## Development of a QA system with semantic search

Keywords: RNN, Word2Vec, Python, vector space, embedding, embedding, NLP, QA, semantic proximity

The subject of the article is the development of a question-and-answer knowledge base system at Pacific State University. The study of search quality in various variations — with simple search by text occurrence, indexing based on the "frequency importance" of words in a document, as well as using search query optimization based on semantic proximity and syntactic importance of search query sentence members based on RNN artificial neural networks of the Word2vec and GloVe families.

## Разработка QA системы с семантическим поиском"

Ключевые слова: RNN, Word2Vec, Python, векторное пространство, embedding, вложения, NLP, QA, семантическая близость

Предметом статьи является разработка вопросно-ответной системы базы знаний Тихоокеанского Государственного Университета. 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 на основе RNN искусственных нейронных сетей семейств Word2vec и GloVe.

## Термины и определения

QA – Вопросно-ответная система.

RNN – Рекуррентные нейронные сети.

OHE – One-Hot Encoding.

TF-IDF – обратная частота распространенности.

Word2Vec – семейство моделей для векторизации слов

Skip-gram – словосочетание с пропуском

## Введение

Вопросно-ответная система (QA-система; от англ. QA — англ. Question-answering system) — информационная система, способная принимать вопросы и отвечать на них на естественном языке. Актуальность разработки подобной системы 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.

Статья организована следующим образом. Раздел 2 представляет описание теоретической основы разработанных алгоритмов, раздел 3 содержит описание алгоритмов разработанного приложения и задействованных технологий, а раздел 4 сравнительный анализ эффективности разработанных алгоритмов, примененных к задаче поиска. Заключение, итоги, потенциальные направления будущих исследований и разработок представлены в разделе 5.

## Описание алгоритмов разработанного приложения и задействованных технологий

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. В 2001 году идея использовать ИНС для анализа текста привела к рождению одной из первых embedding-моделей NLP. Однако, как модели представить слово какого-то языка в этом численном виде чтобы оперировать им? В русскоязычной литературе такие числовые векторы называются эмбеддингами, которые получены из слов или других языковых сущностей. 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, такие эмбеддинги называют One-Hot Encoding (OHE) ембеддинги [1]. Как правило в чистом виде такие векторные представления редко используются т.к. вектор не хранит семантической информации о контексте используемого слова, а также необходимости переобновлять всю нумерацию пространства после добавления слова в словарь.

Следующим шагом в теории составлении векторных представлений стал «частотный эмбеддинг», в котором каждому слову в позицию, соответствующую его номеру, ставится в соответствие число - частота слова в корпусе и документе, а точнее, скорректированное значение частоты: формулу TF-IDF. Эта аббревиатура означает "term frequency — inverse document frequency.

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

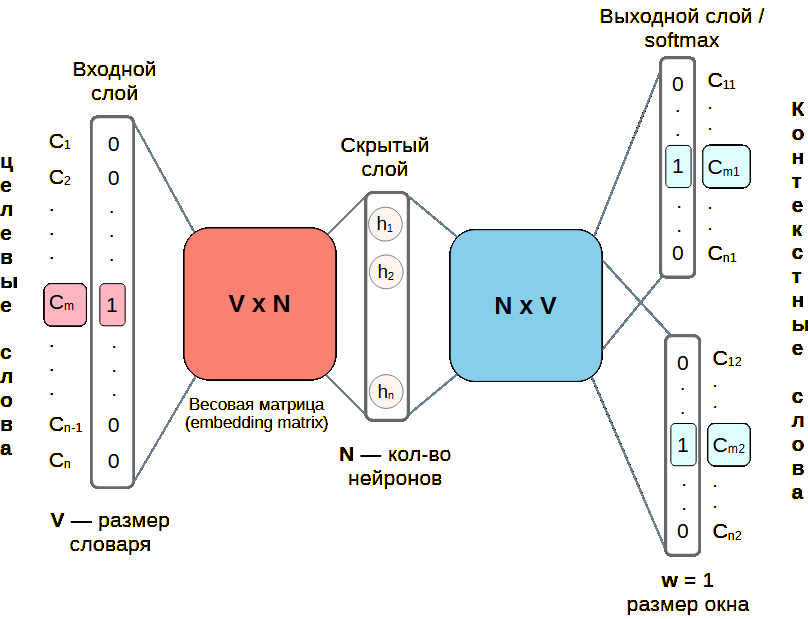
где TF — это частота слова в тексте. IDF — существенно более интересная вещь: это логарифм обратной частоты распространенности слова в корпусе (коллекции текстов). Показатель обратной частоты будет выше, если определённое слово с большой частотой используется в конкретном тексте, но редко - в других 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.

