

**ВИЗУАЛИЗАЦИЯ СВЯЗАННЫХ СУЩНОСТЕЙ НА ОСНОВЕ
СЕМАНТИЧЕСКОГО АНАЛИЗА ТЕКСТОВЫХ ДАННЫХ ПРИ
ПОМОЩИ МОДЕЛИ WORD2VEC НА ЯЗЫКЕ PYTHON**

Выпускная квалификационная (дипломная) работа

Разработчик:
Программист Python
Турков Леонид Алексеевич

Санкт-Петербург

2024

Содержание

ВВЕДЕНИЕ.....	3
ГЛАВА I. Модели семантического анализа на языке Python.....	5
§ 1.1 Роль и задачи семантического анализа в современном мире.....	5
§ 1.2 Анализ свойств и особенностей различных моделей семантического анализа. Обоснование выбора модели семантического анализа.....	13
ГЛАВА II. Реализация библиотеки для работы и визуализации данных пользовательской модели Word2Vec.....	23
§ 2.1 Рассмотрение основных библиотек, разработка структуры библиотеки, существующие инструменты визуализации.....	23
§ 2.1 Реализация модулей библиотеки и интерпретация результатов.....	33
ЗАКЛЮЧЕНИЕ	55
СПИСОК ИСТОЧНИКОВ.....	57
Приложение № 1	59

ВВЕДЕНИЕ

В современной науке в последнее десятилетие активно применяются технологии nlp различного рода. При помощи них удаётся находить нестандартные закономерности в языковых структурах и анализировать сложные социальные явления.

Данная работа посвящена визуализации данных, получаемых при помощи обучения моделей семантического анализа Word2Vec. Это векторная модель, которая работает по принципу частности появления слова в схожих контекстах. Каждому слову в пространстве присваивается свой вектор, направление которого можно сравнить с направлениями векторов других слов, представленных в словаре. Одной из ключевых особенностей моделей данного класса является их лёгкость и нетребовательность к производительности, что позволяет использовать без затруднения многим исследователям и другим заинтересованным лицам.

Тем не менее, ввиду развития основное библиотеки модели genism используемые ранее библиотеки для создания графов знаний, позволяющих выделять значимые сущности при помощи простой визуализации словаря, уже не поддерживаются и не актуальны для текущей версии библиотеки.

Актуальность данной работы обусловлена необходимостью создания простой библиотеки для быстрой работы с семантической моделью Word2Vec, обученной на пользовательских данных и необходимостью простой визуализации данных в направленный граф.

Объектом данной работы является класс семантических моделей Word2Vec.

Предметом данной работы выступает библиотека для простого обучения и визуализации моделей Word2Vec.

Целью данной работы является реализация библиотеки для обучения и визуализации модели в граф.

Для достижения вышеуказанной цели необходимо выполнить следующие задачи:

1. Рассмотреть модели семантического анализа в контексте современности.
2. Найти обоснование применения моделей класса Word2Vec в современных задачах, связанных с семантическим анализом.
3. Уточнить структуру проекта и используемые библиотеки, необходимы для его реализации.
4. Описать и прокомментировать реализацию библиотеки.

При написании данной работы были использованы устоявшиеся практики при написании кода на Python, принципы структурирования и описания программы.

Данная работа состоит из введения, двух глав, одна теоретическая, вторая практическая, каждая из которых содержит два параграфа, заключения, списка источников и приложения.

В данном исследовании авторами были использованы библиотеки genism, nltk, networkx, pymorphy2, pandas, re и др., а также система управления версиями git и github.

ГЛАВА I. Модели семантического анализа на языке Python.

§ 1.1 Роль и задачи семантического анализа в современном мире.

Семантический анализ (или семантическая обработка) относится к области обработки естественного языка (Natural Language Processing, NLP) и является процессом понимания значения и смысла текстовой информации. Целью семантического анализа является выявление смысла за пределами простого синтаксиса и лексики текста.

В контексте NLP и компьютерной лингвистики, семантический анализ может включать в себя следующие задачи:

1. Выделение ключевых сущностей (NER - Named Entity Recognition):
Определение и классификация именованных сущностей в тексте, таких как имена людей, места, организации и т. д.
2. Анализ тональности: Определение эмоциональной окраски текста, например, положительной, отрицательной или нейтральной, что может быть полезно в задачах анализа отзывов, мониторинга социальных медиа и т. д.
3. Извлечение информации (IE - Information Extraction): Извлечение структурированной информации из текста, такой как отношения между сущностями, факты и события.
4. Семантический поиск: Разработка методов для улучшения релевантности результатов поиска, учитывая семантическое сходство запроса и документов.
5. Семантический анализ текстовых данных: Преобразование текстовых данных в числовые представления, например, с использованием векторных представлений слов, чтобы компьютер мог понимать семантические отношения между словами.
6. Машинный перевод: Перевод текста с одного языка на другой с учетом семантики и контекста.

7. Семантическое сравнение: Оценка семантического сходства или различия между текстовыми элементами, такими как слова, фразы или документы.

Для решения этих задач используются различные методы и технологии, включая машинное обучение, глубокое обучение, статистические методы, а также использование предварительно обученных моделей для семантического анализа текста.

История развития семантического анализа тесно связана с развитием области искусственного интеллекта (ИИ) и обработки естественного языка (Natural Language Processing, NLP). Взглянем на основные этапы этого развития.

Первые шаги в направлении семантического анализа были сделаны в рамках исследований по искусственному интеллекту в 1950-60 годах. Однако, в эти ранние годы средства и вычислительные ресурсы были ограничены, и исследователи сосредотачивались в первую очередь на синтаксическом анализе и понимании структуры предложений.

Одним из значимых событий в развитии семантического анализа было создание WordNet в конце 1980-х годов. WordNet — это лексическая база данных, представляющая собой семантическую сеть, где слова группируются по семантическим отношениям. Она стала важным ресурсом для изучения семантических связей слов и их значений.

В конце 20-го века начали активно использоваться векторные модели для представления семантической близости слов. Модели, такие как Latent Semantic Analysis (LSA), позволяли представлять слова в векторном пространстве, учитывая их семантическое сходство на основе статистики встречаемости в текстах.

В последнем десятилетии произошел значительный прорыв с появлением методов Word Embeddings, в частности, Word2Vec, разработанного компанией Google. Word2Vec позволяет представлять слова в непрерывных векторах так, чтобы близкие по смыслу слова имели близкие векторы.

С развитием глубокого обучения, в частности моделей на основе трансформеров, таких как BERT (Bidirectional Encoder Representations from Transformers), семантический анализ стал более контекстуальным и мощным. BERT способен учитывать контекст слов в предложении, что значительно улучшает его способности в понимании семантических отношений.

В настоящее время технологии семантического анализа активно применяются в различных областях, таких как поиск в интернете, анализ социальных медиа, машинный перевод, чат-боты, медицинская обработка текстов и многое другое. Предварительно обученные модели, такие как GPT (Generative Pre-trained Transformer), демонстрируют уровень семантического понимания, который приводит к новым возможностям в обработке естественного языка.

В целом, история семантического анализа отражает постоянный прогресс в направлении более точного и глубокого понимания смысла текстовой информации, от ранних исследований до современных технологий глубокого обучения.

Семантический анализ играет ключевую роль в современном мире, особенно в контексте обработки больших объемов текстовой информации и развития технологий искусственного интеллекта. Вот несколько основных сфер, где семантический анализ имеет важное значение:

1. Поиск и ранжирование информации: семантический анализ помогает улучшить качество поисковых систем и ранжирование результатов, учитывая не только ключевые слова, но и семантическое сходство запросов и документов. Это особенно важно при работе с огромными объемами данных в интернете.
2. Анализ социальных медиа и отзывов: в мире социальных медиа и интернет-отзывов семантический анализ позволяет понимать тональность высказываний, выявлять тренды, анализировать отзывы о продуктах и услугах, а также следить за общественным мнением.

3. Контекстуальный поиск и разработка виртуальных ассистентов: семантический анализ необходим для построения контекстуальных поисковых систем и разработки виртуальных ассистентов, которые способны понимать и интерпретировать запросы пользователей в соответствии с их контекстом.
4. Машинный перевод: семантический анализ играет важную роль в совершенствовании систем машинного перевода, улучшая точность и качество переводов, так как семантическое понимание контекста является ключевым элементом в успешном переводе.
5. Извлечение информации из текстов: семантический анализ используется для извлечения структурированной информации из текстовых источников, что полезно в области обработки данных, анализа новостей, исследований и других сферах.
6. Развитие чат-ботов и интерфейсов: при создании чат-ботов и других интерфейсов общения с компьютером важно, чтобы они могли понимать не только синтаксис, но и семантику запросов пользователей.
7. Бизнес-аналитика и принятие решений: семантический анализ может быть использован в бизнес-аналитике для обработки и понимания текстовых данных, таких как отчеты, новости, комментарии клиентов, что помогает предприятиям принимать информированные решения.
8. Медицинская обработка текстов: в медицинской области семантический анализ может помочь в извлечении информации из медицинских текстов, а также в обработке больших объемов литературы для обновления баз данных исследований.

Семантический анализ в современном мире является неотъемлемой частью развития интеллектуальных систем и технологий, которые сталкиваются с огромными объемами текстовой информации, требующей понимания и адекватной обработки.

