Реферат

Пояснительная записка выпускной квалификационной работы магистра содержит 93 стр., 4 табл., 19 рис., 67 источников, 2 приложения.

Целью работы является разработка модификации метода сравнения коротких документов, проектирование и реализация программной системы для демонстрации методики.

Объект исследования – определение подобия коротких текстовых описаний фильмов.

Предмет исследования – алгоритмы и методы определения подобия документов.

Проанализированы существующие подходы к обработке текстовой информации и выдаче персональных рекомендаций, рассмотрены основные методы поиска подобия текстовых документов, выделены их достоинства и недостатки. Проанализированы особенности работы с документами в виде коротких описаний фильмов и предложена модификация популярного алгоритма, нацеленная на усовершенствование определения подобия коротких документов. Реализована многопользовательская программная система для демонстрации этого алгоритма на основе списка рекомендованных фильмов с возможностью гибкой конфигурации настроек анализа документов.

Научная новизна заключается в разработке программной модели, реализующей алгоритм определения подобия коротких текстовых объектов, основывающийся на предварительно обработанной коллекции документов.

ФИЛЬМ, РЕКОМЕНДАЦИИ, ДОКУМЕНТ, КОНТЕКСТ, ВЕС, АЛГОРИТМ, МЕРА СХОДСТВА, КОЭФФИЦИЕНТ, ЧАСТОТА, МЕТОД

СОДЕРЖАНИЕ

[Введение 8](#_Toc42969243)

[1 Анализ методов выдачи рекомендаций 10](#_Toc42969244)

[1.1 Контентная фильтрация 12](#_Toc42969245)

[1.2 Коллаборативная фильтрация 13](#_Toc42969246)

[1.3 Гибридные рекомендательные системы 15](#_Toc42969247)

[1.4 Проблемы рекомендательных систем 15](#_Toc42969248)

[1.4.1 Проблема холодного старта 16](#_Toc42969249)

[1.4.2 Проблема разреженных данных 16](#_Toc42969250)

[1.4.3 Проблема одинаковых данных 17](#_Toc42969251)

[1.4.4 Проблема безопасности личных данных 17](#_Toc42969252)

[1.5 Обзор существующих систем 18](#_Toc42969253)

[1.6 Выводы по разделу 18](#_Toc42969254)

[2 Обработка текстов на естественном языке 20](#_Toc42969255)

[2.1 Строковые метрики 23](#_Toc42969256)

[2.2 Методы поиска подобия документов на основе их содержания 26](#_Toc42969257)

[2.2.1 Word2Vec 26](#_Toc42969258)

[2.2.2 LSA 28](#_Toc42969259)

[2.2.3 ESA 28](#_Toc42969260)

[2.3 Топологические методы, основанные на использовании семантических сетей 29](#_Toc42969261)

[2.4 Выводы по разделу 30](#_Toc42969262)

[3 Обзор метода латентно-семантического подобия 31](#_Toc42969263)

[3.1 Анализ алгоритма LSA 31](#_Toc42969264)

[3.1.1 Предобработка текста 33](#_Toc42969265)

[3.1.2 Векторное представление текста 34](#_Toc42969266)

[3.1.3 Составление матрицы слово-документ 35](#_Toc42969267)

[3.1.4 Сингулярное разложение матриц (SVD) 38](#_Toc42969268)

[3.1.5 Определение подобия документов 40](#_Toc42969269)

[3.2 Разработка модификации алгоритма LSA 41](#_Toc42969270)

[3.3 Выводы по разделу 42](#_Toc42969271)

[4 Проектирование рекомендательной системы AdviseMe 44](#_Toc42969272)

[4.1 Анализ требований к системе 44](#_Toc42969273)

[4.2 Проектирование базы данных 47](#_Toc42969274)

[4.3 Проектирование приложения 48](#_Toc42969275)

[4.4 Проектирование процесса выдачи рекомендаций фильмов 51](#_Toc42969276)

[4.5 Выводы по разделу 57](#_Toc42969277)

[5 Оценка эффективности рекомендательной системы AdviseMe 58](#_Toc42969278)

[5.1 Анализ мер для оценивания рекомендательных систем 58](#_Toc42969279)

[5.2 Мера F1 58](#_Toc42969280)

[5.3 Исследование эффективности разрабатываемой рекомендательной системы 60](#_Toc42969281)

[5.4 Выводы по разделу 63](#_Toc42969282)

[6 Охрана труда 64](#_Toc42969283)

[6.1 Анализ характеристик и потенциальных опасностей и вредностей объекта исследования 64](#_Toc42969284)

[6.2 Выбор и обоснование мероприятий для создания нормальных и безопасных условий труда 67](#_Toc42969285)

[6.2.1 Расчёт необходимой площади окон для бокового естественного освещения 70](#_Toc42969286)

[6.3 Обеспечение пожарной безопасности 73](#_Toc42969287)

[Заключение 74](#_Toc42969288)

[Список использованных источников 75](#_Toc42969289)

[Приложение А Программный код 82](#_Toc42969290)

[Приложение Б Графическая часть 88](#_Toc42969291)

Введение

**Актуальность темы.** Совершенствование методов обработки ествественноязыковых текстов является актуальной проблемой в условиях расширения набора задач, выполняемых с помощью автоматического извлечения информации. Алгоритмы обработки текстов естественного языка применяют для усовершенствования систем обучения, редактирования и аннотирования текстов, перевода с иностранных языков, распознавания речи, систем рекомендаций, автоматизации колл-центров и ряда других задач. Следовательно, с увеличением спроса на автоматизацию большого количества информации повышается актуальность исследования методов повышения эффективности алгоритмов.

**Цель работы**. Целью работы является разработка модифицированного метода определения подобия коротких текстов для составления списка рекомендуемых фильмов, проектирование и реализация программной системы для демонстрации метода.

**Задачи исследования:**

* анализ существующих подходов к обработке текстовой информации;
* анализ метрик для определения подобия документов;
* анализ подходов к реализации рекомендательных систем;
* разработка собственной метрики для определения подобия фильмов по их краткому текстовому описанию;
* проектирование и реализация программной системы, демонстрирующей разработанный метод;
* оценка эффективности полученного алгоритма.

**Объект исследования.** Определение подобия коротких текстовых описаний фильмов.

**Предмет исследования.** Алгоритмы и методы определения близости документов.

**Методы исследований.** Анализ существующих подходов к обработке текстовой инофрмации, моделям ее представления, основным методам определения подобия текстовых документов.