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. В 2013 году 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 [1]. 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 [2, 3].

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

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. 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».

Вместо алгоритма непрерывного мешка слов, используемого для предсказания следующего слова по нескольким предыдущим по контексту, модель Word2Vec использует Skip-gram (словосочетание с пропуском). Цель этой модели прямо противоположная предыдущей модели – предсказать окружающие слова на основе центрального.

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

Архитектура модели представлена на рисунке 1. 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 веса после обучения и моделью предсказываются векторы, которые соответствуют словам близким по контексту входному — то есть при обучении находившихся слева и с права в тексте (окно w=1). Для Skip-gram модели объективная функция трансформируется, относительно CBOW, в сумму логарифмической вероятности окружающих n-искомых слов вокруг целевого слова [2]:

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

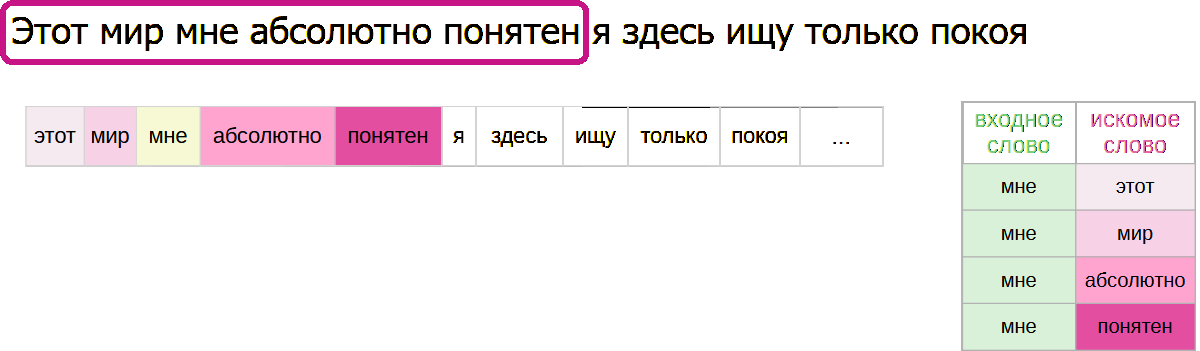


Рисунок 2. Принцип составления датасета для получения Skip-gram-модели Word2Vec для корпуса текста (рамка — контекстное окно, градации цвета - вероятности)

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. В программе разрабатываемой QA-системы 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 — поэтому достаточно моделей семейства Word2Vec и дистиллированных WordCNN моделей. Для программы разработано два алгоритма, использующих Word2Vec.

1. Алгоритм синтаксического анализа запроса, выявление основной части запроса. На рисунке 3 представлена блок-схема алгоритма