Семантический анализ активно используется в различных компаниях и отраслях для разнообразных целей. Некоторые компании, которые внедрили семантический анализ в свою деятельность, включают в себя:

1. Google. Применяет семантический анализ для улучшения качества поисковых запросов и результатов, а также для оптимизации работы своих интеллектуальных алгоритмов.
2. Facebook. Использует семантический анализ для анализа и классификации содержания, а также для предоставления рекомендаций пользователям.
3. Amazon. Применяет семантический анализ для улучшения рекомендательных систем и анализа отзывов клиентов.
4. Microsoft. Внедряет семантический анализ в различные продукты, включая поисковые системы и технологии искусственного интеллекта.
5. IBM. Занимается исследованиями в области семантического анализа для улучшения систем анализа данных и обработки естественного языка.
6. Uber. Применяет семантический анализ для анализа данных и обратной связи от пользователей, а также для оптимизации работы своих алгоритмов маршрутизации.
7. Apple. Внедряет семантический анализ в свои голосовые ассистенты (например, Siri) для более точного понимания запросов пользователей.
8. Tesla. Использует семантический анализ для обработки данных и обратной связи от автомобилей, что помогает в разработке и улучшении автопилота и других функций.

Эти компании применяют семантический анализ для оптимизации процессов, улучшения пользовательских интерфейсов, анализа данных и принятия более информированных решений в различных областях своей деятельности.

Перспективы развития семантического анализа предоставляют захватывающие возможности для улучшения способности компьютеров понимать и обрабатывать текстовую информацию. В первую очередь,

интеграция семантического анализа с передовыми методами глубокого обучения предвещает создание более сложных и контекстно-чувствительных моделей. Это позволит эффективнее учитывать семантические зависимости в тексте, повышая точность в задачах, таких как извлечение ключевой информации и классификация текстов по тональности. Развитие гибридных подходов, комбинирующих силы семантического анализа и глубокого обучения, предоставит более мощные инструменты для решения сложных задач в обработке естественного языка.

Кроме того, с ростом интереса к мультимодальным данным, семантический анализ будет стремиться к интеграции с обработкой аудио, видео и изображений. Это открывает новые перспективы для создания современных систем, способных анализировать и понимать контент в различных форматах. Такой подход укрепит способности систем в распознавании семантических связей в тексте, звуке и визуальных данных, повышая их универсальность и применимость в различных областях, включая развлечения, медицину, образование и многие другие. Перспективы семантического анализа пролегают путь к более интеллектуальным системам обработки информации, улучшающим взаимодействие между человеком и компьютером в цифровой эпохе.

Визуализация семантического анализа представляет собой неотъемлемый инструмент для более глубокого понимания содержания текстов и извлечения смысловой информации. Прежде всего, она обеспечивает интуитивное восприятие семантических отношений между словами и понятиями, что делает процесс анализа более доступным и понятным. Визуальные представления, такие как облако слов, граф семантических связей или тематические карты, позволяют исследователям и аналитикам оперативно выявлять ключевые темы, выделять наиболее значимые слова и отслеживать динамику смыслового содержания текста.

Одним из важных аспектов визуализации семантического анализа является способность выявления семантических связей и зависимостей между словами. Это позволяет отслеживать тонкости смысла в тексте, выявлять контекстуальные

взаимосвязи и исследовать семантическую близость между терминами. Такие визуализации не только помогают в простом восприятии информации, но и акцентируют внимание на важных элементах текста, что может быть критически важно для обобщения, анализа тональности, идентификации ключевых сущностей и выявления общих тематик.

Кроме того, визуализация семантического анализа облегчает коммуникацию результатов исследования. Она позволяет исследователям и специалистам в области обработки естественного языка обмениваться информацией с коллегами и заинтересованными сторонами, даже если они не обладают глубокими знаниями в данной области. Таким образом, визуализация семантического анализа становится ключевым инструментом для исследования текстов и принятия обоснованных решений в различных областях, включая медицину, бизнес-аналитику, образование и другие.

Таким образом, в данном параграфе работы нами были рассмотрены вопросы, связанные с семантическим анализом текстов и его приложениями. Была описана краткая история семантического анализа.

Была предоставлена информация о важности семантического анализа в современном мире, подчеркивая его роль в различных областях, таких как информационные технологии, медицина, бизнес и образование. Обсуждены применения, начиная от улучшенного поиска в интернете до создания более эффективных систем образования и интеллектуальных ассистентов.

Также были рассмотрены перспективы развития семантического анализа, включая интеграцию с глубоким обучением, обработку мультимодальных данных и использование в передовых технологиях, таких как трансформеры.

Рассмотрение визуализации семантического анализа выявило ее важную роль в облегчении восприятия результатов, выделении ключевых тем и слов, а также в поддержке коммуникации между исследователями и другими заинтересованными сторонами.

Итоговый вывод подчеркивает значимость семантического анализа в современной науке и технологиях, выделяя его как важный инструмент для понимания и обработки текстовой информации в различных контекстах.

В следующем параграфе мы рассмотрим различные модели для семантического анализа текстов и обоснуем выбор одной из них в качестве использования для визуализации данных модели.

§ 1.2 Анализ свойств и особенностей различных моделей семантического анализа. Обоснование выбора модели семантического анализа.

Существует множество моделей для семантического анализа текстов, и выбор конкретной зависит от задачи, требований и доступных ресурсов. Ниже приведен обзор некоторых из наиболее широко используемых моделей:

1. Word Embeddings

- Word2Vec, разработанный Google создает векторные представления слов, учитывая их семантические отношения.
- GloVe (Global Vectors for Word Representation) использует статистику совстречаемости слов в тексте для создания векторных представлений.

2. FastText

- Разработанный Facebook, FastText расширяет идеи Word2Vec, добавляя поддержку для подслов (subword embeddings), что полезно для работы с морфологией и неизвестными словами.

3. BERT (Bidirectional Encoder Representations from Transformers):

- Разработанный Google, BERT — это трансформерная модель, способная понимать контекст и взаимосвязи слов в предложении, что делает ее особенно эффективной для различных задач, таких как вопросно-ответная система, извлечение информации и многое другое.

4. ELMo (Embeddings from Language Models):

- ELMo создает векторные представления слов, учитывая их семантический контекст в предложении. Эта модель также основана на трансформерах.

5. USE (Universal Sentence Encoder):

- Разработанный Google, USE создает векторные представления для предложений, что полезно для задач классификации текста и определения семантической близости.

6. XLNet:

- Также основанный на трансформерах, XLNet учитывает билингвальные зависимости слов и обладает высокой производительностью в различных задачах обработки естественного языка.

7. Doc2Vec (Paragraph Vectors):

- Подобно Word2Vec, но работает на уровне абзацев или документов, создавая векторные представления для целых текстовых блоков.

8. RoBERTa (Robustly optimized BERT approach):

- RoBERTa - это улучшенная версия BERT, предназначенная для более эффективного обучения и более точного представления смысла в тексте.

Это лишь несколько примеров. Каждая из этих моделей имеет свои преимущества и ограничения, и выбор зависит от конкретных требований проекта. Также важно отметить, что в сфере обработки естественного языка появляются новые модели и усовершенствования с течением времени.

В контексте нашего исследования отдельный интерес представляет использования моделей семантического анализа с целью сравнения смысловой близости двух слов. Ключевая идея состоит в формировании таким образом набора сущностей, которые потенциально могут иметь неочевидную связь между собой.

Далее нам необходимо определиться какие из существующих моделей являются наиболее предпочтительным для детального рассмотрения в нашем исследовании.

Для сравнения семантической близости слов важно выбирать модели, которые эффективно захватывают семантические отношения между словами. Некоторые из наиболее подходящих моделей для этой задачи представлены ниже.

- Word2Vec: Word2Vec прекрасно справляется с выделением семантической близости между словами, создавая векторные представления, которые учитывают семантические отношения. В

основе этой модели лежит идея, что слова, используемые в похожих контекстах, имеют схожий смысл, что полезно для сравнения семантической близости.

- FastText: Модель FastText, расширяя идеи Word2Vec, добавляет поддержку для подслов, что может быть особенно полезно в случаях, когда слова могут быть неизвестными или иметь похожую морфологию.
- BERT (и его вариации): BERT и его улучшенные версии, такие как RoBERTa, обеспечивают выдающуюся производительность в задачах, связанных с семантической близостью. Благодаря способности учесть контекст и взаимосвязь слов в предложении, они превосходно подходят для задач, требующих высокой точности в определении семантической близости.
- USE (Universal Sentence Encoder): Эта модель, разработанная Google, специально предназначена для создания векторных представлений для предложений и подходит для сравнения семантической близости не только слов, но и целых предложений.

Теперь нам необходимо детально рассмотреть преимущества и недостатки этих моделей.

Модель Word2Vec представляет собой мощный инструмент для измерения семантической близости слов, обладая несколькими значительными преимуществами. Во-первых, Word2Vec способна захватывать семантические отношения между словами в контексте их употребления. Это достигается благодаря использованию нейронной сети, которая обучается предсказывать близкие по смыслу слова по их контексту. Таким образом, полученные векторные представления слов отражают их семантическую схожесть на основе совместного контекста.

Во-вторых, Word2Vec создает плотные векторные представления, что означает, что векторы слов имеют непрерывные значения. Это важно при

сравнении семантической близости, поскольку плотные векторы обеспечивают более точное представление семантических отношений и позволяют эффективное измерение схожести между словами.

Наконец, модель Word2Vec обладает относительной простотой и легкостью в использовании. Она не требует большого объема данных для обучения и обеспечивает быстрые вычисления при измерении семантической близости, что является преимуществом в приложениях, где важна эффективность.

Среди недостатков модели Word2Vec можно выделить ограничения в учете семантических контекстов. Так как модель рассматривает слова внутри локальных окон, она может упускать долгосрочные зависимости и семантические взаимосвязи, особенно в длинных текстах или в случае сложных структур предложений.

Еще одним недостатком является ограниченная способность обрабатывать синонимы и слова, имеющие несколько значений (полисемичные слова). Word2Vec может создавать одинаковые векторы для синонимов, что ограничивает его точность в разрешении тонких семантических различий.