**Научная новизна.** Новизна заключается в разработке рекомендательной системы, реализующей алгоритм определения подобия фильмов по их описаниям.

**Практическое значение полученных результатов.** Разработанные алгоритмы и модули могут быть основой рекомендательной системы фильмов.

**Аппробация результатов исследования.** Основные положения и результаты докладывались и обсуждались, получили положительную оценку на следующих конференциях и семинарах:

Информатика, управляющие системы, математическое и компьютерное моделирование (ИУСМКМ-2018) / Материалы студенческой секции IX Международной научно-технической конференции «Информатика, управляющие системы, математическое и компьютерное моделирование» (ИУСМКМ-2018). – Донецк: ДонНТУ, 2018.

Современные информационные технологии в образовании и научных исследованиях (СИТОНИ-2019) / Материалы VI Международной научно-технической конференции «Современные информационные технологии в образовании и научных исследованиях» (СИТОНИ-2019). – Донецк: ДонНТУ, 2019.

Информатика, управляющие системы, математическое и компьютерное моделирование (ИУСМКМ-2020) / Материалы студенческой секции X Международной научно-технической конференции «Информатика, управляющие системы, математическое и компьютерное моделирование» (ИУСМКМ-2020). – Донецк: ДонНТУ, 2020.

**Публикации.** По материалам выполненного исследования опубликованы три статьи по материалам конференций [21, 66, 67].

# Анализ методов выдачи рекомендаций

Предпочтения пользователей анализируются различными системами для получения наибольшей выгоды. Некоторые системы могут быть полностью основаны на рекомендациях, например, сервисы по подбору жилья. Однако чаще всего предоставление рекомендаций интегрировано в системы как дополнительное средство привлечения интереса к сервису и увеличению количества активных пользователей, а значит, только частично способствуют получению коммерческой прибыли. Наиболее обширно рекомендации используют интернет-магазины. Большое количество хороших товаров так и останутся невостребованными, если система полагается на неупорядоченный список объектов. А в случае, когда пользователь не может найти подходящий товар, он покидает систему, и сервис лишается потенциального покупателя и возможной прибыли.

Системы выдачи рекомендаций основаны на определении, что понимать под оценкой предпочтения пользователем объекта, с которым тот ещё не взаимодействовал, и используют для этого определённые алгоритмы [[1](https://link.springer.com/content/pdf/10.1007/978-3-319-29659-3.pdf)]. Целью методов «предсказания» заинтересованности пользователя в объекте является выявление скрытых и явных признаков того, что пользователь будет взаимодействовать с объектом и повышать свою удовлетворённость общей системой, а также на определение групп пользователей со схожими интересами, которым с большой вероятностью понравятся одинаковые объекты. В современных условиях информационной перегрузки наибольшую эффективность имеют персональные рекомендации, для составления которых используют всю доступную открытую информацию о пользователе, начиная от информации в профиле, зарегистрированном в системе, и заканчивая потенциальными интересами, которые определяются по данным, полученным из других систем и связанным с интересами других пользователей системы. Персональные рекомендации сейчас наиболее популярны за счёт их эффективности и возможностям получения прибыли, хотя также активно обсуждается и моральная сторона использования личных данных пользователя.

С развитием рекомендательных систем от пользователя требуется давать доступ ко всё более личным данным [[2](http://www.mecs-press.org/ijieeb/ijieeb-v9-n3/IJIEEB-V9-N3-4.pdf)]. Изначально системы использовали только внутренние данные, например, историю покупок, просмотров и т.п, далее системы стали запрашивать данные из социальных сетей пользователей, чтобы находить менее явную информацию о сходстве вкусов пользователей, и сейчас наибольшее распространение получили системы, которые собирают данные о местоположении пользователей, за счёт прироста использования мобильных устройств.

Классификацию рекомендательных систем можно представить как на рисунке Рисунок 1.1 [3].

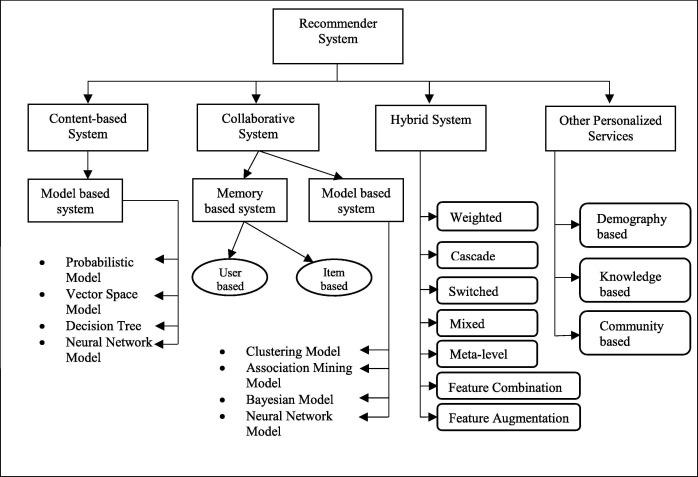


Рисунок 1.1 - Классификация методов выдачи рекомендаций

Данная классификация представляет комбинацию из подходов к составлению списков рекомендаций по типу данных, которые используются при анализе (данные об объектах, о пользователях или данные, полученные из дополнительных внешних источников) и по способу применения подходов (комбинирование их различных частей).

## Контентная фильтрация

Одним из способов создания списка объектов, которые могут быть интересны пользователю, является модель, основанная на использовании информации об объектах и взаимодействии пользователя с ними – контентная фильтрация [66]. Такие системы строятся на двух принципах [[3](https://www.sciencedirect.com/science/article/pii/S1319157819304963)]:

1. выделение атрибутов объектов, которые имеют вес в данной предметной области, включая усреднённый рейтинг пользователей системы;
2. компоновка профиля пользователя, который состоит из его интересов, под которыми понимают список объектов, с которыми он взаимодействовал ранее, т.е. пользовательская история покупок, просмотров и т.п.

Таким образом, система получает всю необходимую информацию, чтобы выдавать в особом списке пользователю новые объекты на основании его прошлых действий. Однако такая система требует дополнительной настройки, если требуется предоставлять рекомендации по более, чем одной предметной области, например, для такого крупного интернет-магазина как Amazon.