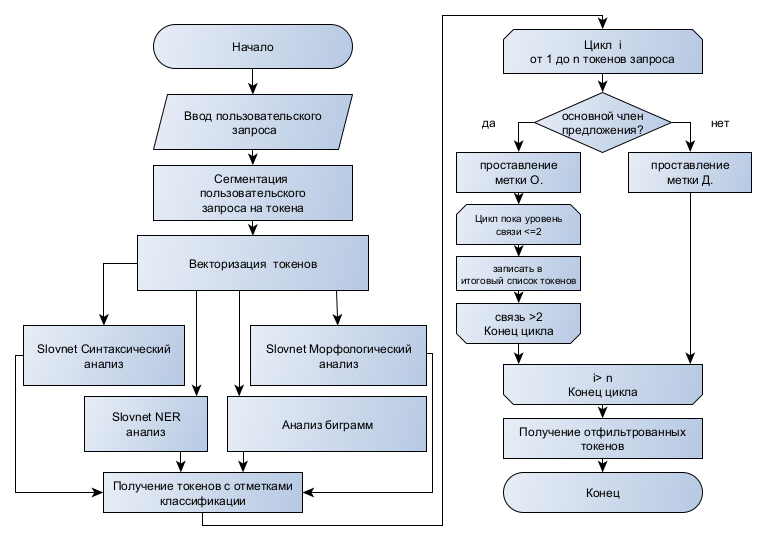


Рисунок 3. Алгоритм синтаксического анализа запроса, выявление основной части

* 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 повышенную омонимичность с другими морфемами слов что вносит излишнюю энтропию в анализ.
* После сегментации на токены — документ (запрос) векторизуются 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».
* После векторизации параллельно происходит применение к документу (с уже векторизованными 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 именованных сущностей (NER) двумя моделями. Для выявления биграмм в рамках данной работы была обучена статистическая модель на корпусе текстов проекта «Leipzig Corpora Collection» [5] и «Corus» [6]. Для выявления именованных сущностей используется предобученная модель из проекта Slovnet [7]. В токены документа которые относятся к биграммам 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 именованных сущностей выше чем нарицательных существительных, NER, позднее в алгоритме оптимизации NER точно также, как и словосочетания исключаются из оптимизаций.
* Полученные 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 — то есть основные и дополнительные члены предложения — подлежащее, сказуемое, одно определение или дополнение

1. Алгоритм оптимизации по семантической близости и TF-IDF. На рисунке 4 представлена блок-схема алгоритма

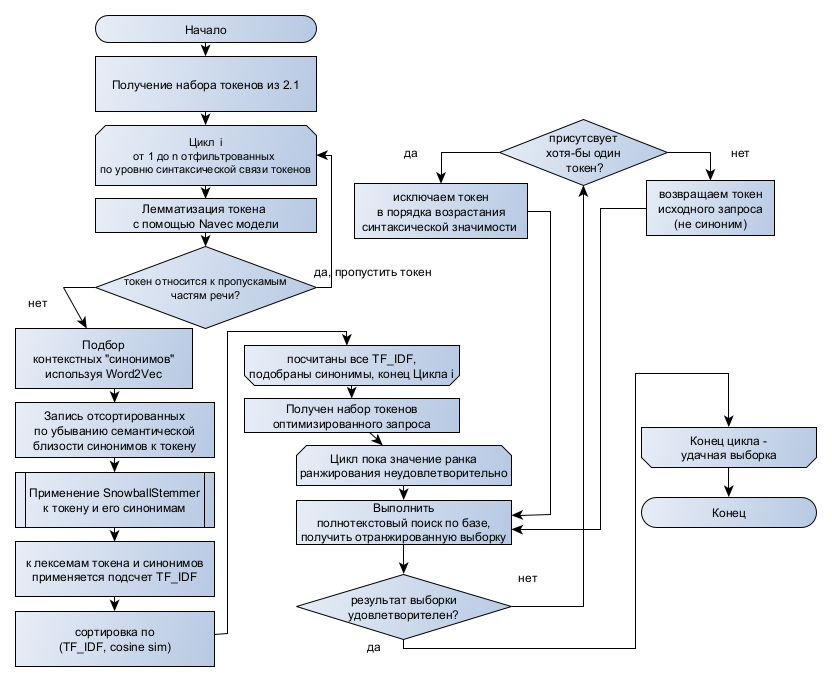


Рисунок 4. Алгоритм оптимизации по семантической близости и TF-IDF

* 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, оставляя 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 записываются 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 ключом 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 оптимизированный список лексем которые имеют наибольшую среднюю меру TF\_IDF нашей базы знаний.
* Полученный 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.