Кроме того, Word2Vec не учитывает порядок слов в предложении, что делает его менее эффективным в задачах, где важна синтаксическая структура. В таких случаях модели, учитывающие порядок слов, например, BERT, могут предоставлять более точные результаты.

Таким образом, несмотря на свои преимущества, модель Word2Vec обладает определенными ограничениями, которые следует учитывать при использовании в задачах сравнения семантической близости слов.

Модель BERT (Bidirectional Encoder Representations from Transformers) представляет собой значительный прорыв в области семантического анализа текстов и сравнения семантической близости слов. Одним из ключевых преимуществ BERT является его способность учсть билингвальные зависимости слов, рассматривая текст в обоих направлениях. Это обеспечивает более глубокое понимание контекста и контингентные представления слов, учитывающие их окружение в предложении.

Еще одним преимуществом BERT является его способность захватывать долгосрочные зависимости и сложные взаимосвязи между словами. Поскольку BERT обучается на больших корпусах текста, он способен улавливать контекстуальные нюансы и тонкости в семантических отношениях, что особенно важно при сравнении близости слов в сложных текстах или в предложениях с множеством вложенных значений.

Дополнительным преимуществом BERT является его способность работать с полисемичными словами и улавливать различия в значениях слов в зависимости от их контекста употребления. Это обеспечивает более точные представления для слов с несколькими значениями, что улучшает качество сравнения семантической близости.

В качестве слабостей модели BERT стоит отметить ее вычислительную сложность и требовательность к ресурсам. Обучение BERT требует большого объема данных и мощности вычислений, что может быть проблемой в ресурсоограниченных сценариях. Также размеры самой модели могут создавать проблемы в интеграции в некоторые приложения с ограниченными ресурсами.

Еще одним ограничением BERT является его относительная медлительность в сравнении с более простыми моделями, такими как Word2Vec, особенно при выполнении операций сравнения на больших объемах текста.

Кроме того, BERT не всегда эффективно справляется с задачами, требующими учета порядка слов в предложении. В некоторых сценариях, где важна синтаксическая структура, более подходящими могут быть модели, специально разработанные для учета этого аспекта.

Соответственно, несмотря на значительные преимущества, модель BERT также имеет свои ограничения, которые следует учитывать при выборе для задач сравнения семантической близости слов.

Модель FastText представляет собой эволюцию и расширение идей, заложенных в модели Word2Vec, и обладает рядом выдающихся преимуществ при сравнении семантической близости слов. Во-первых, ключевым преимуществом является способность FastText учитывать подслова (subword

embeddings). Это особенно полезно в случаях, когда слова могут содержать префиксы, суффиксы или другие морфологические изменения. Благодаря этой особенности, FastText может лучше обрабатывать сложные языковые структуры и работать с неизвестными или редкими словами.

Во-вторых, FastText способна улавливать семантические отношения на уровне морфем, что делает ее более универсальной в различных языках и лингвистических контекстах. Это преимущество особенно ценно в многоязычных сценариях или при работе с текстами, содержащими разнообразные лексические особенности.

Другим важным преимуществом является относительная легкость и эффективность обучения FastText. Модель может быстро адаптироваться к новым данным и не требует огромного объема обучающих данных для достижения хороших результатов. Это делает FastText более доступной и привлекательной для использования в сценариях с ограниченными ресурсами.

Одним из недостатков FastText является ограниченная способность учитывать контекст и порядок слов в предложении. В отличие от некоторых более сложных моделей, таких как BERT, которые учитывают билингвальные зависимости, FastText ограничивается локальными контекстами и может терять часть семантической информации.

Еще одним ограничением является более грубое представление семантических отношений в сравнении с более сложными моделями. FastText создает векторы слов, учитывая их морфемную структуру, но не всегда точно отражает тонкости смысла в тексте. Это может быть проблемой при работе с текстами, где важны нюансы в семантических отношениях.

Также следует учесть, что FastText может иметь проблемы при обработке длинных текстов и в задачах, требующих учета глубоких синтаксических структур. В таких сценариях более сложные модели, способные учесть долгосрочные зависимости, могут предоставить более точные результаты.

Можно заключить, что FastText, как и другие модели, не лишен недостатков и требует тщательного подбора задач для решения.

Модель Universal Sentence Encoder (USE) представляет собой инновационный подход к извлечению векторных представлений для предложений и текстовых блоков, что обеспечивает несколько выдающихся преимуществ при сравнении семантической близости слов. Во-первых, USE способен генерировать компактные и информативные векторы для целых предложений, что позволяет сравнивать семантическую близость не только отдельных слов, но и структур более высокого уровня.

Еще одной сильной стороной USE является его способность улавливать семантическую схожесть в различных контекстах. Модель обучается на широком спектре данных, что позволяет ей создавать векторные представления, учитывающие разнообразные стили и тематики. Это делает USE универсальным инструментом для сравнения семантической близости в различных областях, включая терминологию и выражения с разнообразными значениями.

Кроме того, USE обучается учитывать синтаксические и семантические особенности текста, что позволяет ему точно захватывать смысловые отношения между словами и выражениями. Это особенно важно при сравнении слов в разных контекстах и при обработке сложных языковых конструкций.

Ограничением USE является его тенденция к сглаживанию тонких семантических различий между словами. В связи с тем, что модель обучается усреднять информацию в предложениях, она может упускать некоторые детали и нюансы, что может быть важно при анализе слов с близкими, но неидентичными значениями.

Еще одним ограничением является относительная ограниченность в обучении на многоязычных данных. USE может иметь ограниченную способность эффективно работать с языками, для которых у него меньше обучающих данных. Это может сказаться на качестве сравнения семантической близости в таких языках.

Кроме того, USE может также оказаться менее гибким в сравнении с некоторыми более сложными моделями, такими как BERT, в контексте разнообразных задач обработки естественного языка. Он сконцентрирован в

первую очередь на семантическом понимании предложений и может не так эффективно решать задачи, требующие дополнительных синтаксических знаний.

После того, как мы рассмотрели наиболее предпочтительные модели и некоторые их особенности необходимо обосновать выбор одной из ранее перечисленных моделей. Согласно нашему представлению, Word2Vec является наилучшим вариантом для использования в данном исследовании. Ниже будут изложены доводы в пользу такого решения.

Модель Word2Vec изначально привлекает внимание своей простотой и легкостью в реализации. Это делает ее высокоэффективной для быстрого внедрения в различные приложения и использования в различного рода исследованиях, особенно в сравнении с более сложными моделями, требующими значительных вычислительных ресурсов и времени для обучения.

Word2Vec успешно моделирует семантические отношения между словами, учитывая их векторные представления на основе контекста. Эта способность приводит к тому, что слова с близким смыслом имеют близкие векторные представления, что делает Word2Vec надежным инструментом для анализа семантической близости.

Word2Vec создает плотные векторные представления слов, что обеспечивает их обобщение на новые данные. Эта характеристика особенно полезна в случаях, когда у нас есть ограниченный объем обучающих данных, и требуется обобщение на разнообразные текстовые контексты.

Также модель позволяет проводить над векторами операции сложения и вычитания, что расширяет возможный инструментарий для пользователя.

Word2Vec имеет широко распространенные предобученные модели, обученные на крупных корпусах текста. Это позволяет использовать эти модели без необходимости обучения с нуля, что упрощает задачу анализа семантической близости слов для конкретных задач и областей.

Векторные представления, созданные с использованием Word2Vec, могут быть успешно применены в различных задачах обработки естественного языка,

включая классификацию текста, кластеризацию, и извлечение информации, что подчеркивает универсальность этой модели.

В целом, эти аргументы подчеркивают преимущества Word2Vec, делая его привлекательным выбором для анализа семантической близости слов в ряде практических приложений.

В ходе нашего обсуждения моделей для анализа семантической близости слов были рассмотрены несколько ключевых аспектов и существующих подходов в данной области. Мы рассмотрели несколько выдающихся моделей, включая Word2Vec, FastText, BERT и Universal Sentence Encoder (USE), выявив их преимущества и ограничения.

Таким образом, Word2Vec, будучи одной из первых успешных моделей в этой области, продолжает оставаться востребованной благодаря своей простоте, эффективности и способности захватывать семантические отношения между словами. Она легко интегрируется в различные приложения и может быть успешно использована в условиях ограниченных ресурсов.

FastText, расширяя идеи Word2Vec, привносит улучшения, учитывая морфемную структуру слов и способствуя более точной обработке различных языковых форм.

BERT и USE представляют собой более современные и сложные модели, способные учесть контекст, билингвальные зависимости и работать с высокоуровневыми текстовыми структурами. Однако, их вычислительная сложность и требовательность к ресурсам может быть недоступной в ресурсоограниченных сценариях.

Каждая из этих моделей имеет свои преимущества и ограничения, и выбор конкретной модели зависит от требований задачи, объема данных, доступных ресурсов, и контекста применения. Данный обзор позволяет лучше понимать сильные стороны каждой модели и обеспечивает основу для выбора наиболее подходящей в конкретной ситуации.

Наш выбор в пользу использования Word2Vec обоснован лёгкостью модели, возможностью использования в условиях ограниченных вычислительных

мощностей, наличием готовых предобученных моделей, которые могут быть легко созданы пользователями на основе собственных обучающих корпусов, а также возможностями проведения различных операций над векторными представлениями слов.

В следующей главе мы рассмотрим функционал и структуру предлагаемой библиотеки, подробно разберём реализуемый код, а также продемонстрируем результаты работы библиотеки.

ГЛАВА II. Реализация библиотеки для работы и визуализации данных пользовательской модели Word2Vec.

§ 2.1 Рассмотрение основных библиотек, разработка структуры библиотеки, существующие инструменты визуализации.

Среди библиотек, необходимых для реализации проекта мы выделяем следующие:

1. gensim.

