

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**РЕАЛИЗАЦИЯ СЕМАНТИЧЕСКОГО ПОИСКА НА КОРПУСЕ
НОВОСТНЫХ ДОКУМЕНТОВ**

КУРСОВАЯ РАБОТА

студента 3 курса 351 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Янченко Вадима Александровича

Научный руководитель
доцент, к. ф.-м. н.

С. В. Папшев

Заведующий кафедрой
доцент, к. ф.-м. н.

С. В. Миронов

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Алгоритмы построения эмбеддингов	5
1.1 Алгоритм модели Word2Vec	6
1.1.1 Архитектура модели Skip-Gram	6
1.2 Алгоритм модели GloVe.....	8
2 Понятие информационно-поискового тезауруса	10
3 Семантический поиск.....	11
3.1 Алгоритм работы семантического поиска	11
3.2 Формирование и хранение эмбеддингов документов	12
3.2.1 Получение эмбеддингов	12
3.2.2 Хранение эмбеддингов	13
4 Разработка системы семантического поиска	15
4.1 Предварительная обработка данных	15
4.2 Построение моделей на основе корпуса новостных документов	16
4.2.1 Построение модели Word2Vec	16
4.2.2 Построение модели GloVe	18
4.3 Сравнение тезаурусов	18
4.4 Расширение моделей	20
4.4.1 Получение эмбеддингов документов	21
4.5 Реализация алгоритма семантического поиска на основе корпуса новостей	22
5 Анализ работы семантического поиска на основе разных моделей	25
ЗАКЛЮЧЕНИЕ	27
Приложение А Программный код, выполняющий обработку текста	28
Приложение Б Построение тезауруса по модели эмбеддинга.....	32
Приложение В Получение случайных 20 слов	33
Приложение Г Построение графа	35
Приложение Д Расширение модели, построенной на корпусе новостных документов.....	37
Приложение Е Получение и сохранение эмбеддингов документов	38
Приложение Ж Реализация алгоритма семантического поиска.....	40

ВВЕДЕНИЕ

В современном мире объём текстовой информации стремительно растёт: ежедневно публикуются миллионы новостных статей, научных публикаций, блогов и других текстов. Это приводит к тому, что извлечение релевантной информации становится всё более сложной и ресурсоёмкой задачей. Особенно остро данная проблема стоит в сфере новостной аналитики, где важно не просто находить тексты по ключевым словам, но и понимать их смысл, контекст и скрытые связи между понятиями. В условиях информационного перенасыщения возрастаёт потребность в инструментах, которые способны анализировать тексты не только на уровне лексики, но и на уровне семантики.

Классические подходы к поиску информации, такие как полнотекстовый поиск, не всегда обеспечивают необходимый уровень точности и полноты, особенно в случае, когда пользовательский запрос выражен не теми словами, которые используются в целевых документах. Это может происходить из-за синонимии, омонимии, различий в стилистике и формулировках. В результате возникает необходимость в применении более интеллектуальных методов обработки естественного языка, которые позволяют учитывать смысловую близость слов и фраз.

Одним из наиболее перспективных направлений в области анализа текстов является использование эмбеддингов — векторных представлений слов, фраз и даже целых документов. Такие представления позволяют преобразовать текстовую информацию в числовой вид, сохраняющий семантические связи между элементами текста. Эмбеддинги, полученные с помощью моделей Word2Vec и GloVe, широко используются в задачах классификации, кластеризации, тематического моделирования и, в том числе, в семантическом поиске.

Семантический поиск, в отличие от традиционного, основывается не на простом совпадении слов, а на вычислении смысловой близости между запросом и содержанием документов. Это позволяет находить более релевантные результаты, учитывать контекст и преодолевать ограничения, связанные с различиями в формулировках. Особенно актуальным это становится в работе с новостными данными, где важна оперативность, полнота и точность информационного поиска.

Эмбеддинги, полученные с помощью моделей, которые обучены на огромном корпусе документов, например, Национальном корпусе русского языка

(НКРЯ), обладают высокой универсальностью и охватывают широкий спектр лексических и синтаксических конструкций. Однако такие модели формируют обобщённые представления слов и выражений, которые отражают усреднённые языковые закономерности, присущие общему языку. В результате они не способны в полной мере учитывать специфику отдельных предметных областей. Поэтому необходимо использовать эмбеддинги, обученные или дообученные на специализированных корпусах, отражающих специфику выбранной тематики — в данном случае, новостных текстов.

Таким образом, возникает проблема в создании модели, которая бы отражала наиболее полно предметную область, и, как следствие, лучше всего отвечала на запросы пользователей.

Цель курсовой работы — разработать систему семантического поиска по корпусу новостных документов на основе эмбеддингов.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить теоретические основы построения векторных представлений слов и документов, а также ознакомиться с архитектурами моделей Word2Vec и GloVe, исследовать понятие тезауруса и методы его формирования на основе эмбеддингов.
2. Построить модели Word2Vec и GloVe на корпусе новостных документов и проанализировать их.
3. Разработать алгоритм семантического поиска.
4. Провести сравнительный анализ моделей Word2Vec и GloVe в контексте задачи семантического поиска.

1 Алгоритмы построения эмбеддингов

В контексте обработки естественного языка слова рассматриваются как элементы, обладающие лексико-семантическими характеристиками. Базовой единицей анализа выступает лемма — начальная форма слова. От неё образуются различные словоформы, отражающие морфологические изменения (например, *дом* — *дома*, *дому* и т.д.).

Одна лемма может иметь несколько значений — явление, известное как полисемия. Это затрудняет однозначную интерпретацию и требует учёта контекста. Кроме того, в языке существуют слова, близкие по значению, — синонимы. Классическим формальным критерием синонимии считается взаимозаменяемость слов в высказывании без изменения его условий истинности. Однако на практике полная синонимичность встречается редко: лексемы могут различаться стилистически (например, *жена* — *супруга*) или семантически по оттенкам (*мокрый* — *влажный*).

Помимо синонимии важную роль играет сходство слов — более общее понятие, не предполагающее полной тождественности значений. Например, кошка и собака — не синонимы, но они семантически близки по ряду признаков (домашние животные, млекопитающие и т.д.). Анализ сходства между словами позволяет работать на уровне поверхностной лексики, не опираясь на формальные модели значений, что существенно упрощает вычислительную обработку.

Мера лексического сходства используется в широком спектре задач — от оценки семантической близости слов и фраз до построения более сложных моделей смыслового соответствия между предложениями и текстами. Именно на этих принципах основывается построение векторных представлений слов (эмбеддингов), где слова отображаются в пространстве таким образом, что семантически близкие слова располагаются ближе друг к другу.

Ключевая идея состоит в представлении слова как вектора в многомерном пространстве. Самый простой способ перехода от слова к вектору состоит в следующем. Пусть V — множество всех слов. Слово $w_i \in V$ представим как вектор размера $n = |V|$, в котором все координаты равны 0 кроме i -ой координаты, которая равна 1.

1.1 Алгоритм модели Word2Vec

Рассмотрим другой способ представление слова как вектора — эмбеддинг. Эмбеддинги имеет количество размерностей d от 50 до 1000, что существенно меньше количества слов $|V|$ во всех документах. Кроме того, эти векторы плотные: вместо того, чтобы большинство координат были равны 0, значения будут вещественными числами, которые могут быть и отрицательными.

Эмбеддинги Word2Vec являются статическими, это означает, что каждому слову соответствует ровно один эмбеддинг. Такие модели как BERT, основанные на архитектуре трансформеров, позволяют создавать контекстуальные эмбеддинги, которые сопоставляют разный вектор к слову в зависимости от контекста.

Модель Word2Vec реализуется в двух основных архитектурах: CBOW и Skip-Gram.

1.1.1 Архитектура модели Skip-Gram

Модель Skip-Gram является одной из двух основных архитектур Word2Vec и направлена на предсказание контекстных слов по центральному слову. То есть, имея слово w_t , модель обучается предсказывать слова, находящиеся в некотором окне вокруг него: $w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c}$, где c — размер окна.

Модель Skip-Gram представляет собой простую нейросеть с одним скрытым слоем. Она состоит из следующих компонентов:

1. Входной слой. Каждое слово из словаря V кодируется как one-hot вектор $\mathbf{x} \in \mathbb{R}^{|V|}$, в котором все элементы равны нулю, кроме одного, соответствующего индексу слова w .
2. Скрытый слой. Представлен матрицей весов $W \in \mathbb{R}^{|V| \times d}$, где d — размерность эмбеддингового пространства. Преобразование входа:

$$\mathbf{h} = \mathbf{x}^\top W$$

Вектор $\mathbf{h} \in \mathbb{R}^d$ и является искомым эмбеддингом слова w .