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.

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 полнотекстового 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 осуществляется 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 [8], get\_has\_in\_postgres\_TF\_IDF — функция, определяющая метрику TF\_IDF по формуле (1) определенного токена в рамках базы знаний [9], которые вызываются в ходе работы основной логики алгоритмов (Рисунок 3, 4). 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 именованных 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. Результатом работы функции является строка, к которой применены алгоритмы оптимизации, кроме того в случае вызова из графического интерфейса — то заполняются соответствующие 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:

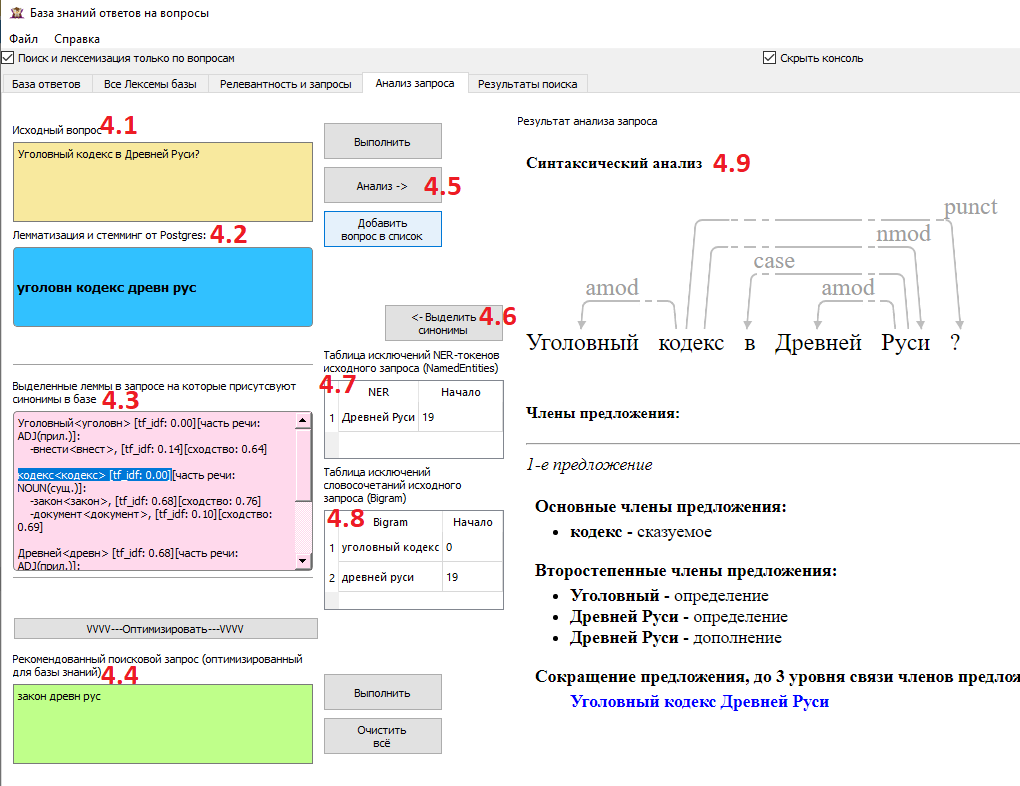
actions\metrics\_getTopKrelevN.py — расчет топ-K элементов (требующихся для расчета MAP@K), функция поочередно применяет полнотекстовый поиск с/без оптимизации к набору релевантных/нерелевантных запросов из тестирования, и возвращает сумму релевантных элементов в выдаче (исходя из отмеченной доли релевантности)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