Это библиотека, разработанная для работы с тематическим моделированием и векторным представлением текстовых данных. Она предоставляет эффективные инструменты для анализа семантической структуры текста, извлечения тем и создания векторных представлений слов. Разработанная Радимом Рефаэлем и выпущенная в 2009 году, Gensim является одной из ключевых библиотек в области обработки естественного языка.

Основные характеристики Gensim включают в себя:

- Модульность и Простоту Использования. Gensim предоставляет простой и интуитивно понятный интерфейс, что делает его доступным для широкого круга пользователей. Модульная структура библиотеки позволяет легко комбинировать различные компоненты для решения конкретных задач.
- Тематическое Моделирование. Библиотека специализируется на тематическом моделировании, таком как Latent Semantic Analysis (LSA) и Latent Dirichlet Allocation (LDA). Эти методы позволяют выявлять темы в коллекциях текстов и определять, какие слова связаны с каждой темой.
- Векторные Представления Слов. Gensim предоставляет средства для обучения векторных представлений слов, таких как Word2Vec. Эти векторы могут быть использованы для измерения семантической близости слов, кластеризации и других задач анализа текстов.

- Эффективность и Масштабируемость. Gensim оптимизирован для работы с большими объемами данных. Многие алгоритмы реализованы с учетом эффективности, что позволяет проводить анализ текстов на масштабах, соответствующих современным требованиям.
- Поддержка Специализированных Форматов Данных. Библиотека может обрабатывать различные форматы текстовых данных, включая корпусы в форматах, таких как Bag of Words (BoW) или Term Frequency-Inverse Document Frequency (TF-IDF).
- Расширяемость и Сообщество. Gensim является открытым проектом, и его API легко расширяется. Сообщество пользователей активно обсуждает и дополняет функционал библиотеки, что способствует ее развитию и обеспечивает доступ к различным расширениям.

2. nltk.

Natural Language Toolkit (NLTK) — это библиотека программирования на языке Python, предназначенная для работы с обработкой естественного языка (Natural Language Processing, NLP). Она предоставляет разнообразные инструменты и ресурсы, необходимые для анализа текста и работы с лингвистическими данными. Разработанная командой исследователей и инженеров, возглавляемой Стивеном Бёрдом и Эдем Лопесом, NLTK была выпущена в 2001 году и с тех пор стала одной из наиболее широко используемых библиотек в области NLP.

Основные характеристики NLTK включают в себя следующее:

- NLTK предоставляет доступ к различным текстовым корпусам и лексическим ресурсам для обучения и оценки алгоритмов NLP. Эти ресурсы включают в себя множество языков и доменов, что делает их полезными для разнообразных задач.
- Библиотека поддерживает работу с текстами на различных языках, предоставляя инструменты для токенизации, лемматизации,

стемминга, частеречной разметки и других основных задач обработки текста.

- NLTK включает в себя множество алгоритмов и методов для решения задач NLP, таких как синтаксический анализ, машинное обучение для классификации и кластеризации текстов, генерация текста и другие.
- Библиотека обеспечивает возможность использования многих алгоритмов машинного обучения для обработки текстовых данных, включая байесовские классификаторы, деревья принятия решений и многое другое.

NLTK активно используется в академических и индустриальных исследованиях, а также в образовательных целях. Ее модульность и гибкость позволяют адаптировать ее к различным задачам обработки текста, что делает ее одним из ключевых инструментов в области обработки естественного языка.

3. Rymorphy2

библиотека для морфологического анализа русского текста на языке программирования Python. Разработанная Иваном Бегтином, rymorphy2 предоставляет возможность извлечения грамматической информации из слов и их нормализации, что обеспечивает более глубокий анализ структуры русского языка. Эта библиотека основана на словаре и правилах, что делает ее мощным инструментом для лемматизации и определения форм слов.

Основные характеристики rymorphy2 включают:

- Морфологический Анализ: Rymorphy2 обеспечивает возможность проведения морфологического анализа русских слов, включая извлечение грамматических характеристик, таких как часть речи, род, число, падеж и т.д.
- Лемматизация: Библиотека позволяет лемматизировать слова, возвращая их начальные формы. Это полезно для сокращения слов до

их базовых форм и сопоставления словоформ разных грамматических форм.

- Нормализация: PyMorph2 позволяет нормализовать слова, приводя их к единой форме. Это полезно для стандартизации текста и улучшения точности анализа.
- Обработка Неизвестных Слов: Библиотека способна обрабатывать неизвестные слова, используя статистические методы и контекстную информацию.
- Поддержка Грамматических Тегов: PyMorph2 поддерживает работу с грамматическими тегами, что обеспечивает детальное понимание грамматической структуры слов.
- Простота Использования: Библиотека имеет простой интерфейс, что делает ее доступной для широкого круга пользователей. Она может быть легко интегрирована в различные приложения и сценарии обработки текста на русском языке.

PyMorph2 активно используется в приложениях обработки естественного языка, включая поисковые системы, аналитические инструменты и различные проекты, где требуется глубокий морфологический анализ русского текста. Его простота и эффективность делают его важным инструментом для работы с русскоязычными текстами в контексте NLP.

4. network

NetworkX - это библиотека программирования на языке Python, предназначенная для работы с теорией графов. Разработанная Джаредом Лоджом, NetworkX предоставляет инструменты для создания, анализа и визуализации различных типов графов и сетей. Эта библиотека играет ключевую роль в исследованиях и приложениях, связанных с комплексными сетевыми структурами.

Основные характеристики NetworkX включают:

- Представление Графов. NetworkX предоставляет гибкую структуру для представления различных типов графов, включая ориентированные и неориентированные графы, взвешенные и невзвешенные, мультиграфы и др.
- Алгоритмы Графов. Библиотека включает богатый набор алгоритмов для работы с графиками, включая алгоритмы поиска кратчайших путей, центральности, кластеризации, потоков в сети и другие.
- Создание и Манипулирование Графами. NetworkX предоставляет простые и интуитивно понятные средства для создания, редактирования и манипулирования графиками. Это включает в себя добавление и удаление узлов и рёбер, изменение атрибутов графов и т.д.
- Визуализация. Библиотека обеспечивает возможность визуализации графов с помощью различных методов, включая отображение узлов и рёбер, цветовую кодировку атрибутов, использование различных макетов и т.д.
- Генерация Графов. NetworkX включает в себя функции для генерации различных классов графов, таких как полные графы, случайные графы, графы на основе моделей случайных сетей и другие.
- Импорт и Экспорт Данных Графов. Библиотека обеспечивает возможность импорта и экспорта графов из и в различные форматы данных, что облегчает работу с графовыми данными из внешних источников.

NetworkX широко используется в академических исследованиях и индустриальных приложениях для моделирования и анализа сложных сетевых структур, таких как социальные сети, транспортные сети, биологические сети и многие другие. Его гибкость, расширяемость и обширные возможности делают его важным инструментом в области теории графов и сетевого анализа.

Ниже представлено краткое описание сопровождающих модулей, необходимых для работы библиотеки.

5. Pandas:

Описание: Pandas представляет собой библиотеку для обработки и анализа данных в языке программирования Python. Она предоставляет высокоуровневые структуры данных, такие как DataFrame, для удобного манипулирования и анализа табличных данных.

Применение: Pandas активно используется для чистки, преобразования и анализа данных в областях работы с данными, машинного обучения и статистики.

6. Pathlib:

Описание: Pathlib является модулем в стандартной библиотеке Python, предоставляющим объектно-ориентированный интерфейс для работы с путями файловой системы. Он облегчает работу с путями к файлам и директориям.

Применение: Pathlib используется для создания, чтения и манипуляции путями файлов, что делает его удобным инструментом для операций ввода-вывода.

7. BeautifulSoup (bs4):

Описание: BeautifulSoup - это библиотека для извлечения данных из HTML и XML файлов. Она предоставляет простые методы для навигации и поиска информации в структурах веб-страниц.

Применение: BeautifulSoup широко используется для веб-скрапинга и обработки HTML-кода для извлечения нужной информации.

8. Regular Expressions (re):

Описание: Модуль re в Python предоставляет функционал для работы с регулярными выражениями. Регулярные выражения используются для поиска и манипуляции текстовых данных с использованием паттернов.

Применение: re используется для поиска, извлечения и замены текста на основе заданных шаблонов, что полезно для обработки строк.

9. Matplotlib:

Описание: Matplotlib - это библиотека для создания графиков и визуализации данных в Python. Она предоставляет множество функций для создания различных типов графиков и диаграмм.

Применение: Matplotlib используется для визуализации данных, создания графиков и диаграмм в научных и инженерных приложениях.

10.CSV (csv):

Описание: Модуль csv в Python обеспечивает функционал для работы с файлами в формате CSV (Comma-Separated Values). Он позволяет читать и записывать данные в формате таблицы, разделенной запятыми.

Применение: csv используется для обработки данных, хранящихся в формате CSV, что часто встречается в работе с табличными данными.

Структура проекта представлена следующим образом.

1. Project – корневая папка.

- a. Src – основная папка с модулями проекта.

- i. Paths.py – пути к директориям, находятся независимо от операционной системы.

- ii. Preprocessor.py – модуль для очистки и предобработки данных для обучения.

- iii. Train_model.py – модуль для создания и тренировки пользовательских моделей.

- iv. Visualization.py – модуль для создания графа знаний на основе словаря модели и входного слова или списка слов пользователей.

- v. Use_example.py – модуль с примерами использования кода библиотеки.

- b. Models – директория для хранения моделей.

- i. Custom – папка для хранения пользовательских моделей.

- c. Data – папка для хранения данных

- i. Train – папка для хранения тренировочных данных.

