))

\_modules\_initializators = [os.path.basename(x)[:-3] for x in \_modules\_initializators if os.path.isfile(x)]

\_modules\_initializators = [x for x in \_modules\_initializators if not x.startswith('\_')]

for \_module\_name in \_modules\_initializators:

\_modul = import\_module(f'{PACKAGE\_NAME}.initializators.{\_module\_name}')

\_provider = getattr(\_modul, str('main'), None)

if not \_provider:

\_provider = getattr(\_modul, str(\_module\_name), None)

if isfunction(\_provider):

INIT\_PROVIDERS[\_module\_name] = \_provider

setattr(sys.modules[f'{PACKAGE\_NAME}.initializators'], \_module\_name, \_provider)

from info\_service.events import DYN\_FUNC\_PROVIDERS as EVENT\_PROVIDERS

if not len(EVENT\_PROVIDERS):

\_modules\_events = glob.glob(os.path.join(MAIN\_PACKAGE\_DIR, 'events', "\*.py"))

\_modules\_events = [os.path.basename(x)[:-3] for x in \_modules\_events if os.path.isfile(x)]

\_modules\_events = [x for x in \_modules\_events if not x.startswith('\_')]

for \_module\_name in \_modules\_events:

\_modul = import\_module(f'{PACKAGE\_NAME}.events.{\_module\_name}')

\_provider = getattr(\_modul, str('main'), None)

if not \_provider:

\_provider = getattr(\_modul, str(\_module\_name), None)

if isfunction(\_provider):

EVENT\_PROVIDERS[\_module\_name] = \_provider

setattr(sys.modules[f'{PACKAGE\_NAME}.events'], \_module\_name, \_provider)

# / Импорт динамических провайдеров, и их доимпортирование специально для pyInstaller

class SecondWindow(QtWidgets.QDialog):

def \_\_init\_\_(self, dialog\_ui=None):

super().\_\_init\_\_()

self.setWindowTitle('Диалоговое окно')

super(SecondWindow, self).\_\_init\_\_()

if not dialog\_ui:

self.setGeometry(100, 100, 200, 100)

else:

uic.loadUi(dialog\_ui, self)

time.sleep(0.2)

class MainWindow(QtWidgets.QMainWindow):

"""

Основное универсальное окно програмы

"""

def \_\_init\_\_(self):

"""

Загрузка инфтерфейса, присоединение обработчиков событий

"""

self.MAINWINDOW\_LOCAL\_STORAGE = {}

super(MainWindow, self).\_\_init\_\_()

uic.loadUi(f'{PACKAGE\_NAME}.ui', self)

self.\_load\_events\_handlers()

time.sleep(0.5)

def open\_second\_window(self, init\_callback=None, ok\_callback=None, cancel\_callback=None, dialog\_ui=None):

"""

Диалоговое окно

:param init\_callback:

:param ok\_callback:

:param cancel\_callback:

:param dialog\_ui:

"""

second\_window = SecondWindow(dialog\_ui=dialog\_ui)

setattr(second\_window, 'main\_window', self)

if init\_callback:

setattr(second\_window.\_\_class\_\_, f'init\_callback', init\_callback)

init\_callback(second\_window)

if ok\_callback:

setattr(second\_window.\_\_class\_\_, f'ok\_callback', ok\_callback)

second\_window.accepted.connect(getattr(second\_window, f'ok\_callback'))

if cancel\_callback:

setattr(second\_window.\_\_class\_\_, f'cancel\_callback', cancel\_callback)

second\_window.rejected.connect(getattr(second\_window, f'cancel\_callback'))

second\_window.exec\_()

def closeEvent(self, event):

"""

Закрытие программы, закрытие потоков, плохо работает

:param event:

"""

print('closing..')

window.stdout\_write\_loop = None

window.program\_init\_loop = None

window.program\_actions\_loop = None

self.\_program\_init\_thread = None

self.\_forever\_run\_stdout\_write\_thread = None

self.\_forever\_run\_actions\_loop\_thread = None

time.sleep(0.2)

event.accept()

def \_load\_events\_handlers(self):