При разработке рекомендательной системы использование контентной фильтрации решает проблему холодного старта для объектов, так как каждый объект имеет набор параметров, которые учитываются в алгоритме рекомендаций. Однако для новых пользователей система не сможет составить список персонализированных рекомендаций, так как у таких пользователей ещё нет истории использования. Решить эту проблему можно, предложив пользователю список популярных объектов и отследить его с ними взаимодействие. К недостаткам системы также можно отнести проблему одинаковых данных: рекомендации пользователя ограничены только объектами, которые похожи на объекты из его истории. Решить данную проблему можно с помощью генетических алгоритмов [[4](https://www.atlantis-press.com/journals/ijcis/25885050/view)].

## Коллаборативная фильтрация

В отличие от контентной фильтрации, где необходимо заранее определить наиболее полный и эффективный список характеристик объектов, который этом может отличаться для объектов из разных отраслей, коллаборативная фильтрация не зависит от сферы, в которой её применяют. Общий принцип таких систем основан на составлении таблиц предпочтений пользователей [[5](https://www.researchgate.net/publication/200121027_Collaborative_Filtering_Recommender_Systems)]. Пользователей со схожими интересами можно объединять в группы и затем рассчитывать рекомендации для определённого пользователя в зависимости от его схожести с такими группами. В своём списке рекомендаций пользователь получает те объекты, с которыми он ещё не взаимодействовал, но которые понравились пользователям из какой-либо группы, к которой принадлежит данный пользователь.

Благодаря тому, что происходит привязка к истории не только определённого пользователя, но и к пользователям из групп, к которым он принадлежит, список рекомендаций получается разнообразным и обширным, а также легко изменяется под текущие интересы пользователя, которые могут изменяться со временем.

Способы коллаборативной фильтрации можно распределить в две группы: memory-based и model-based [[7](http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/algorithms.html)]**.**

Memory-based подход использует все имеющиеся данные для поиска схожих объектов или пользователей. Поиск схожих пользователей называют user-item фильтрацией, а поиск схожих объектов – и item-item фильтрацией. User-item – классическое представление коллаборативной фильтрации, поиск объектов, которые понравились пользователям, которые ранее так же оценили те же объекты, что и текущий пользователь. Item-item фильтрация – поиск объектов, которые также понравились пользователям, оценившим объект, который просматривает текущий пользователь. Алгоритмы, используемые для такого подхода, основаны на арифметических операциях, которые определяют расстояние (степень схожести) между объектами. Наиболее известными и эффективными являются косинусное сходство и корреляция Пирсона [[6](https://arxiv.org/abs/1905.07790)].

Model-based подход основан на использовании машинного обучения для предсказания оценки, которую может поставить пользователь. Алгоритмы этого подхода можно разделить на три группы: кластеризация, матричная факторизация и многослойные нейросети. Кластеризация позволяет разделить объекты на группы по выявленным скрытым характеристикам. Считается, что объект принадлежит группе, если он имеет признаки, схожие с другими объектами группы. Наиболее используемые алгоритмы кластеризации – KNN, SOM [[8](https://www.researchgate.net/publication/261355115_Review_based_on_data_clustering_algorithms)]. Матричная факторизация используется для оптимизационных задач, в случае с рекомендательной системой – для выявления скрытых факторов, которые влияют на рекомендации. На примере системы рекомендации фильмов (см. рис. Рисунок 1.2) алгоритм разложения матриц ищет скрытые закономерности для объектов и для пользователей в таблице пользователь-объект и предсказывает, насколько высоко пользователи могут оценить объекты, с которыми ещё не взаимодействовали.

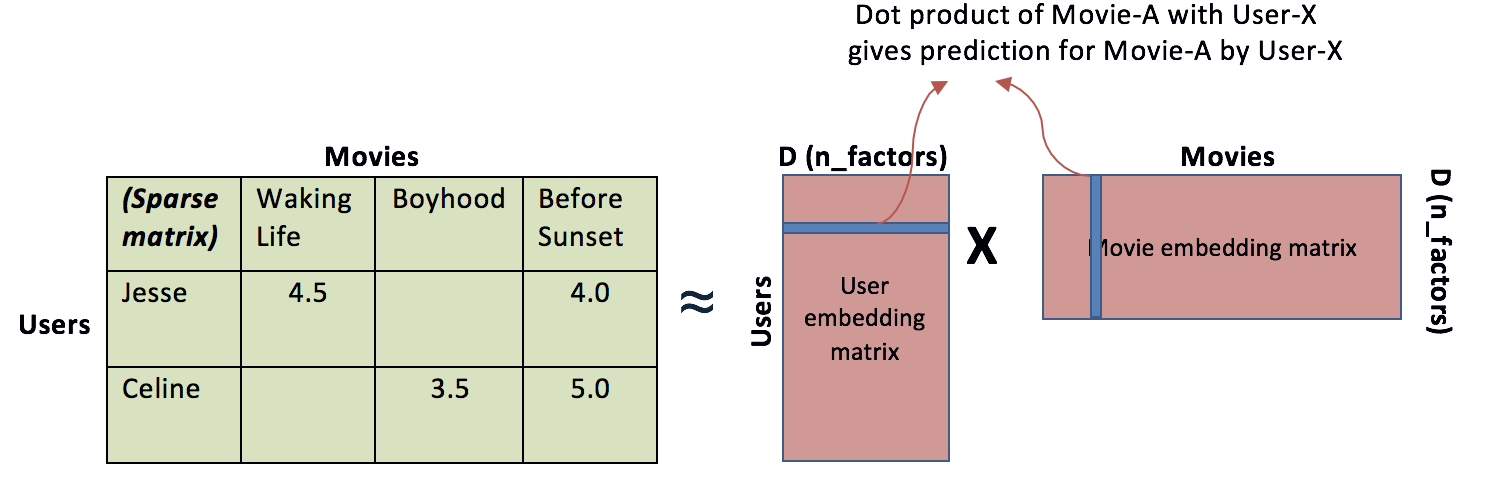


Рисунок 1.2- Процесс выявления возможных оценок к фильмам

Матрица предсказанных оценок строится вследствие скалярного произведения матриц «пользователи-признаки» и «фильмы-признаки», где признаки – это свойства, присущие фильмам, и по которым можно категоризировать пользователей. Например, жанр фильма – предпочитаемый жанр у пользователя, год выпуска фильма – предпочитаемое пользователем время выпуска фильмов и т.п.

ollaborative

ﬁltering (CF) with alternating least squares (ALS) algorithm is the most imperative

techniques which are used for building a movie recommendation engine.