1. Raw – в этой директории хранятся сырые необработанные данные.
- ii. Output – папка для хранения выходных данных.
- d. Venv – виртуальная среда.
- e. .gitignore – файл с указаниями для игнорирования отслеживания git изменений в конкретных файлах.
- f. Requirements.txt – список всех зависимостей.

На изображении 1 представлена структура папок в текстовом редакторе VSCode.

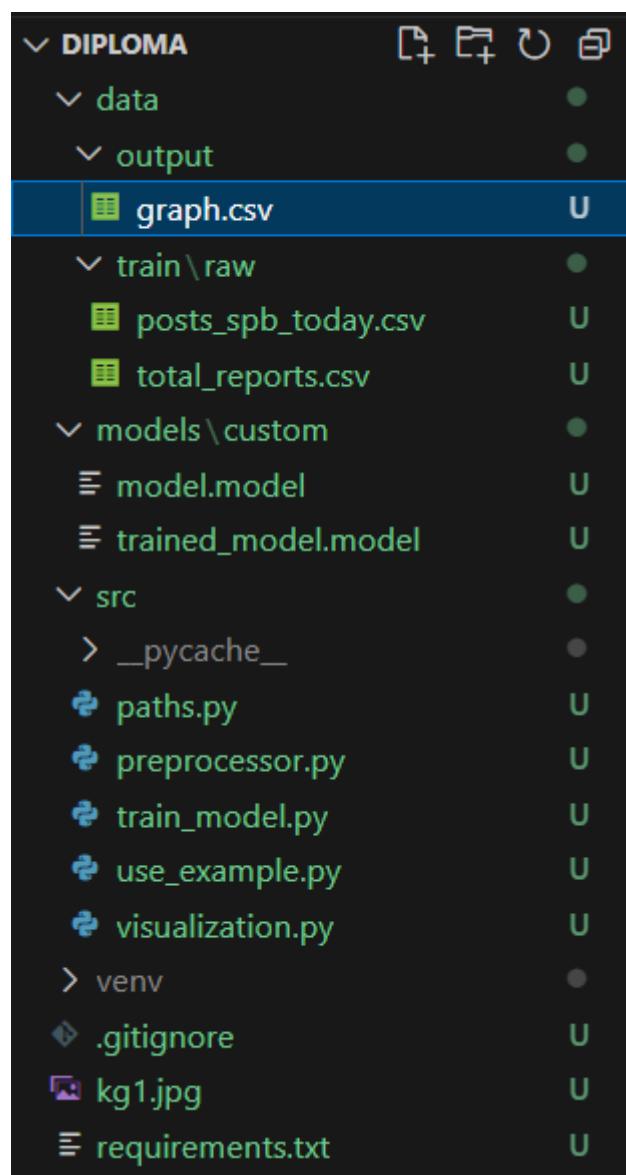


Рисунок 1.

Список всех зависимостей:

```
beautifulsoup4==4.12.3
bs4==0.0.2
certifi==2023.11.17
charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
contourpy==1.2.0
cycler==0.12.1
DAWG-Python==0.7.2
docopt==0.6.2
fonttools==4.47.2
gensim==4.3.2
idna==3.6
importlib-resources==6.1.1
joblib==1.3.2
kiwisolver==1.4.5
lxml==5.1.0
matplotlib==3.8.2
networkx==3.2.1
nltk==3.8.1
numpy==1.26.3
packaging==23.2
pandas==2.1.4
pillow==10.2.0
pymorphy2==0.9.1
pymorphy2-dicts-ru==2.4.417127.4579844
pyparsing==3.1.1
python-dateutil==2.8.2
pytz==2023.3.post1
regex==2023.12.25
requests==2.31.0
scipy==1.11.4
six==1.16.0
smart-open==6.4.0
soupsieve==2.5
tqdm==4.66.1
tzdata==2023.4
urllib3==2.1.0
zipp==3.17.0
```

Изначально, мы предполагали использовать библиотеку vec2graph для визуализации графа.

Это библиотека была специально создана для визуализации графа из моделей Word2Vec, но, к сожалению, давно не обновлялась своими создателями. К тому же, она была предназначена для работы с моделями типа Kyedvectors, отличительной чертой которых является отсутствие возможности дообучения.

По сути, они похожи чем-то на словарь в плане использования. Модели же типа Word2Vec не обладают атрибутом `vocab`, что на корню ломает возможность использовать данный инструмент.

Поэтому нами было принято решения написать визуализатор самостоятельно, используя широкие возможности библиотеки `networkx`, которая, как нами было отмечено ранее обладает большими возможностями при создании графов знаний.

Таким образом в данном параграфе мы рассмотрели основные использованные нами модули при создании библиотеки. К ним относятся `genism`, `nltk`, `networkx`, `pytmorphy2`, а также вспомогательные библиотеки `pandas`, `lxml`, `csv`, `pathlib`, `bs4`, `re` и `Matplotlib`. Соответственно, ввиду сложности и многоаспектности данных библиотек, автоматически были подтянуты другие модули в качестве зависимостей.

Мы также рассмотрели структуру проекта, выделили основным папки и директории и обосновали сложности связанные с использованием библиотеки `vec2graph`.

В следующем параграфе мы рассмотрим непосредственно код и исполнение модулей проекта.

§ 2.1 Реализация модулей библиотеки и интерпретация результатов.

Начать рассмотрение модулей на наш взгляд следует с модуля paths.

Импорт библиотек представлен ниже.

```
from pathlib import Path
```

Задачей данного модуля является сохранение констант путей в переменные для дальнейшего упрощённого доступа к ним.

В первую очередь нам необходимо получить доступ к полному пути текущего файла, чтобы ориентироваться в пространстве папок.

```
# Получаем строку, содержащую путь к рабочей директории:
```

```
dir_path = str(Path.cwd())
```

Далее нам необходимо разбить путь на составляющие, чтобы потом его объединить. Центральной идеей является возможность использования библиотеки с разных операционных систем, а в них, как известно, символы разделителей путей используются разные.

```
def get_dir_path(system_devider: str, exact_path: list) -> str:
```

```
    """ Функция объединяет путь на основе символа-разделителя
```

```
    tmp_path = str(dir_path).split(system_devider)
```

```
    tmp_path = tmp_path[:len(tmp_path)]
```

```
    joined_path = [j for i in [tmp_path, exact_path] for j in i]
```

```
    goal_dir_path = system_devider.join(joined_path)
```

```
    return goal_dir_path
```

При помощи данной функции мы разделяем путь на элементы списка, где значениями являются наименования папок. Затем снова объединяем их уже с конкретной папкой при помощи специального разделителя нужной нам операционной системы.

Чтобы определить операционную систему прописываем следующую функцию:

```
def get_system_dir_path(goal_path: list, current_path: str=dir_path) -> str:
```

```
### Функция определяет разделитель и возвращает целевой путь до
директории
```

```
if '/' in curren_path:
    # Linux & MacOS
    system_dir_path = get_dir_path('/', goal_path)

    return system_dir_path

elif '\\' in curren_path:
    # Windows
    system_dir_path = get_dir_path('\\', goal_path)

    return system_dir_path

else:
    print('Error, cannot define path devider.')
```

Оставляем сообщение на случай, если что-то пойдёт не так и мы не сможем
правильно определить разделитель.

Далее инициализируем пути к папкам.

```
# Пути к папкам с данными и моделями
```

```
RAW_DATA_DIR_PATH = get_system_dir_path(['data', 'train', 'raw'])
```

```
CLEAN_DATA_DIR_PATH = get_system_dir_path(['data', 'train', 'clean'])
```

```
OUTPUT_DATA_DIR_PATH = get_system_dir_path(['data', 'output'])
```

```
CUSTOM_MODELS_DIR_PATH = get_system_dir_path(['models', 'custom'])
```

В будущем эти пути нам пригодятся при определении мест для сохранения
моделей и данных, а также мест откуда эти самые данные брать.

Следующим модулем, с которым нам следует разобраться, является `preprocessor.py`

В качестве тренировочных данных используются спарсенные комментарии в социальной сети ВК из группы дтпчп, а также набор жалоб на городском портале.

Как было отмечено нами ранее суть данного модуля заключается в обработке первичных данных и преобразовании их в обучающий датасет для используемой нами модели.

Импорт необходимых библиотек представлен ниже.

```
import re
import pandas as pd
import nltk
import nltk.data
import pymorphy2
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, RegexpTokenizer
from pathlib import Path

from paths import RAW_DATA_DIR_PATH
```

Также нам необходимо загрузить в модуль пунктуацию и стопслова из `nltk`.

```
nltk.download('punkt')
nltk.download('stopwords')
```

Далее нами был создан и инициализирован класс процессора.

```
class Preprocessor:
```

```
    """
```

Класс для предобработки данных из расширений файлов `.csv` и `.txt`.

```
    """
```

```
def __init__(self, filename: str, column: str=' ', delimiter: str=';'):
```

```

# Функция инициализирует класс.

self.file_path = str(Path(RAW_DATA_DIR_PATH).joinpath(filename)) # Путь
к файлу

self.tokenizer = nltk.data.load('tokenizers/punkt/russian.pickle') # Получение
токенизатора

self.column = column # Наименование колонки для работы с файловыми
расширениями .csv

self.delimeter = delimiter # Разделитель для работы с файловыми
расширениями .csv

```

Данный код представляет класс с именем Preprocessor, который служит для предобработки данных из файлов с расширениями .csv и .txt. Ниже приведено краткое описание каждой части кода:

Метод `__init__`:

Это конструктор класса, который инициализирует объекты данного класса при его создании.

Принимает параметры: `filename` (строка с именем файла), `column` (строка с наименованием колонки, используемой для работы с файлами формата .csv), `delimiter` (строка с разделителем для работы с файлами формата .csv).