3. Выходной слой Представлен второй матрицей весов $W' \in \mathbb{R}^{d \times |V|}$. Результат выходного слоя:

$$\mathbf{u} = W'^\top \mathbf{h}$$

где $\mathbf{u} \in \mathbb{R}^{|V|}$ — логиты для каждого слова в словаре.

4. Softmax. На выходе применяется softmax-функция, преобразующая логиты в вероятности:

$$P(w_O | w_I) = \frac{\exp(\mathbf{v}'_{w_O} \cdot \mathbf{v}_{w_I})}{\sum_{w \in V} \exp(\mathbf{v}'_w \cdot \mathbf{v}_{w_I})}$$

Здесь:

- w_I — входное (центральное) слово;
- w_O — слово из контекста;
- \mathbf{v}_{w_I} — вектор из матрицы W (входной эмбеддинг);
- \mathbf{v}'_{w_O} — вектор из матрицы W' (выходной эмбеддинг).

Рассмотрим функцию потерь. Для одного примера (w_I, w_O) используется кросс-энтропия:

$$\mathcal{L} = -\log P(w_O | w_I)$$

Общая функция потерь по корпусу:

$$\mathcal{L}_{\text{total}} = - \sum_{t=1}^T \sum_{\substack{-c \leq j \leq c \\ j \neq 0}} \log P(w_{t+j} | w_t)$$

где T — длина корпуса.

Эмбеддингом является как матрица (W), так и матрица(W') — часто используется только W .

Стоит отметить, что несмотря на отсутствие явного задания семантики при обучении, модели типа Word2Vec способны выявлять семантические закономерности на основе статистических свойств текстового корпуса. Это объясняется тем, что модели обучаются на контекстных соотношениях между словами, отражающих их типичную совместную встречаемость.

Характерный пример — аналогичные отношения между словами: векторное представление слова «королева» может быть получено как результат векторной операции: король — мужчина + женщина = королева, где каждое слово представляется как эмбеддинг.

Другой пример: отношение «мужчина» : «женщина» аналогично отношению «дядя» : «тётя». Такие примеры иллюстрируют способность модели захватывать не только лексическую, но и семантическую структуру языка на уровне распределения контекстов.

Полноценный расчёт softmax-функции в модели Skip-Gram требует вычисления скалярного произведения с каждым словом в словаре, что приводит к высокой вычислительной сложности: $\mathcal{O}(|V|)$ на каждый пример. Для решения этой проблемы в модели Word2Vec используется приближённый метод оптимизации — negative sampling.

1.2 Алгоритм модели GloVe

Модель GloVe (Global Vectors) предназначена для построения векторных представлений слов на основе глобальной статистики совместных появлений слов в тексте. В отличие от моделей Word2Vec, которые обучаются на локальных контекстных окнах, GloVe использует информацию о числе совместных появлений слов во всём корпусе, агрегируя её в специальную матрицу.

На первом этапе строится матрица совместных появлений $X \in \mathbb{R}^{|V| \times |V|}$, где X_{ij} — количество раз, когда слово j встречается в контексте слова i . Контекст обычно ограничивается окном фиксированного размера (например, 5 слов влево и вправо), при этом можно учитывать взвешивание по расстоянию до центрального слова.

GloVe обучает два векторных представления: $w_i \in \mathbb{R}^d$ — для центрального слова i и $\tilde{w}_j \in \mathbb{R}^d$ — для контекстного слова j , а также два скалярных смещения b_i и \tilde{b}_j . Обучение направлено на аппроксимацию следующего равенства:

$$w_i^\top \tilde{w}_j + b_i + \tilde{b}_j \approx \log X_{ij}$$

Целевая функция, которую минимизирует модель, записывается в следующем виде:

$$J = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} f(X_{ij}) \left(w_i^\top \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

где $f(X_{ij})$ — весовая функция, ограничивающая влияние часто встречающихся пар слов, например:

$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^\alpha & \text{если } x < x_{\max} \\ 1 & \text{иначе} \end{cases}$$

На практике используются значения $\alpha = 0,75$ и $x_{\max} = 100$.

После завершения обучения итоговое представление слова i получается как сумма обученных векторов:

$$v_i = w_i + \tilde{w}_i$$

Таким образом, каждая лексема кодируется плотным вектором фиксированной размерности, отражающим её глобальные статистические связи с другими словами корпуса.

2 Понятие информационно-поискового тезауруса

В информационном поиске широко используется механизм расширения запроса, основанный на добавлении семантически связанных терминов. Это позволяет находить релевантные документы, даже если в них отсутствуют исходные ключевые слова.

Для определения связанных по смыслу слов применяется тезаурус — лексический ресурс, фиксирующий семантические отношения между словами (синонимия, антонимия, гипонимия и др.). В отличие от обычных словарей, тезаурусы структурируют связи между терминами.

Тезаурусы бывают экспертными и автоматически построенными с помощью статистических и нейросетевых методов. Последние особенно важны для областей с быстро меняющейся лексикой, таких как новостные тексты.

Один из наиболее известных тезаурусов — WordNet, содержащий около 155000 лексем, организованных в синсеты (совокупности синонимов). В WordNet синонимы считаются взаимозаменяемыми, если замена не меняет значения высказывания хотя бы в некоторых контекстах.

3 Семантический поиск

Семантический поиск — это современный этап эволюции систем информационного поиска, основной задачей которого является не просто нахождение документов, содержащих заданные пользователем слова, а выявление наиболее смыслосообразных и релевантных результатов. Суть этого подхода заключается в том, чтобы приблизить машинное восприятие текста к человеческому пониманию, опираясь не только на внешнюю форму запроса, но и на его содержание, контекст и предполагаемое значение.

На протяжении десятилетий традиционные поисковые алгоритмы основывались на прямом сопоставлении слов из пользовательского запроса с документами. Такой подход, хоть и прост в реализации, имеет ряд существенных ограничений, особенно в тех случаях, когда пользователь использует неточные формулировки, обобщённые термины или специфические выражения. В реальности человек крайне редко формулирует свой запрос с использованием тех же терминов, которые использованы в искомом документе. Это создает разрыв между запросом и ответом, который семантический поиск стремится преодолеть.

Семантический поиск позволяет выходить за пределы лексического уровня языка, поднимаясь на уровень смысла и понятий. Это означает, что система стремится не к буквальному, а к содержательному совпадению. Происходит своего рода «перевод» текста на универсальный язык смыслов, в котором сравнение осуществляется не между словами, а между их внутренними представлениями. Таким образом, даже если запрос и документ имеют разную формулировку, они могут быть распознаны как близкие по значению.

3.1 Алгоритм работы семантического поиска

Семантический поиск основывается на идее сопоставления смыслового содержания запроса и документов, а не их поверхностного совпадения по ключевым словам. В основе его работы лежит представление текста в виде векторных эмбеддингов, отражающих значения слов, фраз или целых предложений в числовой форме.

Классическая схема семантического поиска включает несколько ключевых этапов:

1. Предобработка текста.

На этом этапе происходит очистка текста, токенизация, приведение слов к базовой форме (лемматизация), а также удаление второстепенных элементов вроде стоп-слов. Это стандартный этап, подготавливающий данные к последующему преобразованию в вектора.

2. Преобразование в эмбеддинги.

И запрос, и документы кодируются векторными представлениями с помощью заранее обученной модели. Эти векторы отражают смысл текста и располагаются в общем семантическом пространстве, где близкие по значению тексты имеют схожие координаты.

3. Вычисление сходства.

После получения эмбеддингов производится сравнение запроса с документами. Как правило, используется косинусное сходство — метрика, измеряющая угол между векторами. Чем ближе векторы, тем выше семантическая релевантность.

4. Ранжирование результатов.

Документы сортируются по степени семантической близости к запросу. Дополнительно могут использоваться фильтры, бизнес-логика, временные ограничения и прочие факторы для уточнения выдачи.

Особенность семантического подхода в том, что он работает не только с совпадениями слов, но и с их значениями в контексте. Например, запрос «новости о выборах» может быть сопоставлен с документом, где слово «выборы» заменено на «голосование» или «избирательная кампания», что невозможно при классическом поиске по словам.