"""

Присоединение обработчиков событий к окну

"""

for event\_name, callable\_ in EVENT\_PROVIDERS.items():

callable\_(self)

print(f'\n+++>"{event\_name}" handlers connected to MainWindow... ')

async def \_init\_program\_event(self):

"""

Инициализаторы исполняющиеся в параллельном потоке программы

"""

global \_LOG\_LAST\_TIMESTAMP, \_all\_inits\_runned

ui\_disable\_clear\_console = ACTION\_PROVIDERS.get('ui\_disable\_clear\_console', None)

if ui\_disable\_clear\_console:

ui\_disable\_clear\_console(self)

for init\_name, callable\_ in INIT\_PROVIDERS.items():

if iscoroutinefunction(callable\_):

await callable\_(self)

else:

clcor = asyncio.coroutine(callable\_)

await clcor(self)

\_LOG\_LAST\_TIMESTAMP = 0

print(f'\n---> "{init\_name}" init executed... \n')

\_LOG\_LAST\_TIMESTAMP = 0

print(f'\n--->>> ALL init executed... \n')

ui\_enable\_clear\_console = ACTION\_PROVIDERS.get('ui\_enable\_clear\_console', None)

if ui\_enable\_clear\_console:

ui\_enable\_clear\_console(self)

\_all\_inits\_runned = True

def write\_to\_window\_s(qtmain\_wind, message, set\_to\_field=False):

"""

Функция обработки sys.stdout.write(), перегрузка глобальная перегрузка вывода для присоединенных процедур

:param qtmain\_wind:

:param message:

:param set\_to\_field:

"""

assert message

global \_LOG\_LAST\_TIMESTAMP, \_LOG\_SAFE\_QT\_BUFFER, \_LOG\_SAFE\_QT\_BUFFER\_INFIELD, \_clearing\_field\_flag

if not message:

return

# Очитка поле после 1000 строк. После стирания в виджете QT подождем

if \_LOG\_SAFE\_QT\_BUFFER\_INFIELD.count('\n') > 1000 or message == '[main\_window.clear\_console]':

\_clearing\_field\_flag = True

\_LOG\_SAFE\_QT\_BUFFER = ''

\_LOG\_SAFE\_QT\_BUFFER\_INFIELD = ''

qtmain\_wind.TextConsoleView.clear()

time.sleep(6.0)

\_LOG\_LAST\_TIMESTAMP = 0

\_clearing\_field\_flag = False

\_LOG\_SAFE\_QT\_BUFFER = f'{\_LOG\_SAFE\_QT\_BUFFER}{message}'

if set\_to\_field and not \_clearing\_field\_flag:

\_LOG\_LAST\_TIMESTAMP = time.time()

allen = len(\_LOG\_SAFE\_QT\_BUFFER)

inflen = len(\_LOG\_SAFE\_QT\_BUFFER\_INFIELD)

if allen > inflen:

insert = \_LOG\_SAFE\_QT\_BUFFER[-(allen - inflen):]

# Не забивать в insertPlainText больше 800 символов за раз

chunks = [insert[i:i + 800] for i in range(0, len(insert), 800)]

for cnk in chunks:

if not cnk:

continue

\_LOG\_SAFE\_QT\_BUFFER\_INFIELD = f'{\_LOG\_SAFE\_QT\_BUFFER\_INFIELD}{cnk}'

qtmain\_wind.TextConsoleView.insertPlainText((' ' if cnk == '\n' else '') + cnk)