Collaborative

ﬁltering (CF) with alternating least squares (ALS) algorithm is the most imperative

techniques which are used for building a movie recommendation engine.

Collaborative

ﬁltering (CF) with alternating least squares (ALS) algorithm is the most imperative

techniques which are used for building a movie recommendation engine.

## Гибридные рекомендательные системы

Гибридные рекомендательные системы объединяют подходы контентной и коллаборативной фильтраций, чтобы избежать недостатков одного и другого подходов. Существует несколько подходов к созданию гибридной рекомендательной системы:

1. получить результаты модулей коллаборативной и контентной фильтрации, объединить эти результаты, например, реализовать линейную корреляцию рейтингов [9], выбирать более точные рекомендации [10] или рекомендации, которые больше похожи на историю пользователя [11];
2. внедрить методы контентной фильтрации в систему, основанную на коллаборативной фильтрации, и наоборот; [12]
3. создать общую модель, основанную на методах и коллаборативной, и контентной фильтрации, например, систему, основанную на вероятностном латентном семантическом анализе (PLSA) [13].

## Проблемы рекомендательных систем

Подходы контентной и коллаборативной фильтрации по своему определению не могут предоставить точные рекомендации, так как обладают определёнными недостатками. В данном подразделе рассмотрены недостатки обоих подходов.

### Проблема холодного старта

Любая новая рекомендательная система сталкивается с ситуацией, когда данных недостаточно для составления списков рекомендаций. В зависимости от реализации системы необходимыми данными может быть информация от пользователей или о товарах. Наиболее подвержена такой проблема система, основанная на коллаборативной фильтрации. При малом количестве пользователей сложно анализировать поведение определённого пользователя, а отсутствие взаимодействия пользователей с товаром может привести к тому, что этот товар так и не появится в списке рекомендованных.

### Проблема разреженных данных

В системах, которые полагаются на рейтинги объектов, проставленные пользователями, существует проблема разброса (англ. «sparsity»). У каждого пользователя своё видение шкалы оценки. Например, некоторые оценивают товар только по своему впечатлению от использования, а другие вкладывают в оценку больше значимых факторов, таких как доставка и внешний вид.

Эта проблема также может быть связана с холодным стартом. При малом количестве оценок от пользователей вариация среднего будет выше, чем в случае, когда пользовательских оценок много. Рекомендательная система при усреднении рейтинга объекта должна учитывать такие отклонения и корректировать оценку [[14](https://habr.com/ru/company/lanit/blog/420499/)]: установить «безопасное» среднее значение, которое будет учитываться при подсчёте реального среднего рейтинга или установить интервал доверия для каждого продукта на основе его рейтинга и использовать нижнюю границу этого интервала.

Пользователь также может действовать нетипично [[15](https://www.researchgate.net/publication/49463311_Fulfilling_the_Needs_of_Gray-Sheep_Users_in_Recommender_Systems_A_Clustering_Solution)], то есть у них нет устоявшихся вкусов и они высоко оценивают объекты, противоположные по характеристикам, например, фильмы ужасов и романтичные комедии. При коллаборативном подходе система будет основываться на данных такого пользователя и рекомендовать романтические комедии зрителям, которые любят фильмы ужасов, хотя скорее всего такая рекомендация будет неверной. Это сказывается отрицательно на опыте зрителя в пользовании системой.

### Проблема одинаковых данных

Рекомендательные системы, основанные на контентной фильтрации, наиболее подвержены такой ситуации, когда пользователь становится ограничен в объёме предлагаемых объектов. То есть рекомендательная система работает корректно, предоставляет пользователю новые объекты, которые подходят под его профиль, но пользовательский опыт при этом ухудшается, так как отсутствует разнообразие выбора и элемент неожиданности. При этом понижаются шансы, что пользователь начнёт обращать внимание на что-то новое, а это отразится на его заинтересованности в пользовании системой, а значит, система лишится потенциального клиента.

### Проблема безопасности личных данных

Рекомендательные системы собирают личную информацию о пользователе, которая, возможно, не связана с историей взаимодействия с объектами, такую как день рождения, имя, страна проживания и т.д. Это позволяет улучшить качество рекомендация, сделав их более персональными. Однако в некоторых случаях создатели системы запрашивают больше информации, чем необходимо для функционирования системы рекомендаций, а в случае, если система недостаточно защищена от внешних взломов, то информация о пользователях может попасть в руки злоумышленников. Особым случаем раскрытия личной информации является невозможность установить, кто именно пользуется аккаунтом, так как пользователь системы мог дать доступ своему доверенному лицу, которое вследствие этого получило доступ к рекомендациям владельца аккаунта.

## Обзор существующих систем

Netflix – компания, предоставляющая сервис для просмотра фильмов, сериалов и документальных передач в режиме онлайн. На данный момент является крупнейшим стриминговым сервисом в мире. Эта компания также известна тем, что их конкурс Netflix Prize (2006 г.) дал толчок развитию рекомендательных систем.

Netflix использует в своей рекомендательной системе комбинацию статистических методов и методов машинного обучения [[16](https://dl.acm.org/doi/pdf/10.1145/2843948)], которые включают обучение с учителем (классификация, регрессия) и обучение без учителя (уменьшение размерности с помощью кластеризации и сжатия, т.е. выделение темы [[17](http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf)], которое основано на разложении матриц [[18](https://datajobs.com/data-science-repo/Recommender-Systems-%5bNetflix%5d.pdf)]). Также в системе используются алгоритмы, уменьшающие количество параметров модели [[19](https://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/Regular-Paterek.pdf)], и вероятностные графические модели, которые можно применять к различным предметным областям [20].

Pandora это пример успешного применения рекомендательной системы для подбора музыки. Pandora использует метод ближайших соседей на основе memory-based алгоритмов. Составление списка ближайших соседей для пользователя основано на его предпочтениях, и этот список содержит группы подходящих песен. Время работы алгоритма для создания плейлиста пользователя зависит о количества характеристик песен (например, жанр, темп и т.д.).