3.2 Формирование и хранение эмбеддингов документов

Для реализации семантического поиска необходимо, чтобы как пользовательские запросы, так и документы, по которым осуществляется поиск, были представлены в виде векторных эмбеддингов. Эмбеддинг документа — это числовое представление его смыслового содержания в виде вектора фиксированной размерности. Такой вектор позволяет сравнивать документ с другими объектами в пространстве признаков и находить наиболее близкие по смыслу тексты.

3.2.1 Получение эмбеддингов

Эмбеддинги документов могут быть получены разными способами, в зависимости от используемой языковой модели:

- Среднее по эмбеддингам слов: один из простейших подходов, когда для каждого слова в документе берётся эмбеддинг (например, с использованием Word2Vec или GloVe), а затем рассчитывается среднее значение по всем словам. Такой метод работает быстро, но теряет информацию о порядке слов и контексте.
- Модели предложений и документов: современные модели, такие как BERT и его производные, позволяют получать контекстно-зависимые представления целых предложений или документов. Обычно для этого используется [CLS]-токен (в случае BERT) или пуллинг эмбеддингов по всем токенам. Такие представления дают более точную оценку смысла текста.
- Фиксация эмбеддингов: после того как эмбеддинг для каждого документа получен, его можно сохранить отдельно от исходного текста, чтобы в дальнейшем не производить повторных вычислений.

3.2.2 Хранение эмбеддингов

Поскольку при семантическом поиске основная задача — быстро найти векторы, близкие к вектору запроса, необходимо эффективное хранение эмбеддингов и поддержка операций поиска по векторному пространству.

Существует несколько практических подходов к хранению эмбеддингов:

1. Наиболее простой способ — сохранить эмбеддинги в виде обычных массивов чисел (например, в формате .npy, .json, .csv) и сопоставить их с идентификаторами документов. Это может подойти для небольших коллекций, но неэффективно для быстрого поиска: при каждом запросе придётся сравнивать вектор запроса с каждым вектором в базе.
2. Чтобы обеспечить масштабируемость и быстродействие, эмбеддинги сохраняются в специальных индексных структурах, предназначенных для приближённого поиска ближайших векторов (ANN — Approximate Nearest Neighbors). Такие индексы позволяют значительно сократить количество сравниваемых векторов за счёт предварительной организации данных.

Наиболее популярные технологии:

- FAISS (Facebook AI Similarity Search) Библиотека от Facebook, которая позволяет создавать и хранить векторные индексы, включая кластеризацию, квантование и сокращение размерности. Поддерживает хранение миллионов векторов и эффективный поиск по ним.
- HNSW (Hierarchical Navigable Small World) Графовая структура, ко-

торая обеспечивает быстрый и точный поиск ближайших соседей. Поддерживается в библиотеках, таких как nmslib, annoy, и плагинах для Elasticsearch и других систем.

3. Для продвинутых приложений используются специализированные СУБД, разработанные специально для хранения и обработки эмбеддингов:

- Pinecone — облачная векторная БД, поддерживающая миллионы объектов, масштабирование и фильтрацию.
- Weaviate — векторная СУБД с возможностью выполнять семантический поиск и хранить связанные метаданные.
- Milvus — высокопроизводительная open-source система для работы с векторами.

4 Разработка системы семантического поиска

Весь программный код написан на языке программирования python версии 3.12.6. В программном коде используются следующие библиотеки:

- pandas для удобного управления данными;
- dask как быстрая альтернатива pandas, так как распараллеливает вычисления;
- natasha для предобработки текста
- gensim для работы с моделями;
- networkx для работы с графами;
- faiss для семантического поиска.

Ознакомиться со всем проектом можно в репозитории¹ в github.

4.1 Предварительная обработка данных

Рассмотрим данные, которые представляют из себя новости, взятые из официального сайта СГУ. Каждый документ характеризуется следующими параметрами:

- уникальный идентификатор новости;
- заголовок новости;
- необработанный текст новости;
- обработанный текст новости;

В таблице 4.1 приведён пример представления одной из новостей в используемом формате.

News_Id	News_Title	News_Text	News_Tokens
1	Студенты и сотрудники СГУ присоединились к шествию «Бессмертного полка»	9 мая состоялась общенародная акция «Бессмертный полк», в которой приняли...	["май состояться общенародный акция бессмертный полк который принять участие студент сотрудник "два год ограничение пандемия долгожданный шествие проходить традиционный формат ..."]

Таблица 4.1 – Пример представления одной из новостей в используемом формате

Для получения обработанного текста выполняются следующие этапы:

1. Удаление стоп-слов.
2. Лемматизация слов.
3. Удаление имен людей, городов, стран, наименования различных компаний.

¹<https://github.com/Mindero/Thesaurus-construction>

Стоп-слова — это слова, которые встречаются в языке чрезвычайно часто, но, как правило, не несут самостоятельной смысловой нагрузки в контексте информационного поиска или анализа текста. К таким словам относятся, например, союзы, предлоги, частицы, местоимения: «и», «в», «на», «что», «как», «это», и т.п. Их исключение позволяет снизить «шум» в текстах, повысить эффективность алгоритмов и сосредоточиться на лексемах, действительно значимых для понимания содержания документа.

Лемматизация, в свою очередь, представляет собой процесс приведения слова к его начальной, словарной форме — лемме. Это позволяет уменьшить количество уникальных словоформ в тексте, объединив различные грамматические формы одного и того же слова (например, «пошёл», «идёт», «шли» — к лемме «идти»). Благодаря лемматизации повышается точность анализа, поскольку слова с одинаковым значением, но разными формами, будут обрабатываться как единое понятие.

Программный код, выполняющий обработку текста, можно рассмотреть в приложении А.

Ознакомиться со статистикой полученных данных можно в табл. 4.2.

Статистика	Значение
Количество новостей	31077
Общее количество предложений	423175
Среднее количество предложений на новость	13.62
Общее количество токенов	4293732
Среднее количество токенов на новость	138.16
Максимальное количество токенов на новость	166
Минимальное количество токенов на новость	1
Количество уникальных токенов	45204

Таблица 4.2 – Статистика по полученным данным

4.2 Построение моделей на основе корпуса новостных документов

4.2.1 Построение модели Word2Vec

Обучим модель Word2Vec на обработанных текстах новостей. В качестве входа новостей будут подаваться предложения, размер получаемых эмбеддингов равен 300, а размер окна 5. Кроме этого, будут отброшены слова, которые встречаются менее 5 раз, потому что эмбеддинги таких слов не будут отражать смысл этого слова. Программный код, выполняющий это, расположен ниже.

```

1 import dask.dataframe as dd
2 import dask.bag as db
3 from gensim.models import Word2Vec
4 docs = dd.read_parquet("../output.pq/")
5 texts = docs['News_Tokens'].compute()
6 bag = db.from_sequence(texts)
7 sentences = bag.flatten().map(lambda x: x.split()).compute()
8 model = Word2Vec(sentences=sentences, vector_size=300, window=5,min_count=5,
   ↪ sg=0)
9 model.save('word2vec_sent_5_sg0.model')

```

Результатом обучения будет бинарный файл, в котором к каждому слову представлен эмбеддинг.

Получив модель, построим тезаурус. Ознакомиться с кодом, выполняющим это можно в приложении Б. В качестве результата получим таблицу, в которой каждая строка имеет вид: слово и его ближайшие синонимы. Отношение близости между синонимами в построенном тезаурусе не менее 0.7, что является хорошей отсечкой для удаления некорректных результатов модели и шума.

Первые 10 строчек полученного файла отражены в табл. 4.3. Второй элемент в паре равен косинусному расстоянию между словами.