Устанавливает атрибуты объекта, включая путь к файлу (`file_path`), токенизатор (`tokenizer` из библиотеки `nltk`), наименование колонки (`column`) и разделитель (`delimiter`).

Атрибуты:

`file_path`: Путь к файлу, который представляет собой комбинацию пути к директории `RAW_DATA_DIR_PATH` и переданного имени файла (`filename`).

`tokenizer`: Токенизатор, используемый для разбиения текста на токены. В данном случае, это токенизатор для русского языка из библиотеки `nltk`.

`column`: Наименование колонки, используемой при работе с файлами формата .csv.

`delimiter`: Разделитель, используемый при работе с файлами формата .csv.

Таким образом, данный класс предоставляет общую структуру для предобработки данных из файлов с расширениями .csv и .txt, а его атрибуты и методы можно использовать для работы с соответствующими файловыми форматами.

Затем переходим к методам предобработки данных.

```
def review_to_word_list(self, review: str, remove_stopwords: bool=True) -> list:  
    # Функция преобразует предложение в список слов и возвращает этот  
    список.  
  
    # Избавление от лишнего в данных  
    review = re.sub(r"http[s]?://(?:[a-zA-Z][0-9]][$-_@.&+]|[*\(\)],](?:%[0-9a-fA-F][0-9a-fA-F])+", " ", review)  
  
    review_text = BeautifulSoup(review, "lxml").get_text()  
    review_text = re.sub("[^а-яА-Я]", " ", review_text)  
  
    words = review_text.lower().split()  
  
    if remove_stopwords:  
        stops = stopwords.words("russian")  
        words = [w for w in words if w not in stops]  
  
    return words  
  
def review_to_sentence(self, review: str, remove_stopwords: bool=False) -> list:  
    # Функция преобразует текст в список предложений и возвращает этот  
    список.  
  
    raw_sentences = self.tokenizer.tokenize(review.strip())
```

```
sentences = []

for raw in raw_sentences:
    if len(raw_sentences) > 0:
        sentences.append(self.review_to_word_list(raw, remove_stopwords))

return sentences
```

Этот код содержит два метода внутри класса, и оба предназначены для предобработки текста. Давайте разберем каждый метод:

Метод `review_to_word_list`:

Описание: Этот метод принимает строку текста (`review`) и преобразует ее в список слов, выполняя несколько этапов предобработки.

Шаги предобработки:

Избавление от ссылок в тексте с использованием регулярного выражения.

Преобразование HTML-разметки в обычный текст с помощью `BeautifulSoup`.

Удаление всех символов, не являющихся буквами русского алфавита.

Приведение текста к нижнему регистру.

Разделение текста на слова.

Опционально удаление стоп-слов (если `remove_stopwords=True`).

Возвращаемое значение: Список слов.

Метод `review_to_sentence`:

Описание: Этот метод принимает строку текста (`review`) и преобразует ее в список предложений, используя токенизатор (`tokenizer`). Каждое предложение представлено списком слов.

Шаги:

Использование токенизатора для разделения текста на предложения.

Для каждого предложения вызывается метод review_to_word_list, который преобразует предложение в список слов.

Возвращаемое значение: Список предложений, каждое из которых представлено списком слов.

При помощи следующей функции мы получаем список слов в их начальной форме.

```
def normalize_words(self, sentences_list: list) -> list:  
    #Функция приводит слова в начальную форму русского языка и  
    #возвращает двумерный список предложений.  
  
    morph = pymorphy2.MorphAnalyzer()  
  
    for i in sentences_list:  
        for j in i:  
  
            form_list = morph.normal_forms(j)  
            index = i.index(j)  
            i[index] = form_list[0]  
  
    return sentences_list
```

Подробнее:

Метод normalize_words:

Описание: Этот метод принимает двумерный список предложений (sentences_list), где каждое предложение представлено списком слов. Затем он приводит каждое слово в начальную форму русского языка с использованием библиотеки PyMorphy2.

Шаги:

Создается экземпляр класса MorphAnalyzer из библиотеки PyMorphy2.

Для каждого слова в каждом предложении вызывается метод normal_forms, который возвращает список нормальных форм данного слова.

Исходное слово в предложении заменяется на его нормальную форму.

Возвращаемое значение: Двумерный список предложений, где каждое слово приведено в начальную форму русского языка.

Этот метод осуществляет лемматизацию (приведение слов к их базовым формам) входного списка предложений.

Затем используем финальную функцию, которая возвращает непосредственно датасет для обучения.

```
def clean_file(self) -> list:  
    # Функция очищает файл с расширением .csv или .txt и приводит его в вид  
    # датасета для обучения модели, возвращает двумерный список.  
  
    preprocessed_data = []  
  
    # Если файл .csv  
    if '.csv' in self.file_path:  
        clean_sents = []  
  
        # Инициализация DataFrame  
        data = pd.read_csv(self.file_path,  
                           delimiter=self.delimeter).dropna(subset=self.column)  
  
        print("Parsing sentences from training set...")  
  
        # Очистка текста  
        for review in data[self.column]:  
            clean_sents += self.review_to_sentence(review, self.tokenizer)  
  
        preprocessed_data = self.normalize_words(clean_sents)  
  
    # Если файл .txt  
    elif '.txt' in self.file_path:
```

```

with open(self.file_path, 'r', encoding='utf-8') as f:
    text = f.read()

    text = re.sub('\n', ' ', text)
    sents = sent_tokenize(text)

    punct = '!"#$%&()*+,.;=>?@[\\]^`{|}~,“«»†*—^-“'
    clean_sents = []

# Очистка текста
print("Parsing sentences from training set...")

for sent in sents:
    s = [w.lower().strip(punct) for w in sent.split()]
    clean_sents.append(s)

preprocessed_data = self.normalize_words(clean_sents)

# Если расширение файла неизвестно
else:
    print('Datatype is not supported for preprocessing training data.\nPlease, use
.csv or .txt file')

return preprocessed_data

```

Метод clean_file:

Описание: Этот метод выполняет очистку файла с расширением .csv или .txt и преобразует его в формат датасета, который может быть использован для обучения модели. Результат представляет собой двумерный список, где каждое предложение представлено списком слов в начальной форме.

Шаги:

Если файл имеет расширение .csv, метод использует библиотеку Pandas для чтения данных из файла и очистки текста, вызывая метод review_to_sentence для разделения текста на предложения и метод normalize_words для лемматизации слов.

Если файл имеет расширение .txt, метод считывает текст из файла, разбивает его на предложения и слова, а затем проводит лемматизацию слов с использованием метода normalize_words.

Если расширение файла не .csv или .txt, выводится сообщение о том, что данный тип файла не поддерживается.

Возвращаемое значение: Двумерный список предложений, где каждое слово приведено в начальную форму русского языка.

С модулем препроцессора на этом всё. Далее переходим непосредственно к модулю обучения модели train_model.py.

Импорт:

```
from gensim.models import word2vec  
from pathlib import Path  
  
from preprocessor import Preprocessor  
import paths
```

Инициализация класса:

```
class Custom_model:  
    ""  
  
    Основной класс библиотеки, создающий и тренирующий модели.  
    ""  
  
    def __init__(self, file_name: str='posts_spb_today.csv', column:str='text', workers:  
int=4, min_count: int=5, window:int=10, sample: float=1e-3):  
        # Функция инициализирует класс  
  
        self.file_name = file_name # Имя файла  
        self.workers = workers # Число потоков
```

```
    self.min_count = min_count # Минимальное количество повторения слов  
для вхождения в корпус модели  
    self.window = window # Окно наблюдений  
    self.sample = sample # Размер downsampling'a частовстречающихся слов  
    self.column = column # Наименование колонки для работы с файловыми  
расширениями .csv
```

В класс сразу же сообщаются основные параметры модели и информация о данных для обучения.

Затем описываем функцию создания модели.

```
def make_model(self) -> word2vec.Word2Vec:  
    # Функция создаёт и возвращает модель класса Word2Vec, необходимо  
сохранять в переменную.  
  
    training_data = Preprocessor(self.file_name, self.column).clean_file()  
    model = word2vec.Word2Vec(training_data, workers=self.workers,  
min_count=self.min_count, window=self.window, sample=self.sample)  
  
    return model
```

Метод `make_model`:

Описание: Этот метод создает и возвращает модель класса Word2Vec из библиотеки gensim. Модель обучается на предварительно очищенных данных, полученных из файла с использованием класса Preprocessor.

Шаги:

Создается экземпляр класса Preprocessor для предварительной очистки данных из файла с использованием метода `clean_file`.

Модель Word2Vec создается, используя очищенные данные:

`training_data`: Двумерный список предложений, представляющих собой подготовленные текстовые данные.

`workers`: Количество потоков, указанное при инициализации объекта класса `Custom_model`.

`min_count`: Минимальное количество повторений слова, указанное при инициализации объекта класса `Custom_model`.

`window`: Размер окна наблюдения, указанный при инициализации объекта класса `Custom_model`.

`sample`: Размер downsampling'a для частовстречающихся слов, указанный при инициализации объекта класса `Custom_model`.

Возвращаемое значение: Обученная модель класса `Word2Vec`.

Этот метод позволяет создать и обучить модель `Word2Vec` на предварительно подготовленных текстовых данных из файла с использованием параметров, указанных при инициализации объекта класса `Custom_model`.

После создания модели её необходимо сохранить, чтобы не пришлось её обучать заново с нуля.

```
def save_model(self, model: word2vec.Word2Vec,  
model_name:str='model.model') -> None:  
  
    # Функция сохраняет модель в небинарный файл.  
  
    model_path =  
    str(Path(paths.CUSTOM_MODELS_DIR_PATH).joinpath(model_name))  
  
    model.save(model_path)  
  
    print(f'Model saved as {model_name} in  
{paths.CUSTOM_MODELS_DIR_PATH}')
```

Метод `save_model`:

Описание: Этот метод сохраняет модель `Word2Vec` в небинарный файл.