## Выводы по разделу

Анализ проблем рекомендательных систем позволил выявить такие проблемы, которые могут возникнуть у разрабатываемой рекомендательной системы: проблема холодного старта, разреженных или повторяющихся данных и проблема безопасности личных данных пользователей. Для того, чтобы избежать проблемы холодного старта, разреженных данных и проблемы безопасности личных данных решено реализовывать рекомендательную систему на основе методов контентной фильтрации. Для решения проблемы повторяющихся данных при проектировании системы добавлен функционал, позволяющий пользователю скрывать фильмы, которые слишком часто попадаются в списке рекомендованных или которые он заочно оценил как непонравившиеся. Система учитывает такие действия и реже предлагает эти фильмы, а также исключает из списка рекомендаций те фильмы, которые пользователь уже видел.

Учитывая специфику данной работы, т.е. составление рекомендаций только на основе подобия текстовых описаний фильмов, рекомендательная система использует результаты только контентной фильтрации, потому как учёт данных о пользователях, т.е. применение коллаборативной фильтрации, усложняет анализ эффективности модифицированного алгоритма определения схожести текстов.

# Обработка текстов на естественном языке

Автоматическая обработка естественного языка позволяет принимать исходные данные в виде человеческого языка и решать задачи, основываясь на полученных таким образом данных.

Область применения систем обработки текстов на естественном языке сейчас очень обширная. Их применяют для усовершенствования систем обучения, редактирования текстов, аннотирования текстов, перевода с иностранных языков, распознавания речи с целью создания голосовых ассистентов, систем идентификации голоса или эмоций, систем рекомендаций, автоматизации колл-центров, исследования «мёртвых языков» и в целом для упрощения взаимодействия человека с компьютером.

Проблемы при разработке систем, которые должны распознавать естественные языки, связаны с тем, что ведётся работа с широким разнообразием форм и структур. Так, например, система, которая работает только с русским языком, не сможет распознать немецкий, т.к. в этих двух языках разные правила составления предложений и морфологические признаки. Также существуют проблемы лексической неопределённости, например, омонимия, синтаксической неопределённости, то есть выделение значимых частей предложения, а также относительная неопределённость, при которой одно понятие может иметь несколько значений, например, в случае использования местоимений.

На рисунке Рисунок 2.1 отображена последовательность этапов извлечения данных из текста с целью получения полезных знаний.

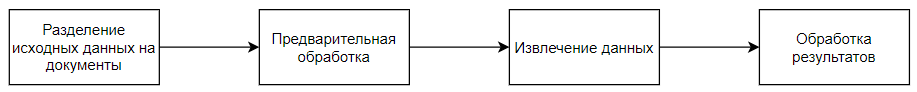


Рисунок 2.1 - Этапы анализа текста

Анализ информации, представленной в текстовом виде, включает в себя [21]:

1. поиск информации;
2. предварительная обработка документов;
3. извлечение полезных знаний;
4. обработка результатов.

На этапе поиска информации определяется набор документов, которые должны быть подвергнуты анализу, и обеспечена их доступность для дальнейшей обработки. В разрабатываемой рекомендательной системе к документам относятся описания фильмов, однако чаще всего алгоритмы работают без тематической привязки, поэтому и исходные данные могут быть разными.

Предварительная обработка документов является общим для всех методов анализа, однако различается в реализации. Все найденные на предыдущем этапе текстовые документы подвергаются предобработке с целью выделения определённой структуры для дальнейшего использования этих данных в методах автоматического определения подобия. Таким образом, из текста удаляются лишние слова и оставшиеся приводятся к единой грамматической форме – так текст обретает более структурированную форму.

На этапе извлечения полезных знаний работают выбранные методы Text Mining для извлечения структурированных данных в текстах. Например, определение частых наборов слов и объединение их в ключевые понятия, вычисление вероятностей принадлежности документа к классу, составление индекса документов для осуществления поиска по ключевым словам, сокращение текста с сохранением смысла, определения степени схожести слов или текстов и др.

Последний этап в процессе обнаружения полезной информации – обработка результатов – решает задачу анализа полученных результатов. Результатом работы в рекомендательной системе является создание списка рекомендованных объектов.

В данной работе рассматривается применение методов извлечения информации для определения степени схожести документов. Существует множество алгоритмов, позволяющих определить близость текстов. Её определяют с разными целями: установить подлинность документа или определить наличие плагиата; выявить спам в email-письмах; в системах рекомендации по содержимому документов; в поисковых системах для индексации документов и т.п. Наиболее схожие тексты имеют одинаковый набор словоформ (обработанных слов), то есть количество вхождений, и их последовательность.

Для того, чтобы определить степень схожести документов, необходимо преобразовать их из естественного языка в такой формат, к которому можно будет применить алгоритмы. Алгоритмы определения схожести в свою очередь имеют определённый формат входных данных (документов) и набор инструкций для определения расстояний между документами, а также легко интерпретируемый выходной результат, чаще всего это число в диапазоне [0, 1].

Схожесть документов так или иначе основана на данных о содержащихся в них словах. Слова могут быть подобны лексически и семантически. Лексическое подобие подразумевает совпадение последовательности символов. Семантическое подобие подразумевает использование слов в похожих контекстах или подобным образом. Данные подходы не выявляют значение слов в таком виде, как их понимает человек, а лишь ищут зависимости и используют статистические метрики.

Алгоритмы лексического подобия называют строковыми метриками (String-based) и представляют собой поиск расстояний между строками текстов, алгоритмы семантического подобия делятся на Corpus-based и Knowledge-based. Corpus-based алгоритмы производят поиск подобных документов на основе исходных данных, полученных для реализации конкретной задачи и даже на определённую тему. Knowledge-based алгоритмы определяют степень схожести документов на основе семантических сетей [45].

## Строковые метрики

Строковые метрики [[22](https://en.wikipedia.org/wiki/String_metric)] это группа алгоритмов, которые работают с последовательностью подстрок. Подстроками могут служить как отдельные слова, так и последовательность слов или букв. Мерами подобия в таких алгоритмах являются метрики поиска расстояний между символами и подстроками. В контексте целых документов это означает поиск подобия между ними на основе статистических данных, полученных о подстроках. Наиболее известны такие string-based алгоритмы определения подобия текстов:

1. Longest Common Substring (LCS) определяет сходство подобия текстов на основе длин всех одинаковых подстрок, содержащихся в текстах;
2. расстояние Дамерова-Левенштейна определяет расстояние между подстроками как минимальное количество операций, необходимых для превращения одной строки в другую;
3. алгоритм n-gram, который сравнивает тексты, основываясь на представлении текста как набора n-грамм и поиска одинаковых n-грамм в сравниваемых текстах.