Word	Most_Similar_Word
май	[('апрель', 0.8788188099861145), ('февраль', 0.8743687272071838), ('ноябрь', 0.8666709661483765), ('октябрь', 0.864989161491394), ('декабрь', 0.855202317237854), ('январь', 0.8505667448043823), ('июнь', 0.842693567276001), ('март', 0.8332036137580872), ('сентябрь', 0.8205611705780029)]
состояться	[('пройти', 0.8456324934959412)]
акция	[]
бессмертный	[('полк', 0.9408268928527832)]
полк	[('бессмертный', 0.9408268928527832)]
который	[]
принять	[('принимать', 0.8362864255905151)]
участие	[]
студент	[]

Таблица 4.3 – Первые 10 строчек тезауруса, построенного моделью Word2Vec

4.2.2 Построение модели GloVe

Для создания модели GloVe склонируем их официальный репозиторий². Далее выполним следующие команды, которые создают эмбеддинги для каждого слова:

```
1 build/vocab_count -min-count 5 -verbose 2 < corpus.txt > vocab.txt
2 build/cooccur -memory 4.0 -vocab-file vocab.txt -verbose 2 -window-size 7 <
   ↳ corpus.txt > cooccurrence.bin
3 build/shuffle -memory 4.0 -verbose 2 < cooccurrence.bin >
   ↳ cooccurrence.shuf.bin
4 build/glove -save-file vectors -threads 12 -input-file cooccurrence.shuf.bin
   ↳ -x-max 10 -iter 15 -vector-size 300 -binary 2 -vocab-file vocab.txt
   ↳ -verbose 2
```

Модель строится по тем же параметрам, что и Word2Vec, а именно:

- размер эмбеддинга равен 300;
- размер окна — 5;
- минимальная частота встречаемости слова — 5.

В качестве результата получим .txt файл, в котором к каждому слову представлен эмбеддинг

Аналогично разделу 4.2.1 построим тезаурус. Ознакомиться с кодом можно в приложении Б.

Первые 10 слов полученного файла отражены в табл. 4.4.

Word	Most_Similar_Word
май	[('апрель', 0.7346850037574768)]
состояться	[]
акция	[]
бессмертный	[('полк', 0.8441691398620605)]
полк	[('бессмертный', 0.844169020652771)]
который	[]
принять	[('участие', 0.7851893305778503)]
участие	[('принять', 0.7851892709732056)]
студент	[]

Таблица 4.4 – Первые 10 строчек тезауруса, построенного моделью GloVe

4.3 Сравнение тезаурусов

Возьмём случайные 10 слов и найдём самые близкие слова по мнению каждой модели. Ознакомиться с программным кодом, который выполняет это

²<https://github.com/stanfordnlp/GloVe>

можно в приложении В. Результат

Полученный результат отражен на табл. 4.5.

Word	GloVe	Word2Vec
баян	[('аккордеон', 0.84)]	[('флейта', 0.89), ('балалайка', 0.88), ('аккордеон', 0.88), ('саксофон', 0.88), ('ария', 0.87), ('барабан', 0.86), ('гитара', 0.86), ('бас', 0.86), ('фортепиано', 0.86), ('укулела', 0.86)]
голод	[('холод', 0.7)]	[('холод', 0.85), ('вражеский', 0.84), ('танк', 0.83), ('убийство', 0.83), ('отважный', 0.83), ('наносить', 0.83), ('жуткий', 0.83), ('фашист', 0.83), ('умерший', 0.83), ('навеки', 0.82)]
орфография	[('пунктуация', 0.86)]	[]
внимательность	[('сосредоточенность', 0.74)]	[('быстро', 0.81), ('коммуникабельность', 0.8)]
копчёный	[('жареный', 0.78)]	[('жареный', 0.98), ('сметана', 0.93), ('жирный', 0.92), ('еловый', 0.91), ('морс', 0.91), ('слабогазировать', 0.91), ('смола', 0.91), ('перила', 0.9), ('сапог', 0.9), ('компот', 0.9)]
тяга	[('становой', 0.84)]	[('становой', 0.84)]
кадмий	[('селенид', 0.88)]	[('селенид', 0.95), (' силанизировать', 0.91), ('рений', 0.9), ('альбумин', 0.87), ('модулятор', 0.87), ('коллаген', 0.87), ('железоитриевый', 0.86), ('поликристаллический', 0.86), ('вычислять', 0.86), ('ватерита', 0.86)]
альпинизм	[('скалолазание', 0.76)]	[('скалолазание', 0.91)]
принять	[('участие', 0.79)]	[('принимать', 0.84)]
жажды	[('утолять', 0.79)]	[('великодушие', 0.81)]

Таблица 4.5 – Самых близких слов по мнению GloVe и Word2Vec для 10 случайных слов из корпуса

Видим, что в целом модель GloVe даёт более точные ответы. Кроме этого, модель Word2Vec предлагает сильно больше близких слов, большинство из которых ничего общего с рассматриваемым словом не имеет. Заметим, что существуют слова, для которых модель дает совсем неправильные ответы (слово *жажды*, *внимательность* и другие).

Для дальнейшего сравнения моделей построим по каждой модели граф. Вершиной в графе является слово, а ребро — факт того, что слово является близким для рассматриваемого с весом, равным косинусному расстоянию между ними. Граф является неориентированным. С программным кодом, генерирующим подобный график, можно ознакомиться в приложении Г.

Характеристики графов отражены в табл. 4.6.

Визуализируем полученные графы с помощью программы Gephi — бесплатный пакет программ для анализа и визуализации сетей. Ознакомиться с результатом можно на рисунках 4.1а, 4.1б. На рисунках цветами выделены кла-

Характеристика	GloVe	Word2Vec
Кол-во вершин	18402	18401
Кол-во ребер	2793	45709
Кол-во неизолированных вершин	2183	7533
Кол-во компонент (без учёта изолированных вершин)	688	360
Кол-во сообществ	734	379
Модулярность	0.618	0.612

Таблица 4.6 – Сравнительная характеристика графов моделей

стеры.

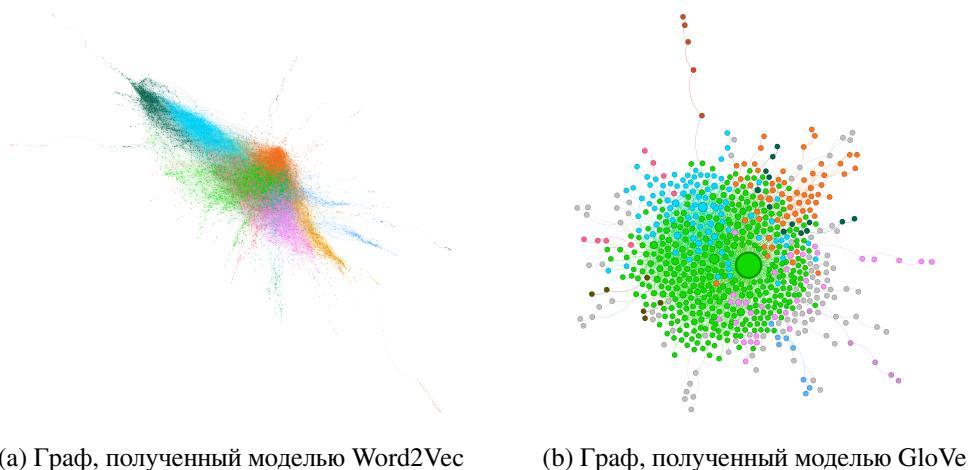


Рисунок 4.1 – Графы, полученные моделями Word2Vec и GloVe

4.4 Расширение моделей

Ограниченный объём входных данных существенно сказывается на качестве получаемых векторных представлений. Модели, такие как Word2Vec или GloVe, эффективно обучаются только при наличии большого и разнообразного корпуса текстов, содержащего широкий спектр лексики и контекстов. В условиях малого корпуса наблюдаются следующие проблемы:

- Слабое покрытие словаря: многие слова, особенно редкие или специфические для определённых тематик, могут вовсе не встретиться в обучающих данных. В результате модель не сможет построить для них эмбеддинги, и они будут игнорироваться при дальнейшей обработке.
- Нестабильность эмбеддингов: даже для слов, входящих в корпус, контексты их употребления могут быть слишком ограниченными, чтобы обеспечить надёжную статистику. Это приводит к шумным или недостоверным

векторным представлениям.

- Недостаточная семантическая выразительность: малая выборка не позволяет модели выявить устойчивые семантические закономерности между словами, что снижает эффективность таких задач, как кластеризация, классификация или семантический поиск.

Для решения данной проблемы возможно два подхода:

1. Увеличение объёма обучающего корпуса, например, за счёт сбора дополнительных новостных текстов, включая архивные данные, региональные источники или тематические агрегаторы.
2. Использование предварительно обученной модели (pretrained embeddings), полученной на большом и разнообразном корпусе (например, на Википедии, НКРЯ и т.д.), с последующим дообучением (fine-tuning) на новостных данных. Такой подход позволяет объединить общее языковое знание с контекстом конкретной предметной области.