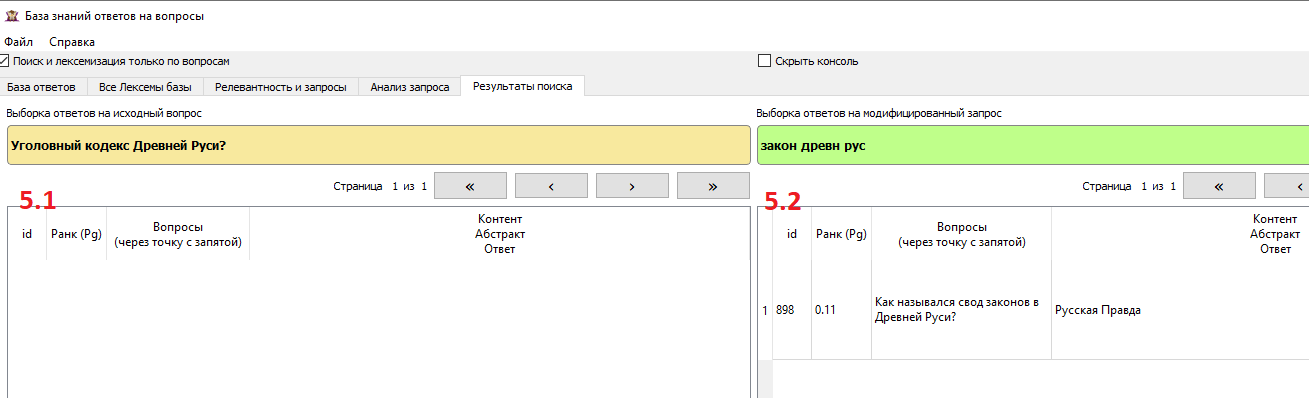
actions\metrics\_getPrecisionAtK.py — это метрика простой точности на K результатах (p@k) описанная в формуле (4)

actions\metrics\_getAvgPrecisionOfKres.py — средняя точность на K результатах (ap@K) описанная в формуле (5).

actions\metrics\_getMeanAvgPrecisionAtK.py — средневзвешенная точность поиска на K элементах (map@K) описанная в формуле (6)

Интерфейс (экран анализа запроса и результатов) программы показан на рисунках 5 и 6

Рисунок 5. Экран анализа запроса

Рисунок 6. Экран результатов выборки

Экран 4 показанный на рисунке 5 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 присутствует 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 алгоритмов, указанных на блок-схемах выше.

## Сравнительный анализ эффективности разработанных алгоритмов, примененных к задаче поиска

Для того чтобы сравнить качество поисковой машины необходимы подходящие метрики. 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.

Формально, рассмотрим *N* объектов и *M* элементов . Результат 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 *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:

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

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:

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

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 [9]:

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

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. Убраны/изменены союзы, предлоги, окончания вопроса присутствующего в базе знаний у 2 запросов. Остальные 6 запросов перефразированы с использованием синонима или изменен порядок членов предложения.

Средневзвешенная точность на K элементов для данного алгоритма поиска ответов по вхождению строки представлена на формуле 6 на таблице 1.

|  |  |
| --- | --- |
| Тип поиска | map@K |
| Поиск по вхождению строки | 0,2 |
| Полнотекстовый поиск Postgres | 0,4 |
| Полнотекстовый поиск с NLP оптимизацией | 0,861 |

## Заключение

Результаты разработки подтверждают, что 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. 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.

## Список использованных источников

1. 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.

2. 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.

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

4. Кукушкин А. “Corus - набор корпусов текста русского языка” // Текст: электронный // natasha.github.io [Электронный ресурс]. URL: https://github.com/natasha/corus.

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

6. 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.

7. Портер М. Russian stemming algorithm // Текст: электронный // snowball.tartarus.org [Электронный ресурс]. URL: http://snowball.tartarus.org/algorithms/russian/stemmer.html.

8. Основы полнотекстового поиска в PostgreSQL // Текст: электронный // pgdocs.ru [Электронный ресурс]. URL: https://eax.me/postgresql-full-text-search/.

9. Метрики качества ранжирования // Текст: электронный // habr.com [Электронный ресурс]. URL: https://habr.com/ru/companies/econtenta/articles/303458/.