Наиболее простой способом определения, являются ли тексты одинаковыми, это приведение документов в формат хешей и сравнение этих значений. Хеш-строка – результат работы алгоритма хеширования, который приводит текст любого размера в строку фиксированного размера. Хеши разных текстов будут иметь различное значение. Однако такой способ слишком прямолинейный и не подходит для определения частичной схожести текстов: любое изменение даже одного знака в одном из текстов приведёт к изменению хеша этого текста. Одним из решений этой проблемы будет разбиение документа, обработанного алгоритмами нормализации, на подстроки, которые будут обработаны алгоритмом n-gram.

Алгоритм шинглов [2[7](https://www.researchgate.net/publication/276487393_Establishing_Semantic_Similarity_of_the_Cluster_Documents_and_Extracting_Key_Entities_in_the_Problem_of_the_Semantic_Analysis_of_News_Texts)] также называют алгоритмом n-грам, так как текст разбивается на массивы токенов (т.е. обработанных слов) по n штук в каждом. Например, при размере шингла равному n=3 текст «It was the best of times» будет представлен как список четырёх массивов из трёх элементов каждый: {[it was the], [was the best], [the best of], [best of times]}. После разбиения каждого документа на подстроки происходит поиск количества одинаковых подстрок в сравниваемых текстах, чем больше совпадений, тем более похожи тексты.

Для определения степени схожести между двумя словами можно представить их в виде векторов и рассчитать расстояние между ними. Для оценки подобия документов при применении String-based алгоритмов используют такие метрики как косинусное расстояние, манхэттенское расстояние, коэффициент Дайса, Жаккара, Евклидово расстояние и др., которые работают с количеством вхождений подстроки в текст. Данные алгоритмы являются наиболее простыми в реализации, а также показывают хорошие результаты при определении подобия текстов [[26](https://core.ac.uk/download/pdf/15169266.pdf)].

Евклидово расстояние (2.1) это расстояние между двумя точками евклидова пространства [[23](https://ru.wikipedia.org/wiki/%D0%95%D0%B2%D0%BA%D0%BB%D0%B8%D0%B4%D0%BE%D0%B2%D0%BE_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE)], которое вычисляется по теореме Пифагора.

, (2.1)

где и – значения векторов двух документов для слова *i*

Подобно Евклидовой метрике, мера Cityblock, также называемая Манхэттеновским расстоянием (2.2), и расстояние Чебышева (2.3) определены в Манхэттеновском пространстве [[2](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%B3%D0%BE%D1%80%D0%BE%D0%B4%D1%81%D0%BA%D0%B8%D1%85_%D0%BA%D0%B2%D0%B0%D1%80%D1%82%D0%B0%D0%BB%D0%BE%D0%B2)4] и пространстве Чебышева [[25](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%A7%D0%B5%D0%B1%D1%8B%D1%88%D1%91%D0%B2%D0%B0)]. В данных формулах a и b – векторы документов, I – номер слова в документе, при условии, что *a* и *b* имеют одинаковый размер.

(2.2)

(2.3)

Наиболее известным методом является нахождение косинусной меры. Косинусная мера подобия слов или документов, представленных векторами *a* и *b* может быть вычислена как (2.4).

(2.4)

Корреляция – это показатель связи между исследуемыми документами. Отсутствие корреляции означает, что изменение одной величины не приводит к изменению другой. Коэффициенты корреляции могут быть положительными и отрицательными. Если при увеличении значения одной величины происходит уменьшение значений другой величины, то их коэффициент корреляции отрицательный. В случае, когда увеличение значений первого объекта приводит к увеличениям значения второго объекта, то можно говорить о положительном коэффициенте.

Корреляция Пирсона (2.5) показывает корректные результаты только при линейной зависимости между величинами. Для того, чтобы применять корреляцию Пирсона, необходимо, чтобы выполнялись следующие условия для исследуемых величин:

1. величины имеют нормальное распределение;
2. величины имеют значения в определённом интервале;
3. размер векторов величин одинаковый.

Недостатком данного метода является неустойчивость к случайным выбросам.

r_{xy}=\frac{\Sigma(x_i-\bar{x})\times(y_i-\bar{y})}{\sqrt{\Sigma(x_i-\bar{x})^2\times\Sigma(y_i-\bar{y})^2}}, (2.5)

где x_i — значения переменной X;

y_i — значения переменной Y;

\bar x — среднее арифметическое для переменной X;

\bar y -среднее арифметическое для переменной Y.

Для того, чтобы интерпретировать результат вычисления метрики, необходимо задать порог доверия, например, 80%, при котором система будет давать однозначный ответ на вопрос «похожи ли объекты?». Таким образом решается проблема нахождения нечётких дубликатов.

Преимуществом представленных подходов является то, что они не требуют определения смысла текста для того, чтобы определить его схожесть с другими текстами. Недостатком является скорость инициализации первичной коллекции документов, т.к. разбиение на шинглы и попарное их сравнение требует много времени. Также данные алгоритмы неспособны выявить похожесть текстов, в которых слишком малое количество похожих слов, но эксперты определяют эти тексты как похожие.

## Методы поиска подобия документов на основе их содержания

Corpus-based алгоритмы анализируют набор документов и с помощью статистических мер выявляют схожесть между этими документами. При изменении набора документов такие алгоритмы должны перестраивать свои базы данных.

### Word2Vec

Word2Vec это набор алгоритмов, разработанных для получения векторных представлений слов. На данный момент наибольшую популярность приобрели подходы CBOW (Continuous Bag Of Words) и Skip-gram. [[28](https://arxiv.org/pdf/1301.3781.pdf)], оба основаны на предварительном обучении.

На рисунке Рисунок 2.2 представлено различие между процессами обработки данных в алгоритмах CBOW и Skip-gram.