Параметры:

`model` (объект `Word2Vec`): Обученная модель `Word2Vec`, которую необходимо сохранить.

`model_name` (строка): Имя файла, под которым сохранится модель (по умолчанию 'model.model').

Шаги:

Генерируется путь к месту сохранения модели с использованием библиотеки Path и указанного имени файла.

Модель сохраняется в указанном месте с использованием метода save.

Выводится сообщение о том, что модель была успешно сохранена.

Возвращаемое значение: None (ничего).

Этот метод обеспечивает сохранение обученной модели Word2Vec в небинарный файл в указанной директории для последующего использования или дальнейшего обучения.

Если в будущем понадобится дообучить модель, мы сможем это сделать при помощи функции тренировки.

```
def train_model(self, model_name: str='model.model', training_data_name: str='total_reports.csv', column: str='Текст', epochs: int=5) -> word2vec.Word2Vec:  
    # Функция тренирует предобученную пользовательскую модель и  
    возвращает новую, необходимо записывать в переменную.
```

```
model_path =  
str(Path(paths.CUSTOM_MODELS_DIR_PATH).joinpath(model_name))
```

```
# Загрузка модели  
model = word2vec.Word2Vec.load(model_path)
```

```
# Предобработка данных  
training_data = Preprocessor(training_data_name, column).clean_file()
```

```
# Создание словаря модели и тренировка на датасете
```

```
model.build_vocab(training_data, update=True)
```

```
model.train(training_data, total_examples=model.corpus_count, epochs=epochs)
```

```
return model
```

Метод train_model:

Описание: Этот метод тренирует предварительно обученную пользовательскую модель Word2Vec, используя новые данные, и возвращает обновленную модель.

Параметры:

model_name (строка): Имя файла с предварительно обученной моделью Word2Vec (по умолчанию 'model.model').

training_data_name (строка): Имя файла с новыми данными для обучения модели (по умолчанию 'total_reports.csv').

column (строка): Наименование колонки в файле новых данных для работы с текстовыми данными (по умолчанию 'Текст').

epochs (целое число): Количество эпох обучения (по умолчанию 5).

Шаги:

Генерируется путь к месту, где хранится предварительно обученная модель Word2Vec.

Загружается предварительно обученная модель с использованием метода Word2Vec.load.

Создается экземпляр класса Preprocessor для предварительной очистки новых данных.

Строится словарь модели, обновляя его с использованием новых данных.

Модель обучается на новых данных с использованием метода train.

Возвращаемое значение: Обновленная модель Word2Vec.

Этот метод позволяет дообучить предварительно обученную модель Word2Vec на новых данных, добавляя эти данные к существующей модели и обучая ее в течение указанного числа эпох.

Больше функция в классе Custom_model нам не потребуется. Нам осталось только визуализировать имеющиеся данные. Для этой задачи мы создали модуль visualization.py.

Импорт:

```
import csv
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
from gensim.models import word2vec

from paths import CUSTOM_MODELS_DIR_PATH,
OUTPUT_DATA_DIR_PATH
```

Инициализация класса:

```
class Visualiztor:
```

```
    """
```

Класс визуализирует данные модели в граф знаний на основе подающегося в него слова и наиболее близких к нему слов.

```
    """
```

```
    def __init__(self, word: str, model_name: str='trained_model.model', depth: int=2,
topn: int=10):
```

Функция инициализирует класс.

```
        self.word = word # Слово, от которого строится граф
        self.model_path =
str(Path(CUSTOM_MODELS_DIR_PATH).joinpath(model_name)) # Путь к
модели Word2Vec
```

```
        self.depth = depth # Глубина отрисовки графа
```

```
        self.topn = topn # Количество ближайших слов для поиска
```

Для работы с networkx нам необходимо выделить и сохранить наши узлы в csv файл, из которого уже потом при помощи networkx мы сможем собрать граф. Обратимся к коду ниже.

```

def wright_nodes_in_csv(self):
    # Функция записывает связанные слова в качестве узлов графа в файл с
    # расширением .csv, а косинусную близость в качестве направления.

    # Загрузка модели
    model = word2vec.Word2Vec.load(self.model_path)

    # Изолирование ключевого слова
    center = [self.word]

    # Создание файла с расширением .csv для записи узлов и направлений
    with open(str(Path(OUTPUT_DATA_DIR_PATH).joinpath('graph.csv')), 'w+', encoding='utf-8') as file:

        writer = csv.writer(file, delimiter=';')
        writer.writerow(['subject', 'object', 'semantic_closeness'])

        new_list = []

        # Итерация глубины
        for i in range(self.depth):

            # Итерация количества связанных слов
            loads = len(center)
            for k in range(loads):
                nodes = model.wv.most_similar(center[k], topn=self.topn)

                # Запись каждого узла в файл
                for j in nodes:

```

```

line = [center[k], j[0], j[1]]
writer.writerow(line)
new_list.append(j[0])

center = new_list
new_list = []

print('Wrote csv.')

```

Метод wright_nodes_in_csv:

Описание: Этот метод записывает связанные слова в качестве узлов графа в файл с расширением .csv, а косинусную близость в качестве направления.

Шаги:

Загружается предварительно обученная модель Word2Vec с использованием метода Word2Vec.load.

Задается ключевое слово (center), для которого будут найдены связанные слова.

Создается файл с расширением .csv для записи узлов и направлений. Заголовок файла содержит три колонки: 'subject', 'object', 'semantic_closeness'.

Итерации проводятся в цикле для каждого уровня глубины (self.depth), где глубина указывает на количество "шагов" по связанным словам от исходного слова.

Для каждого слова в текущем уровне:

Используется метод most_similar для получения связанных слов (nodes) с их косинусными близостями.

Каждая связь записывается в файл .csv в виде строки с тремя значениями: исходное слово (center[k]), связанное слово (j[0]), косинусная близость (j[1]).

Связанные слова добавляются в список new_list.

Обновляется список center для следующего уровня глубины.

Возвращаемое значение: None (ничего). Вместо возвращения результатов, метод выводит сообщение о завершении операции.

Этот метод использует предварительно обученную модель Word2Vec для создания графа слов, где узлы представляют собой слова, а направления между ними - косинусные близости. Полученные данные сохраняются в файл .csv.

Таким образом, мы записали наши узлы в файлы, теперь необходимо их преобразовать в граф, который затем мы сможем сохранить в изображение.

Для этого нами была описана функция сохранения графа в виде изображения.

```
def save_graph_img(self, img_name: str='kg1.jpg', options: dict={'node_color':  
'yellow',    # Цвет узлов  
              'node_size': 1000,      # Размер узлов  
              'width': 1,           # Ширина линий  
              'arrowstyle': '-|>',   # Стиль стрелки  
              'arrowsize': 18,        # Размер стрелки  
              'edge_color':'blue',   # Цвет связи  
              'font_size':20         # Размер шрифта  
            }):  
  
    # Функция создаёт и сохраняет граф в изображение на основе записанных  
    # данных.
```

```
# Запись в файл узлов и направлений  
self.wright_nodes_in_csv()  
  
# Инициализация DataFrame  
columns = ['subject', 'object', 'semantic_closeness']  
df = pd.read_csv(str(Path(OUTPUT_DATA_DIR_PATH).joinpath('graph.csv')),  
                 delimiter=';', names=columns, encoding='utf-8')
```

```
# Создание объекта для записи графа
```

```
G=nx.from_pandas_edgelist(df,"subject","object", edge_attr=True,  
create_using=nx.MultiDiGraph())  
  
# Визуализация в изображение  
plt.figure(figsize=(30,30))  
pos = nx.spring_layout(G)  
  
# Визуализация графа в изображение и сохранение  
nx.draw(G, with_labels=True, pos = pos, **options)  
nx.draw_networkx_edge_labels(G, pos=pos)  
  
# Сохраняем картинку  
plt.savefig(img_name)  
  
print('Image is ready.')
```

Параметры:

img_name (строка): Имя файла для сохранения изображения графа (по умолчанию 'kg1.jpg').

options (словарь): Параметры визуализации графа (по умолчанию содержит различные стили для узлов, связей и т. д.).

Шаги:

Вызывается метод wright_nodes_in_csv для записи связанных слов в файл .csv.

Инициализируется объект DataFrame (df) для чтения данных из файла .csv.

Создается направленный граф (G) с использованием библиотеки NetworkX на основе данных из DataFrame.

Инициализируется графическое окно Matplotlib и задается размер изображения.

Вычисляется позиция узлов графа с использованием алгоритма распределения по кругу (spring_layout).

Граф визуализируется с помощью nx.draw с указанными параметрами в словаре options.

Надписи к узлам добавляются с помощью nx.draw_networkx_edge_labels.

Сохраняется изображение графа в файл с указанным именем (img_name).

Выводится сообщение о том, что изображение графа готово.

Возвращаемое значение: None (ничего).

Этот метод создает и сохраняет изображение графа, визуализированного на основе данных, записанных в файл .csv. Параметры визуализации задаются в словаре options.