Возьмем модель³ ruscorpora_upos_skipgram_300_5_2018, которая была обучена на данных из НКРЯ, с объемом словаря равным 195071 слов. Эту модель будем сливать с моделью Word2Vec и отдельно с Glove, которые обучены на корпусе новостных документов.

Алгоритм получения эмбеддинга в модели, являющейся соединением модели ruscorpora_upos_skipgram_300_5_2018 и модели Word2Vec или GloVe, следующий:

1. Если эмбеддинг слова есть в обоих моделях, то используется среднее арифметическое векторов из обеих моделей.
2. Если эмбеддинг слова есть только в одной из моделей, в итоговой модели используется вектор из той модели, в которой он имеется.

С кодом, выполняющим выше описанный алгоритм, можно ознакомиться в приложении Д:

4.4.1 Получение эмбеддингов документов

Для представления целого документа в векторном виде используется один из базовых и широко применяемых методов — усреднение эмбеддингов слов, входящих в данный документ. Суть подхода заключается в следующем: каждое слово в тексте заменяется своим вектором (эмбеддингом), после чего все эти

³<https://rusvectores.org/ru/models/>

векторы суммируются и результат делится на количество слов. Формально:

$$\bar{d} = \frac{1}{n} \sum_{i=1}^n \bar{w}_i$$

Полученные эмбеддинги документов сохраним в векторном индексе. С программным кодом можно ознакомиться в приложении Е.

4.5 Реализация алгоритма семантического поиска на основе корпуса новостей

Перед началом поиска осуществляется загрузка векторных индексов, содержащих эмбеддинги документов, а также моделей эмбеддингов слов. Кроме того, загружается корпус новостных документов, создаётся отображение от идентификатора новости к её заголовку для последующего вывода результатов.

Алгоритм семантического поиска реализуется в несколько этапов:

1. **Предобработка пользовательского запроса.** Выполняется лемматизация и токенизация текста запроса:
 - удаляются знаки препинания,
 - производится приведение к нижнему регистру,
 - слова приводятся к начальной форме с использованием `ru_stopwords3`.
2. **Получение эмбеддинга запроса.** Для каждого слова запроса, если оно содержится в выбранной модели, извлекается соответствующий эмбеддинг. Вектор запроса формируется как среднее арифметическое всех найденных векторов слов.
3. **Поиск ближайших документов.** С помощью алгоритма ближайших соседей (k-NN), реализованного в `faiss`, осуществляется поиск наиболее релевантных документов на основе евклидовой метрики. По умолчанию возвращаются 5 наиболее близких документов.
4. **Вывод результатов.** По найденным идентификаторам отображаются заголовки соответствующих новостей.

С программным кодом, реализующим алгоритм семантического поиска можно ознакомиться в приложении Ж.

По данному алгоритму рассмотрим пример. В качестве запроса возьмём заголовок одной из новостей: *В.В. Володин поздравил СГУ с праздником Победы».*

Результаты поэтапно:

1. После предобработки запроса получим список токенов:
[‘володин’, ‘поздравить’, ‘сгу’, ‘праздник’, ‘победа’]
2. Преобразуем каждый токен в эмбеддинг через модель. Найдём среднее арифметические полученных эмбеддингов. В результате получим следующий эмбеддинг:
[[0.1731391 0.4556614 0.21552482 0.66046524, ...]]
3. Найдём наиболее релевантные документы по полученному эмбеддингу. В результате получим идентификаторы новостей:
[22160, 2, 29536, 29551, 24466]
4. Преобразуем идентификаторы новостей в их заголовки:
 - СГУ приглашает саратовцев в парк Победы,
 - В.В. Володин поздравил СГУ с праздником Победы,
 - День Победы в СГУ,
 - Концерт для ветеранов,
 - 7 мая в СГУ пройдёт митинг ко Дню победы.

Далее идут примеры работы системы.

Пример 1:

- 1 Запрос: май состояться общенародный акция бессмертный полк
- 2
- 3 First title should be: Студенты и сотрудники СГУ присоединились к шествию
↪ «Бессмертного полка»
- 4
- 5 Студенты и сотрудники СГУ присоединились к шествию «Бессмертного полка»
- 6 Балашовский институт принял участие в торжественных мероприятиях ко Дню Победы
- 7 Студенты СГУ приняли участие в акции, приуроченной ко Дню памяти и скорби
- 8 Делегация Саратовского университета прошла в строю «Бессмертного полка»
- 9 Студенты и сотрудники СГУ стали участниками мероприятий ко Дню Победы

Видим, что самой близкой новостью является та, откуда был взят текст для запроса. Кроме того, заметим, что и другие новости довольно близки по значению к запросу.

Пример 2:

- 1 Запрос: Вклад ученых
- 2
- 3 Ученые СГУ получили президентские гранты

- 4 В.С. Анищенко присвоено звание «Основатель научной школы»
- 5 Продолжается приём заявок на соискание премии Президента в области науки и
→ инноваций
- 6 Учёный СГУ отмечен медалью ордена «За заслуги перед Отечеством» II степени
- 7 Фонд «БАЗИС» открыл приём заявок на конкурсы индивидуальных исследовательских
→ грантов

Пример 3:

- 1 Запрос: Бессмертный полк
- 2
- 3 В СГУ открыт студенческий штаб акции «Бессмертный полк»
- 4 Сегодня в ИФиЖ состоится «Праздник белых журавлей»
- 5 Студенты и сотрудники СГУ стали участниками мероприятий ко Дню Победы
- 6 Сотрудники и студенты могут увековечить память защитников блокадного
→ Ленинграда
- 7 Министерство молодёжной политики предлагает сделать 70 добрых дел

Пример 4:

- 1 Запрос: Чемпионат мира по программированию
- 2
- 3 СГУ участвует в проведении финала чемпионата ACM-ICPC 2013
- 4 Студент ИФКиС стал чемпионом России по кикбоксингу
- 5 Программисты СГУ отправились на финал Чемпионата мира
- 6 Профессор В.Н. Чинилов стал призёром Чемпионата России
- 7 В Саратове пройдёт Неделя информатики

Можно заметить на последнем примере, что в ответе присутствуют новости, которые имеют опосредованное отношение к запросу (новость «Студент ИФКиС стал чемпионом России по кикбоксингу» относится к запросу «Чемпионат мира по программированию» лишь относительно слова *чемпион*). Это может происходить из-за трех причин:

1. Простой способ получения эмбеддинга документа;
2. Поиск k ближайших документов алгоритмом осуществляется алгоритмом KNN (k-nearest neighbor);
3. Эмбеддинги, получаемые из запроса, не учитывают контекст

Однако, в среднем ответы на запрос действительно близки по смыслу к запросу, следовательно модель выдает релевантные документы.

5 Анализ работы семантического поиска на основе разных моделей

Для качественного анализа была сгенерирована тестовая выборка, состоящая из пар: запрос и идентификатор новости. Считается, что модель справилась с задачей, если релевантная новость входит в первые k результатов.

Запросы создавались с помощью модели `microsoft/Phi-4-mini-instruct`, способной понимать русский язык. На вход ей подавался системный промпт и текст новости. Промпт ориентировал модель генерировать короткие, реалистичные и разнообразные запросы, имитируя поведение реальных пользователей. Промпт имеет вид:

- 1 Тебе дают текст новости.
- 2 Сгенерируй 5 реалистичных поисковых запросов, которые могли бы ввести
 - ↪ пользователи в поисковике, чтобы найти эту новость.
- 3 Учитывай, что пользователи часто не знают точных формулировок, могут
 - ↪ использовать ключевые слова, имена, места, приблизительные описания и
 - ↪ синонимы.
- 4 Запросы должны быть короткими (3-7 слов). Формулируй запросы разнообразно и
 - ↪ естественно, как это делают реальные пользователи.

Пример получаемого ответа:

- 1 Саратовский конкурс «Сияние талантов» 2015,
- 2 Юлиана Мелкумян, Саратовский университет, победитель вокала, Конкурс детского
 - ↪ творчества в Саратове, Гран-при,
- 3 Саратовский конкурс, вокалистка, поездка в Пекин,
- 4 Конкурс «Сияние талантов», Саратов, награда в Пекине.

Помимо запросов, сгенерированных с помощью LLM, были добавлены запросы, являющиеся заголовками новости. Таким образом, сформировалась выборка из 3343 запросов.

Результат работы семантического поиска на основе эмбеддингов моделей Word2Vec и Glove продемонстрированы на рис. 5.1. Модель GloVe показывает более высокую точность: нужная новость оказывается на первом месте в 34% запросов и входит в топ-5 в 38% случаев. Для сравнения, модель Word2Vec выдаёт релевантную новость первой в 31% запросов, а в топ-5 — в 35%.