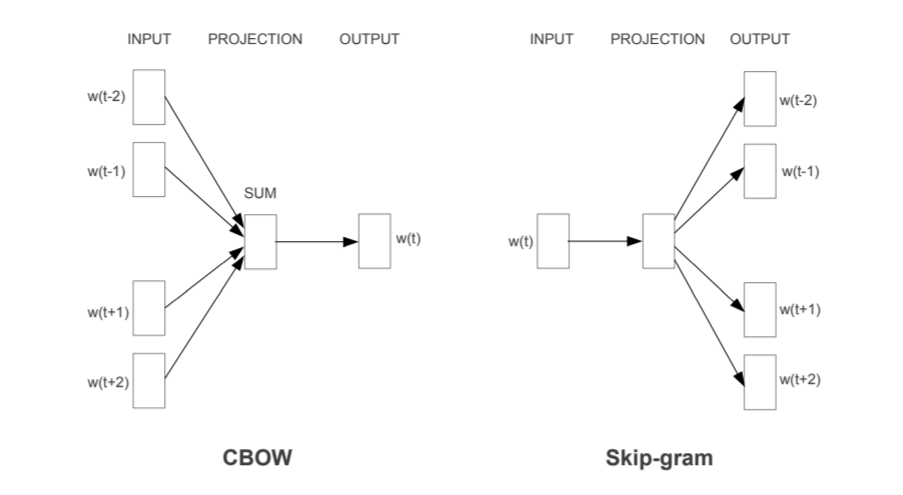


Рисунок 2.2 - Виды алгоритмов метода Word2Vec

В модели CBOW распределённое представление контекста собирается, чтобы предсказать среднее слово. Модель Skip-gram используют, чтобы предсказать контекст по распределённым представлениям слова.

При обучении нейронной системы решают не прямую задачу предсказания слов, а т.н. «фальшивую задачу»: по входному среднему слову (Skip-gram) или нескольким словам (CBOW) алгоритм случайно выбирает одно из слов, похожих на входное, и для каждого слова в тестовой выборке предсказывает вероятность того, что именно оно случайно подходит. Однако настоящей задачей, которая поставлена перед системой, является вычисление весов на скрытых слоях, которые и являются векторами слов в матрице.

Skip-gram хорошо работает при небольших обучающих выборках, предсказывая даже редкие сочетания слов, CBOW обучается гораздо быстрее, чем Skip-gram, и показывает чуть лучшие результаты для наиболее часто встречающихся слов. [[29](http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/)].

### LSA

Модель LSA (Latent Semantic Analysis) использует метод разложения матриц SVD (Singular Value Decomposition), чтобы определять схожесть документов. Впервые его предложили в [[30](https://psycnet.apa.org/record/1997-03612-001)]. При данном подходе термы в документах представляют не n-мерной матрицей, где n – количество разных слов в документах, а уменьшают размерность векторов так, что оси представляют собой группы схожих слов. Таким образом, несложно вычислить сходство между документами методом косинусного сходства.

С помощью данного метода можно выявить синонимы почти так же, как человек. В работах [[31]](%5b31%5d) и [[32]](https://www.cs.cmu.edu/~ralucav/papers/riao.pdf) результат теста TOEFL на выявление синонимов показывает около 60% верных ответов. Данный метод также является робастным, т.е. устойчив к случайным выбросам – редким или слишком частым словам в документах. Также метод LSA не требует обучения, а значит, менее трудоёмкий, чем, например, создание искусственного интеллекта.

Недостатком этого подхода является то, что результаты всегда точные, нет диапазона вероятности сочетаний терминов. Также результаты работы алгоритма сложно интерпретировать, а сложность метода SVD, который является основой LSA, достигает при размерности исходной матрицы *m\*n.*

### ESA

Метод Explicit Semantic Analysis (ESA) представляет документы как взвешенные вектора терминов, взятых из размеченного корпуса текстов. Наиболее часто используют страницы Википедии [33], т.к. каждому слово можно соотнести его понятие, а также каждая страница относится к какой-либо категории.

Алгоритм во многом похож на LSA, т.к. для сравнения документов также используется взвешивание исходной матрицы и косинусная мера для определения подобия, однако ESA не используется матричное разложение и исходная матрица строится не по документам коллекции, а по общему словарю Википедии.

Таким образом, используя алгоритм ESA, можно определить семантическое подобие любых текстов, основываясь на размеченных самим человеком данным, в отличие от LSA, где алгоритм сам находит латентные связи между словами.

## Топологические методы, основанные на использовании семантических сетей

Knowledge-based метрики для выявления схожести между документами используют иерархические семантические сети [[34](https://www.aclweb.org/anthology/W11-2502.pdf)]. Некоторые метрики также используют количественные меры, извлечённые из набора документов.

По сравнению с Corpus-based алгоритмами, которые исследуют семантическое подобие текстов, Knowledge-based алгоритмы исследуют семантическую связь между текстами.

Семантическая связь представлена в таких сетях как MeSH [[35](https://www.nlm.nih.gov/mesh/meshhome.html)] и WordNet [[36](https://wordnet.princeton.edu/)]. Существуют такие Knowledge-based меры, определяющие семантическую связь между текстами, используя все связи семантической сети и используя только отношения гипонимии [[37](https://en.wikipedia.org/wiki/Hyponymy_and_hypernymy)].

По алгоритму Hirst-St-Onge [[38](https://www.researchgate.net/publication/2735129_Lexical_Chains_as_Representations_of_Context_for_the_Detection_and_Correction_of_Malapropisms)] два понятия семантически связаны, если их пути в семантической сети не слишком различны и не слишком часто меняют направление.

Алгоритм Leacock–Chodorow [[39](https://www.researchgate.net/publication/200045856_Combining_Local_Context_and_WordNet_Similarity_for_Word_Sense_Identification)] основан на выявлении кратчайшего пути между понятиями.

Наиболее популярным алгоритмом данного типа является Resnik [[40](http://www.jimdavies.org/summaries/resnik1995.html)], который связывает онтологию и набор документов. Связь между документами вычисляется с помощью общих элементов этих документов и используя узлы семантической сети для существительных.

## Выводы по разделу

Алгоритм шинглов показывает хорошие результаты при сравнении длинных текстов [44], т.к. сравнение словосочетаний без учёта смысла позволяет выявить одинаковые части текстов сравнительно быстро, однако такой способ сравнения текстов работает хуже на небольших текстах, т.к. каждый текст предоставляет слишком ограниченный набор конкретных n‑грамм.