# Кажется что без задержек у меня все вылетает, но это стоит потом отладить, заебался уже. работает - не трогай!

time.sleep(0.5)

if \_\_name\_\_ == '\_\_main\_\_':

app = QtWidgets.QApplication(sys.argv)

window = MainWindow()

window.show()

time.sleep(0.1)

stdout\_write\_loop = asyncio.new\_event\_loop()

program\_init\_loop = asyncio.new\_event\_loop()

program\_actions\_loop = asyncio.new\_event\_loop()

window.stdout\_write\_loop = stdout\_write\_loop

window.program\_init\_loop = program\_init\_loop

window.program\_actions\_loop = program\_actions\_loop

\_forever\_run\_stdout\_write\_thread = None

\_program\_init\_thread = None

\_forever\_run\_actions\_loop\_thread = None

window.\_forever\_run\_stdout\_write\_thread = None

window.\_program\_init\_thread = None

window.\_forever\_run\_actions\_loop\_thread = None

def forever\_run\_stdout\_write\_thread\_target():

"""

Евентлуп треда для принта

https://stackoverflow.com/questions/29692250/restarting-a-thread-in-python

"""

global window

while True:

try:

asyncio.set\_event\_loop(stdout\_write\_loop)

stdout\_write\_loop.run\_forever()

except BaseException as e:

print('{!r}; restarting thread'.format(e))

else:

print('exited normally, bad thread; restarting')

def program\_event\_thread\_target():

"""

Евентлуп треда загрузки инициализаторов

"""

global window

program\_init\_loop.run\_until\_complete(window.\_init\_program\_event())

def forever\_run\_actions\_thread\_target():

"""

Евентлуп треда для действий иницированных элементами гуя

Исполнять неблокирующее ассинхронное действие в эвентпуле отдельного потока:

asyncio.run\_coroutine\_threadsafe(coroutine(), window.program\_actions\_loop)

"""

global window

while True:

try:

asyncio.set\_event\_loop(program\_actions\_loop)

program\_actions\_loop.run\_forever()

except BaseException as e:

print('{!r}; restarting thread'.format(e))

else:

print('exited normally, bad thread; restarting')

class SysRedirect(object):

"""

Класс-перегрузка sys.stdout

"""

def \_\_init\_\_(self, qtmain\_wind):

self.main\_thread = threading.currentThread()

self.write\_thread = self.main\_thread

self.qtmain\_wind = qtmain\_wind

self.terminal = sys.stdout # To continue writing to terminal

def write(self, message):

global \_LOG\_LAST\_TIMESTAMP, window

self.write\_thread = threading.currentThread()

if self.terminal:

self.terminal.write(message)

if message:

try:

message = str(message)

set\_to\_field = False

if time.time() - \_LOG\_LAST\_TIMESTAMP > 2:

set\_to\_field = True

handle = stdout\_write\_loop.call\_soon\_threadsafe(write\_to\_window\_s, self.qtmain\_wind, message, set\_to\_field, context=None)

except Exception as e:

if self.terminal:

self.terminal.write(f'Write exception: {e!r}')

def flush(self):

# this flush method is needed for python 3 compatibility.

# this handles the flush command by doing nothing.

# you might want to specify some extra behavior here.

stdout\_write\_loop.call\_soon\_threadsafe(stdout\_write\_loop.stop)

pass

sys.stdout = SysRedirect(window)

logger = logging.getLogger(f'{PACKAGE\_NAME}')

logger.setLevel(logging.DEBUG)

logger.addHandler(logging.StreamHandler(stream=sys.stdout))

\_forever\_run\_stdout\_write\_thread = threading.Thread(target=forever\_run\_stdout\_write\_thread\_target, daemon=True)

window.\_forever\_run\_stdout\_write\_thread = \_forever\_run\_stdout\_write\_thread

window.\_forever\_run\_stdout\_write\_thread.start()

\_forever\_run\_actions\_loop\_thread = threading.Thread(target=forever\_run\_actions\_thread\_target, daemon=True)

window.\_forever\_run\_actions\_loop\_thread = \_forever\_run\_actions\_loop\_thread

window.\_forever\_run\_actions\_loop\_thread.start()

\_program\_init\_thread = threading.Thread(target=program\_event\_thread\_target, daemon=True)

window.\_program\_init\_thread = \_program\_init\_thread

window.\_program\_init\_thread.start()