Сравнение моделей Word2Vec и GloVe показало, что GloVe справляется лучше. Это объясняется тем, что GloVe обучается на глобальной статистике, что даёт ему преимущество в понимании семантики коротких запросов.

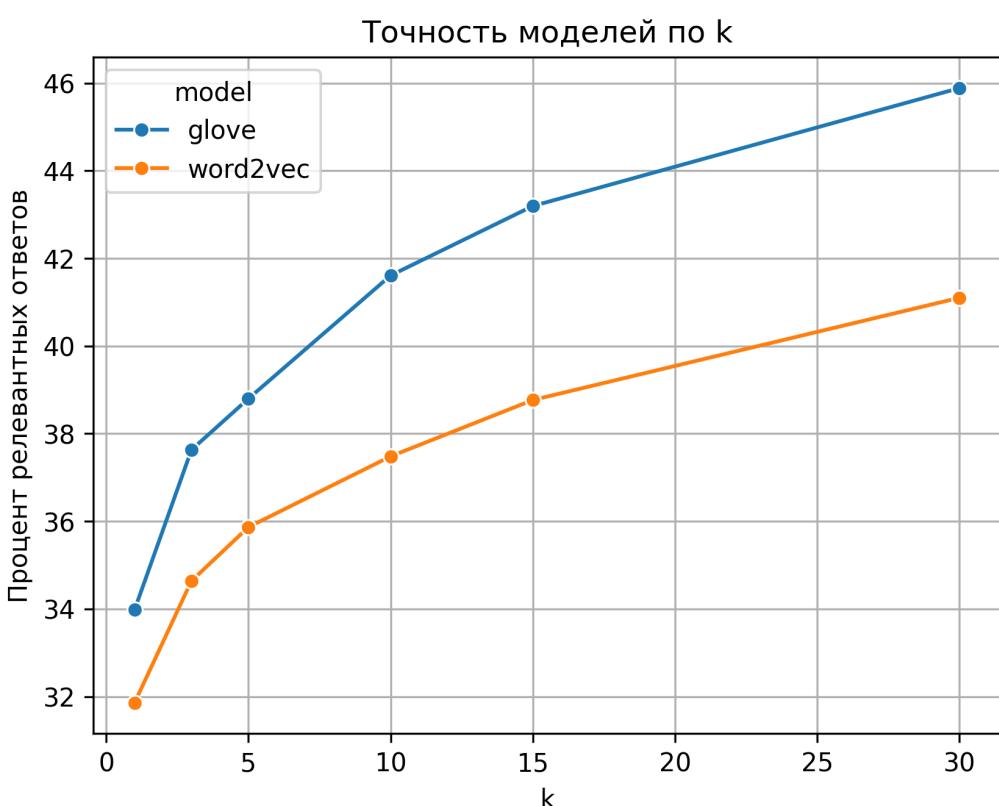


Рисунок 5.1 – Точность моделей Word2Vec и GloVe в задаче семантического поиска на основе корпуса новостей

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была реализована система семантического поиска по корпусу новостных документов с использованием векторных представлений слов и текстов. Основной целью исследования стало сопоставление качества поиска на основе двух популярных моделей эмбеддингов — Word2Vec и GloVe.

На первом этапе была проведена теоретическая проработка алгоритмов построения эмбеддингов, а также изучено понятие информационно-поискового тезауруса. Далее были построены модели Word2Vec и GloVe на специально собранном корпусе новостных текстов. Затем реализован алгоритм семантического поиска, использующий векторные представления как для пользовательских запросов, так и для документов.

Особое внимание было уделено созданию тестовой выборки, включающей как реальные заголовки новостей, так и запросы, сгенерированные с помощью крупной языковой модели (LLM). Это позволило сформировать разнообразную и реалистичную среду для оценки эффективности семантического поиска.

Анализ результатов показал, что модель GloVe превосходит Word2Vec по точности выдачи релевантных результатов. В частности, GloVe возвращает нужную новость на первом месте в 34% случаев, а в топ-5 — в 38%, тогда как Word2Vec — соответственно в 31% и 35%. Кроме того, модель Word2Vec склонна выдавать больше нерелевантных и «шумных» ближайших соседей, особенно при работе с абстрактными понятиями.

Таким образом, можно сделать вывод, что использование эмбеддингов, основанных на глобальной статистике, таких как GloVe, более предпочтительно для задач семантического поиска по новостным документам. Такая модель лучше улавливает смысловые связи между короткими пользовательскими запросами и содержанием новостей, что особенно важно в условиях информационного шума и вариативности формулировок.

ПРИЛОЖЕНИЕ А

Программный код, выполняющий обработку текста

```
1 import pandas as pd
2 import pymorphy3
3 import re
4 import dask.dataframe as dd
5
6 def string_dev(a_in):
7     x_in = re.sub(r'\n', ' ', a_in)
8     y_in = x_in.lower()
9     b_in = re.sub(r'&lt;w+;', ' ', y_in)
10    d_in = re.sub(r'<[^>]*>', ' ', b_in)
11    f_in = re.sub(r'www|.w+.w{2,3}?', ' ', d_in)
12    a_in = re.sub(r'\xad', ' ', f_in)
13    c_in = re.sub(r'\\xa0-', ' ', a_in)
14    u_in = re.sub(r'\\u200e', ' ', c_in)
15    w_in = re.sub(r'\d+', ' ', u_in)
16    y_in = re.sub(r'\- ', ' ', w_in)
17    yy_in = re.sub(r'_+', ' ', y_in)
18    yyy_in =
19        ↳ re.sub(r'[...-----/()\\[\\]\\\\\\,\\-:\\;<>@#%\\|^|+|*|^&^~|\\$|^<>_i-]+', ,
20        ↳ ' ', yy_in)
21    q_in = re.sub(r'[a-z]*', ' ', yyy_in)
22    sss_in = re.sub(r'\b\w{,2}\b', ' ', q_in)
23    qms_in = re.sub(r'\s{2,}', ' ', sss_in)
24    nms_in = qms_in.strip()
25    return nms_in
26
27 def lematization(f_input_list):
28     morph = pymorphy3.MorphAnalyzer()
29     lnorm = []
30     for word in f_input_list:
31         p = morph.parse(word)[0]
32         lnorm.append(p.normal_form)
33     return lnorm
34
35 def del_my_stop_words(word_tokens):
36     stop_words_nltk = {'который', 'кому', 'имя', 'сегодня', 'вчера',
37         ↳ 'завтра', 'также', 'в', 'во', 'свой',
38             ↳ 'это', 'часто', 'зачастую', 'можь', 'сможь', 'а',
39                 ↳ 'без', 'более', 'больше', 'будет',
```

```

36      'будто', 'бы', 'был', 'была', 'были', 'было',
        ↳ 'быть', 'в', 'вам', 'vas', 'вдруг',
37      'ведь', 'во', 'вот', 'впрочем', 'есе', 'всегда',
        ↳ 'всего', 'всех', 'всю', 'вы', 'где',
38      'да', 'даже', 'два', 'для', 'до', 'другой', 'его',
        ↳ 'ее', 'ей', 'ему', 'если', 'есть',
39      'еще', 'же', 'же', 'за', 'зачем', 'здесь', 'и', 'из',
        ↳ 'или', 'им', 'иногда', 'их', 'к',
40      'как', 'какая', 'какой', 'когда', 'конечно', 'кто',
        ↳ 'куда', 'ли', 'лучше', 'между',
41      'меня', 'мне', 'много', 'может', 'можно', 'мой',
        ↳ 'моя', 'мы', 'на', 'над', 'надо',
42      'наконец', 'нас', 'не', 'него', 'нее', 'ней',
        ↳ 'нельзя', 'нет', 'ни', 'нибудь', 'никогда',
43      'ним', 'них', 'ничего', 'но', 'ну', 'о', 'об',
        ↳ 'один', 'он', 'она', 'они', 'опять', 'от',
44      'перед', 'по', 'под', 'после', 'потом', 'потому',
        ↳ 'почти', 'при', 'про', 'раз', 'разве',
45      'с', 'со', 'сам', 'свою', 'себе', 'себя', 'сейчас',
        ↳ 'с', 'со', 'совсем', 'так', 'такой',
46      'там', 'тебя', 'тем', 'теперь', 'то', 'тогда',
        ↳ 'того', 'тоже', 'только', 'том', 'том',
47      'три', 'тут', 'ты', 'у', 'уж', 'уже', 'хорошо',
        ↳ 'хоть', 'что', 'чего', 'чем', 'через',
48      'что', 'чтоб', 'чтобы', 'чуть', 'эти', 'этого',
        ↳ 'этой', 'этом', 'этом', 'эту', 'я',
49      'сказал', 'человек', 'жизнь', 'говорил', 'кажется',
        ↳ 'сказать', 'сегодня', 'сказала',
50      'сказал'}
51
52 my_stop_words = {'сгт', 'свой', 'стать', 'кроме', 'разный', 'около',
        ↳ 'затем', 'помимо', 'ваш', 'вам',
        'некоторый', 'лишь', 'каждый', 'самый', 'также',
        ↳ 'неоднократно', 'ещё', 'сразу', 'среди',
53      'однако', 'вновь', 'иной', 'ныне', 'пока', 'хотя',
        ↳ 'либо', 'немного', 'гораздо', 'ничто',
54      'нередко', 'наоборот', 'впереди', 'таковой', 'мимо',
        ↳ 'тесно', 'вряд', 'нечто', 'почём',
55      'почему', 'любой', 'обратно', 'оттуда', 'очень',
        ↳ 'понапрасну', 'поскольку', 'поэтому',
56      'прежде', 'причём', 'прочий', 'пусть', 'наш',
        ↳ 'несколько', 'никак', 'твой', 'подробный',