Ниже представлен пример скрипта для использования библиотек.

```
from paths import RAW_DATA_DIR_PATH  
from train_model import Custom_model  
from visualization import Visualiztor
```

```
# Инициация модели.
```

```
model = Custom_model().make_model()
```

```
# Нахождение косинусной близости слов.
```

```
print(model.wv.similarity('полицейский', 'тротуар'))
```

```
# Сохранение модели.
```

```
m = Custom_model()  
m.save_model(model=model)
```

```
# Тренировка существующей модели на новых данных.
```

```
trained_model = m.train_model(model_name='model.model')
```

```
# Проверка результатов тренировки.
```

```
print(trained_model.wv.similarity('полицейский', 'тротуар'))
```

```
# Сохранение дотренированной модели.  
m.save_model(model=trained_model, model_name='trained_model.model')
```

```
# Построение графа знаний от указанного слова или списка слов.  
Visualiztor(word='тротуар').save_graph_img()
```

Вывод можно наблюдать на рисунке 2.

```
PS D:\GB\Python_course\final_tasks\diploma & d:/GB/Python_course/final_ta  
sk/diploma/venv/Scripts/python.exe d:/GB/Python_course/final_tasks/diplom  
a/src/use_example.py  
[nltk_data] Downloading package punkt to  
[nltk_data]     C:\Users\thebe\AppData\Roaming\nltk_data...  
[nltk_data]   Package punkt is already up-to-date!  
[nltk_data] Downloading package stopwords to  
[nltk_data]     C:\Users\thebe\AppData\Roaming\nltk_data...  
[nltk_data]   Package stopwords is already up-to-date!  
Parsing sentences from training set...  
d:\GB\Python_course\final_tasks\diploma\src\preprocessor.py:37: MarkupRese  
mblesLocatorWarning: The input looks more like a filename than markup. You  
may want to open this file and pass the filehandle into BeautifulSoup.  
    review_text = BeautifulSoup(review, "lxml").get_text()  
0.37735462  
Model saved as model.model in D:\GB\Python_course\final_tasks\diploma\mode  
ls\custom  
Parsing sentences from training set...  
d:\GB\Python_course\final_tasks\diploma\src\preprocessor.py:37: MarkupRese  
mblesLocatorWarning: The input looks more like a filename than markup. You  
may want to open this file and pass the filehandle into BeautifulSoup.  
    review_text = BeautifulSoup(review, "lxml").get_text()  
0.21578486  
Model saved as trained_model.model in D:\GB\Python_course\final_tasks\dipl  
oma\models\custom  
Wrote csv.  
Image is ready.  
PS D:\GB\Python_course\final_tasks\diploma []
```

Рис. 2

Как мы видим, после дообучения модели расстояние между словами *полицейский* и *тротуар* стало больше, то есть модель стала оценивать их менее схожими по смыслу.

На рисунке 3 представлен результат построения графа знаний с параметрами по умолчанию (*depth=2, topn=10*).

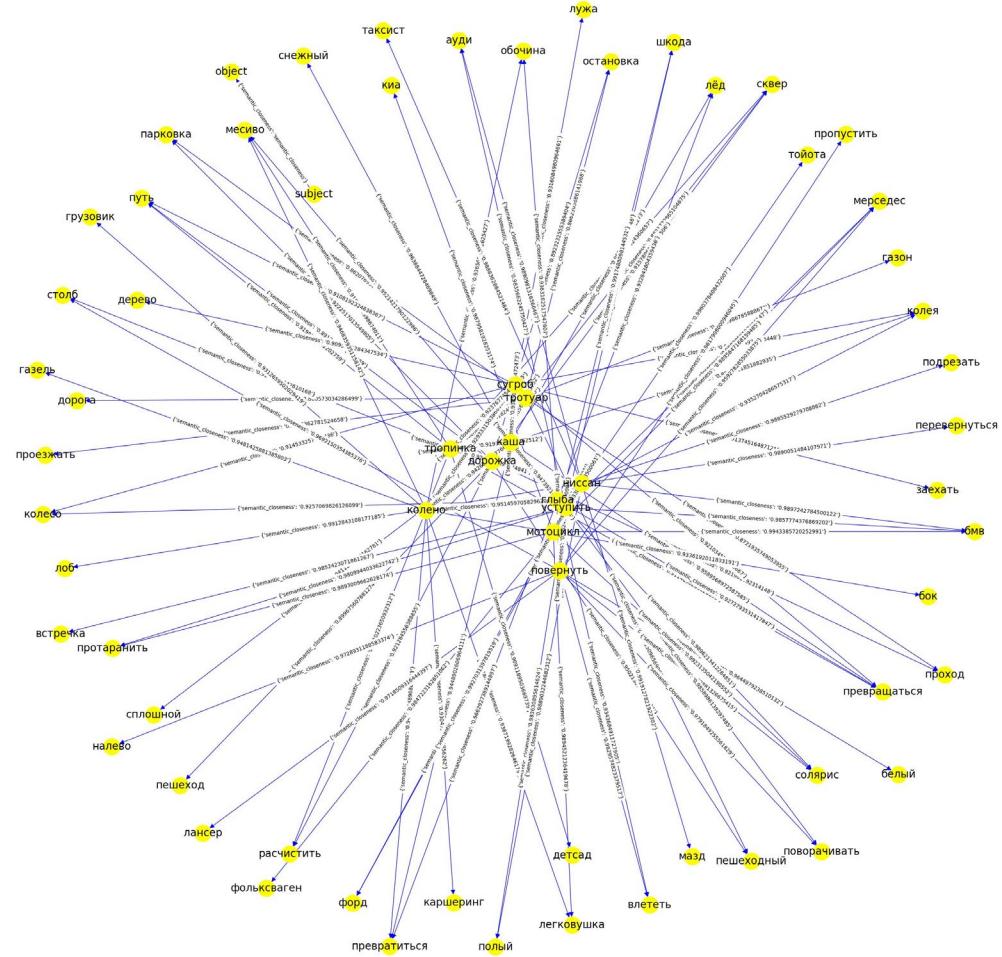


Рис. 3

Таким образом, в данном параграфе мы рассмотрели разработанный нами код и привели результаты его работы. Нам удалось предобработать данные, сделав из них датасет для обучения модели, а также получить граф знаний. Другие графы, отличные от представленного на рисунке 3 можно найти в приложении № 1, код доступен на https://github.com/LeonDeTur/GB_Filal_diploma.

ЗАКЛЮЧЕНИЕ

Нами была предоставлена информация о важности семантического анализа в современном мире, подчеркивая его роль в различных областях, таких как информационные технологии, медицина, бизнес и образование. Обсуждены применения, начиная от улучшенного поиска в интернете до создания более эффективных систем образования и интеллектуальных ассистентов.

Также были рассмотрены перспективы развития семантического анализа, включая интеграцию с глубоким обучением, обработку мультимодальных данных и использование в передовых технологиях, таких как трансформеры.

Рассмотрение визуализации семантического анализа выявило ее важную роль в облегчении восприятия результатов, выделении ключевых тем и слов, а также в поддержке коммуникации между исследователями и другими заинтересованными сторонами.

Нами был обоснован выбор в пользу использования Word2Vec ввиду таких причин как лёгкость модели, возможность использования в условиях ограниченных вычислительных мощностей, наличие готовых предобученных моделей, которые могут быть легко созданы пользователями на основе собственных обучающих корпусов, а также возможностями проведения различных операций над векторными представлениями слов.

Мы рассмотрели основные использованные нами модули при создании библиотеки. К ним относятся genism, nltk, networkx, pymorphy2, а также вспомогательные библиотеки pandas, lxml, csv, pathlib, bs4, re и Matplotlib. Соответственно, ввиду сложности и многоаспектности данных библиотек, автоматически были подтянуты другие модули в качестве зависимостей.

Мы также уточнили структуру проекта, выделили основным папки и директории, и обосновали сложности связанные с использованием библиотеки vec2graph.

Также нами был детально рассмотрен и прокомментирован код библиотеки для работы с обучением моделей Word2Vec и реализован механизм визуализации графа знаний.

Таким образом, все поставленные задачи были выполнены в полном объёме и, соответственно, цель работы достигнута.

СПИСОК ИСТОЧНИКОВ

1. Официальная документация Python / URL: <https://www.python.org/>
2. Semantic data model / URL:
https://en.wikipedia.org/wiki/Semantic_data_model
3. Официальная документация gensim: URL:
https://radimrehurek.com/gensim/auto_examples/index.html
4. Официальная документация nltk / URL: <https://nltk.readthedocs.io/en/latest/>
5. Официальная документация pymorphy2 / URL:
<https://pymorphy2.readthedocs.io/en/stable/>
6. Официальная документация networkx / URL: <https://networkx.org/>
7. Официальная документация pandas/ URL: <https://pandas.pydata.org/docs/>
8. Простой граф знаний на текстовых данных / URL:
<https://newtechaudit.ru/prostoj-graf-znanij-na-tekstovyh-danniyh/>
9. Что такое тезариус и как определить семантическое сходство слов / URL:
https://habr.com/ru/companies/unistar_digital/articles/687148/
- 10.Блокнот Google Colab 2_embeddings.ipynb / URL:
https://colab.research.google.com/github/PragmaticsLab/NLP-course-FinTech/blob/master/seminars/2/2_embeddings.ipynb#scrollTo=qqFhuv7XX-r1
- 11.Гольдберг В. Б. Эволюция структурной парадигматической модели лексико-семантической системы языка // Вестник ТГУ. 2002. №3. URL:
<https://cyberleninka.ru/article/n/evolyutsiya-strukturnoy-paradigmatischeskoy-modeli-leksiko-semanticeskoy-sistemy-yazyka> (дата обращения: 20.01.2024).
- 12.RusVectōrēs: семантические модели для русского языка / URL:
<https://rusvectores.org/ru/>
- 13.Официальная документация vec2graph / URL:
<https://pypi.org/project/vec2graph/>
- 14.Word2Vec vs BERT — Salt Data Labs / URL:
<https://www.saltdatalabs.com/blog/word2vec-vs-bert>

15. Семантический анализ для автоматической обработки естественного языка

/ URL: https://rdc.grfc.ru/2021/09/semantic_analysis/

16. Репозитарий библиотеки на GitHub / URL:

https://github.com/LeonDeTur/GB_Filal_diploma

Приложение № 1