# Подождем пока последний созданный поток не отработает все иниты

sleeper\_count = 0

while not \_all\_inits\_runned and sleeper\_count < 1:

time.sleep(0.33)

sleeper\_count += 1

del window # Создадим запрос на удаление window но он из-за связанности с app не разрушится в момент работы app,

# но зато не произведет ошибку при выходе из программы (т.к. внутри QT ссылка на созданный поток)

qtapp = app.exec() # Блокирующая операция

time.sleep(0.2)

\_forever\_run\_stdout\_write\_thread.join()

time.sleep(0.1)

\_forever\_run\_actions\_loop\_thread.join()

time.sleep(0.1)

\_program\_init\_thread.join()

time.sleep(0.1)

# del \_forever\_run\_stdout\_write\_thread

# del \_forever\_run\_actions\_loop\_thread

# del \_program\_init\_thread

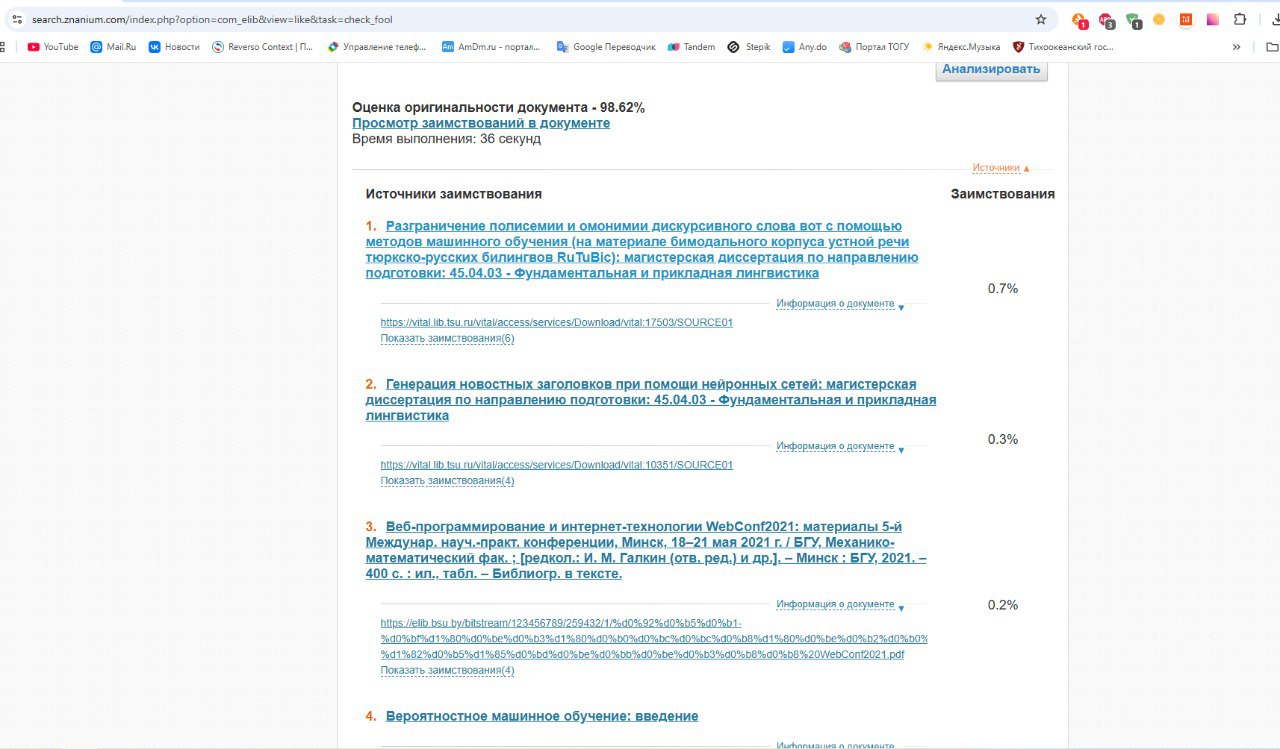
time.sleep(1)

del qtapp

sys.exit()

Приложение Б

(справочное)



https://search.znanium.com/index.php?option=com\_elib&view=like&lang=ru