```

```

58     'информация' }
59
60     stop_words = stop_words_nltk | my_stop_words
61     filtered_sentence = [w for w in word_tokens if w not in stop_words]
62     return filtered_sentence
63
64 from natasha import Segmenter, MorphVocab, NewsEmbedding, NewsMorphTagger,
65     → NewsNERTagger, Doc
66
67 segmenter = Segmenter()
68 morph_vocab = MorphVocab()
69 emb = NewsEmbedding()
70 morph_tagger = NewsMorphTagger(emb)
71 ner_tagger = NewsNERTagger(emb)
72
73 def remove_proper_nouns(text):
74     if not isinstance(text, str) or not text.strip():
75         print("text имеет неожидаемый тип")
76         return "error type"
77     try:
78         doc = Doc(text)
79         doc.segment(segmenter)
80         doc.tag_morph(morph_tagger)
81         doc.tag_ner(ner_tagger)
82         if not doc.spans:
83             return text
84         spans_to_remove = [span for span in doc.spans if span.type in ['PER',
85             → 'LOC', 'ORG']]
86         text_cleaned = text
87         for span in sorted(spans_to_remove, key=lambda x: x.start,
88             → reverse=True):
89             text_cleaned = text_cleaned[:span.start] +
90             → text_cleaned[span.stop:]
91             print(text_cleaned)
92             return text_cleaned.strip()
93     except Exception as e:
94         print(f"Ошибка при обработке текста: {text[:50]}... Ошибка:
95             → {str(e)}")
96         return "error"
97
98 def process_str_of_the_news(string):

```

```

94     string_2 = string_dev(string)
95     if len(string_2) != 0:
96         list_of_sentences = re.split(r '[.!?]', string_2)
97         list_of_sentences = [x.split(' ') for x in list_of_sentences if x]
98         filtered_sentences = []
99         for sentence in list_of_sentences:
100             no_stopwords_1 = del_my_stop_words(sentence)
101             lemmatized = lematization(no_stopwords_1)
102             text = ' '.join(lemmatized)
103             text = re.sub(r '\b\w{1,2}\b', ' ', text)
104             text = re.sub(r '\s{2,}', ' ', text)
105             text = text.strip()
106             if text:
107                 filtered_sentences.append(text)
108
109
110 import pyarrow as pa
111 def apply_tokenization():
112     docs = dd.read_parquet("raw-data.pq").repartition(npartitions=8).loc[:1]
113     print("Read finished")
114     docs['News_Tokens'] = docs['News_Text'].map_partitions(
115         lambda s: s.apply(remove_proper_nouns),
116         meta=( "News_Tokens" , object)
117     )
118     docs['News_Tokens'] = docs["News_Tokens"].map_partitions(
119         lambda s: s.apply(process_str_of_the_news),
120         meta=( "News_Tokens" , object)
121     )
122     print("Returning")
123     schema = {
124         "News_Text": pa.string(),
125         "News_Tokens": pa.list_(pa.string()),
126         "News_Title": pa.string()
127     }
128     return docs.to_parquet("output.pq", schema=schema,
129                           write_metadata_file=True)
130
131 apply_tokenization()

```

ПРИЛОЖЕНИЕ Б

Построение тезауруса по модели эмбеддинга

```
1 import pandas as pd
2 import dask.dataframe as dd
3 import dask.bag as db
4 from gensim.models import Word2Vec
5 docs = dd.read_parquet("../output.pq/")
6 model = Word2Vec.load("word2vec_sent_5_sgu.model")
7 texts = docs['News_Tokens'].compute()
8 bag = db.from_sequence(texts)
9 sentences = bag.flatten()
10 words = sentences.map(lambda x: x.split()).flatten().distinct().compute()
11 results = []
12 threshold = 0.8
13 for word in words:
14     try:
15         # Получаем 10 ближайших слов
16         similar_words = model.wv.most_similar(word, topn=10)
17         filtered_words = [(w, sim) for w, sim in similar_words if sim >=
18             threshold]
19         results.append([word, filtered_words])
20     except KeyError:
21         # Если слово не в модели, пропускаем его
22         print(f"Слово '{word}' не найдено в модели.")
23 df_results = pd.DataFrame(results, columns=["Word", "Most_Similar_Word"])
24 df_results.to_csv("similar_words_sent_5.csv", index=False)
```

ПРИЛОЖЕНИЕ В

Получение случайных 20 слов

```
1 import pandas as pd
2 import os
3
4 # Пути к файлам и соответствующие названия колонок
5 file_paths = {
6     # 'bert': '/bert/sim-words5-bert.csv',
7     'glove': '/glove_python/sim-words_sent_5_glove.csv',
8     'word2vec': '/word2vec/similar_words_sent_5.csv'
9 }
10
11 # Создаем пустой DataFrame с колонкой 'word'
12 data = pd.DataFrame(columns=['word'])
13
14 # Обрабатываем каждый файл
15 for model_name, path in file_paths.items():
16     # Читаем CSV-файл
17     full_path = ".." + path
18     df = pd.read_csv(full_path)
19
20     # Переименовываем колонку Most_Similar_Word в название модели
21     df = df.rename(columns={'Most_Similar_Word': model_name})
22
23     # Если это первый файл, используем его слова как основу
24     if data.empty:
25         data['word'] = df['Word']
26
27     # Добавляем данные в основной DataFrame
28     data[model_name] = df[model_name]
29 import ast
30 # Функция для проверки, является ли значение пустым списком или NaN
31 def is_valid_similar_words(value):
32     if pd.isna(value): # проверяем NaN
33         return False
34     if isinstance(value, str): # если данные хранятся как строки (например,
35     ↪ "[('слово', 0.5), ...]")
36         try:
37             lst = ast.literal_eval(value) # преобразуем строку в список
38             return len(lst) > 0 # True, если список не пустой
39         except (ValueError, SyntaxError):
```

```

39         return False
40     elif isinstance(value, list): # если данные уже в формате списка
41         return len(value) > 0
42     else:
43         return False # на случай других форматов
44
45 # Применяем фильтрацию: оставляем строки, где ВСЕ столбцы с похожими словами
46 #→ не пусты
46 filtered_data = data[
47     data[ 'glove' ].apply(is_valid_similar_words) &
48     data[ 'word2vec' ].apply(is_valid_similar_words)
49 ]
50 n = 25 # кол-во случайных строк
51
52 random_sample = filtered_data.sample(n=n, random_state=42) # random_state для
53 #→ воспроизводимости
54
54 # Сохраняем в новый CSV-файл
55 output_file = 'random_sample_similar_words.csv'
56 random_sample.to_csv(output_file, index=False, encoding='utf-8')

```

ПРИЛОЖЕНИЕ Г

Построение графа

```
1 import pandas as pd
2 import os
3
4 # Пути к файлам и соответствующие названия колонок
5 file_paths = {
6     # 'bert': '/bert/sim-words5-bert.csv',
7     'glove': '/glove_python/sim-words_sent_5_glove.csv',
8     'word2vec': '/word2vec/similar_words_sent_5.csv'
9 }
10
11
12 data = {}
13 for model_name, path in file_paths.items():
14     full_path = ".." + path
15     df = pd.read_csv(full_path)
16     data[model_name] = df
17 import networkx as nx
18 from ast import literal_eval
19
20 graphs = []
21 for model_name, df in data.items():
22     G = nx.Graph() # или nx.DiGraph() для направленного графа
23
24     for _, row in df.iterrows():
25         word = row['Word']
26         similar_words = row['Most_Similar_Word']
27
28         # Пропускаем пустые списки
29         if not similar_words or pd.isna(similar_words):
30             continue
31
32         # Преобразуем строку в список кортежей (если данные в формате строки)
33         if isinstance(similar_words, str):
34             try:
35                 similar_words = literal_eval(similar_words)
36             except (ValueError, SyntaxError):
37                 continue
38
39         # Добавляем рёбра в граф
```

```

40         for similar_word, weight in similar_words:
41             G.add_edge(word, similar_word, weight=weight)
42 graphs[model_name] = G
43 # nx.write_gexf(G, f"{model_name}_graph.gexf") # формат GEXF для Gephi
44 print(f "Граф '{model_name}' содержит {len(G.nodes)} узлов и
45       {len(G.edges)} ребер.")
46 from community import community_louvain
47 for model_name, G in graphs.items():
48     print(f "model name: {model_name}")
49     partition = community_louvain.best_partition(G, weight='weight')
50     # Кол-во кластеров
51     num_clusters = max(partition.values()) + 1
52     print(f "Число кластеров (Louvain): {num_clusters}")
53
54     # Размеры кластеров
55     from collections import Counter
56     cluster_sizes = Counter(partition.values())
57     print(f "Размеры кластеров: {cluster_sizes.most_common(5)}" # Top-5
58           кластеров
59
60     # 3. Модулярность (качество кластеризации)
61     modularity = community_louvain.modularity(partition, G, weight='weight')
62     print(f "Модулярность: {modularity:.3f}")

```

ПРИЛОЖЕНИЕ Д

Расширение модели, построенной на корпусе новостных документов

```
1 from gensim.models import KeyedVectors
2 big =
3     ↳ KeyedVectors.load_word2vec_format("ruscorpora_upos_skipgram_300_5_2018/model.kv",
4     ↳ binary=False)
5 small = KeyedVectors.load("word2vec_sent_5_sgu.vec") // and
6     ↳ glove_sent_5_sgu.model
7
8 small_words = set(small.wv.key_to_index.keys())
9 big_words = set(big.key_to_index.keys())
10
11 all_words = small_words.union(big_words)
12
13 new_vocab = []
14 new_vectors = []
15
16 for word in all_words:
17     if word in small_words and word in big_words:
18         vec = (small.wv[word] + big.get_vector(word)) / 2
19     elif word in small_words:
20         vec = small.wv[word]
21     else:
22         vec = big.get_vector(word)
23
24     new_vocab[word] = len(new_vectors)
25     new_vectors.append(vec)
26
27 new_kv = KeyedVectors(vector_size=big.vector_size)
28 new_kv.add_vectors(list(new_vocab.keys()), new_vectors)
29
30 new_kv.save("word2vec.kv")
```

ПРИЛОЖЕНИЕ Е

Получение и сохранение эмбеддингов документов

```
1 import pandas as pd
2 from gensim.models import KeyedVectors
3 import dask.dataframe as dd
4 import dask.bag as db
5 import numpy as np
6
7 # Пути к файлам
8 file_paths = {
9     'glove': '../glove_python/glove.kv',
10    'word2vec': '../word2vec/word2vec.kv'
11 }
12 word2vec = KeyedVectors.load(file_paths['word2vec'], mmap='r')
13 glove = KeyedVectors.load(file_paths['glove'], mmap='r')
14 docs = dd.read_parquet("../output.pq/")
15 texts = docs['News_Tokens'].compute()
16 bag = db.from_sequence(texts)
17 list_news = bag.map(lambda sent: ' '.join(sent)).map(lambda news:
18     news.split()).compute()
19 ids = docs['News_Id'].compute()
20 def get_doc_embedding(tokens, model):
21     vectors = [model[word] for word in tokens if word in model]
22     if vectors:
23         return np.mean(vectors, axis=0)
24     else:
25         return np.zeros(model.vector_size)
26 glove_embeddings = {}
27 word2vec_embeddings = {}
28 for (news, id) in zip(list_news, ids):
29     word2vec_embeddings[id] = get_doc_embedding(news, word2vec)
30     glove_embeddings[id] = get_doc_embedding(news, glove)
31 import numpy as np
32 import json
33 import faiss
34 folder = 'news-embeddings'
35 # Словари + списки + массивы и id-шники
36 def save_embeddings_dict(emb_dict, prefix):
37     ids = list(emb_dict.keys())
38     vectors = np.array([emb_dict[i] for i in ids], dtype='float32')
```

```
39     index = faiss.IndexFlatL2(vectors.shape[1])
40     index.add(vectors)
41     np.save(f "{folder}/{prefix}_vectors.npy", vectors)
42     faiss.write_index(index, f "{folder}/{prefix}.index")
43
44     with open(f "{folder}/{prefix}_ids.json", "w", encoding="utf-8") as f:
45         json.dump(ids, f)
46
47 save_embeddings_dict(glove_embeddings, "glove")
48 save_embeddings_dict(word2vec_embeddings, "word2vec")
```

ПРИЛОЖЕНИЕ Ж

Реализация алгоритма семантического поиска

```
1 import numpy as np
2 import json
3 import faiss
4
5 def load_index_and_ids(prefix: str, folder: str = "news-embeddings"):
6     # Загрузка FAISS индекса
7     index = faiss.read_index(f"{folder}/{prefix}.index")
8
9     # Загрузка соответствующих ID
10    with open(f"{folder}/{prefix}_ids.json", "r", encoding="utf-8") as f:
11        ids = json.load(f)
12
13    return index, ids
14 db = {
15     "glove": load_index_and_ids("glove"),
16     "word2vec": load_index_and_ids("word2vec")
17 }
18 from gensim.models import KeyedVectors
19 models = {
20     "glove": KeyedVectors.load("../glove_python/glove.kv"),
21     "word2vec": KeyedVectors.load("../word2vec/word2vec.kv")
22 }
23 # Считывание новостей
24 import dask.dataframe as dd
25 docs = dd.read_parquet("../output.pq/", columns=['News_Id', 'News_Title',
26     ↳ 'News_Tokens'])
27 df = docs.compute()
28 id_to_title = dict(zip(df['News_Id'], df['News_Title']))
29 import pymorphy3
30 import re
31 def lematization(f_input_list):          # Лематизация слов в списке
32     morph = pymorphy3.MorphAnalyzer()
33     lnorm = list()
34     for word in f_input_list:
35         p = morph.parse(word)[0]
36         lnorm.append(p.normal_form)
37     return (lnorm)
38 def preprocess_query(query):
39     query = query.lower()
```

```

39     query = re.sub(r "[^w\s]", " ", query)
40     query = re.sub(r '\b\w{,2}\b', '', query)
41     query = re.sub(r '\s{2,}', ' ', query)
42     tokens = query.split()
43     return lematization(tokens)
44 def get_query_embedding(query: str, model_name: str):
45     model: KeyedVectors = models[model_name]
46     tokens = preprocess_query(query=query)
47     # print(f"tokens = {tokens}")
48     vectors = [model[word] for word in tokens if word in model]
49     if not vectors:
50         print(f"Query vector {query} is empty")
51         return np.zeros((1, model.vector_size), dtype='float32')
52     return np.mean(vectors, axis=0).astype('float32').reshape(1, -1)
53 def semantic_search(query: str, model_name: str, k = 5):
54     query_vec = get_query_embedding(query, model_name)
55     # print(f"query_vec = {query_vec}")
56     if np.linalg.norm(query_vec) == 0:
57         return []
58     index, ids = db[model_name]
59     D, I = index.search(query_vec, k)
60     news_ids = [ids[i] for i in I[0]]
61     # print(f"Result_index = {news_ids}")
62     return news_ids
63 text = "... # input there"
64 answers = semantic_search(query=text, model_name="word2vec")
65 answers = [id_to_title[i] for i in answers]
66 for answer in answers:
67     print(answer)

```