Сравнительный анализ подходов LSA и Word2Vec [[41](https://www.researchgate.net/publication/322160621_Corpus_specificity_in_LSA_and_Word2vec_the_role_of_out-of-domain_documents)] показал, что при работе с текстами единой тематики LSA показывает хорошие результаты при небольшом количестве анализируемых документов, в то время как другое исследование [[42](https://www.researchgate.net/publication/308896339_Comparative_study_of_LSA_vs_Word2vec_embeddings_in_small_corpora_a_case_study_in_dreams_database)] определило, что Word2Vec показывает наилучшие результаты только при обучении на больших выборках (ок. 10 млн коротких текстов), что является невозможным в рамках предметной области данной работы. Также по результатам исследований в [43] выявлено, что LSA превосходит ESA в эффективности при использовании косинусной меры сравнения.

Knowledge-based методы являются сложными для реализации, а также не найдены зафиксированные подтверждения большей эффективности этих методов в задачах определения подобия документов, чем LSA.

В связи с этими результатами в данной работе в качестве исходного метода исследования схожести текстов выбран подход LSA (Latent Semantic Analysis).

# Обзор метода латентно-семантического подобия

На основании исследования алгоритмов наиболее подходящим для решения задачи поиска подобных документов был выявлен метод LSA, основанный на составлении матрицы bag-of-words (слово-документ) и использовании косинусной меры сравнения, поэтому данный метод был выбран для дальнейшей модификации.

## Анализ алгоритма LSA

Латентно-семантический анализ (Latent Semantic Analysis) используется для моделирования значений слов с использованием статистических вычислений, применённых к коллекции исследуемых документов. Впервые алгоритм был применён для индексирования текста и выявления его семантической структуры [47], а также использован Landauer и Dumais [48] для моделирования процесса получений новых знаний человеком.

LSA используют как универсальный подход для построения когнитивных моделей, например, моделирование процесса обучения детей [[46]](http://www.dialog-21.ru/digests/dialog2014/materials/pdf/SolovyevAN.pdf), т.к. он зависит только от исходных слов исследуемых документов и позволяет гибкую настройку процесса выявления связей между словами (выбор размерности матрицы, метрики для заполнения значений матрицы терм-документ, метрики схожести). На рисунке Рисунок 3.1 представлена UML диаграмма активности, изображающая шаги классического алгоритма LSA.

Инициализация семантического пространства осуществляется посредством создания матрицы терм-документ *A*. Термы *W*, которые составляют строки матрицы *A*, предварительно обрабатывают. В классическом LSA значениями матрицы *A* является частота вхождений слова в документ , однако с развитием алгоритмов извлечения информации наибольшую популярность приобрела метрика TF-IDF, значения которой представляют собой вес слова слова в документе , и которая, таким образом, решает проблему разреженности данных.

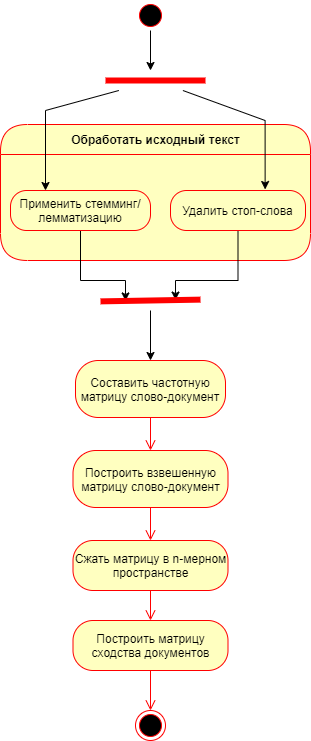


Рисунок 3.1 - Диаграмма активности алгоритма LSA

Главным преимуществом LSA является то, что этот метод позволяет управлять степенью полноты и точности результатов с помощью изменения порядка матриц в алгоритме SVD [49]. Согласно теореме Эккарта–Янга [[50](https://link.springer.com/article/10.1007/BF02288367)], при замене исходной матрицы *A* на результат работы алгоритма SVD (матрицу *A\**) большая часть малозначимой и случайной информации теряется, зато отчетливей проявляются скрытые тематические тренды. По векторам матрицы *A\** вычисляют близость документов, используя меру сходства. Наиболее эффективной и популярной является косинусная мера сходства.

### Предобработка текста

Решение большинства задач по извлечению знаний начинаются с предварительной обработки исходного текста. Целью такой обработки является приведение текста к унифицированному виду, с которым могут работать большинство алгоритмов извлечения информации.

Текст на естественном языке состоит из знаков, которые можно по-разному разбить для дальнейшего анализа. Лингвисты выделяют уровни предложений (синтаксический), слов (морфологический) и фонем [52]. В письменной речи вместо уровня фонем выделяют уровень символов. В связи с этим можно выполнять графематический анализ (выделение токенов), морфологический анализ (выделение основ слов), синтаксический анализ (выявление синтаксических связей слов и грамматической структуры предложений) и семантический анализ (выделение смысла слова по его контексту). Существуют такие методы предобработки текстов [[51](https://zenodo.org/record/3572739#.Xn8tUogzaMo)]:

1. токенизация, то есть разбиение текста на части: предложения, n-граммы, слова;
2. нормализация, которая может включать в себя преобразование цифр в словесный формат или их удаление, удаление знаков препинания, приведение всех слов текста в единый регистр, расшифровка сокращений, удаление слов, не несущих смысл;
3. стемминг – процесс нахождения основы слова; поиск может происходить по заранее заданным таблицам, с помощью усечения окончаний и аффиксов и др.;
4. лемматизация – процесс приведения слова к его нормальной форме, например, приведение глагола к инфинитиву несовершенного вида, а существительного – к единственной форме именительного падежа.

Для обработки текста можно использовать как любой из этих подходов в отдельности, так и их комбинацию.

### Векторное представление текста

Для поиска семантического подобия в любых целях наиболее широко используют векторные модели [5[3](https://web.stanford.edu/~jurafsky/slp3/edbook_oct162019.pdf)]. И хотя существующие сейчас алгоритмы не понимают значения слов так, как человек, но они определяют их значения по тому, как они используются в языке.

Векторные модели определяют слово по его окружению (контексту) или распределению по использованию в тексте. Таким образом, слова, которые имеют подобный контекст скорее всего будут подобными. Это решает проблему омонимии и синонимии, а также позволяет решать большое множество проблем, связанных с обработкой текстов, в том числе и с извлечением скрытых данных.

Представить слово в виде вектора можно разными способами: подсчётом слов из контекста или как точку в n-мерном пространстве, которую проецируют на двумерную плоскость. В таком случае удобно распределять слова по группам, например, для анализа тональности текста. На рисунке отображена категоризация слов по темам в многомерном и в упрощённом двумерном пространствах.

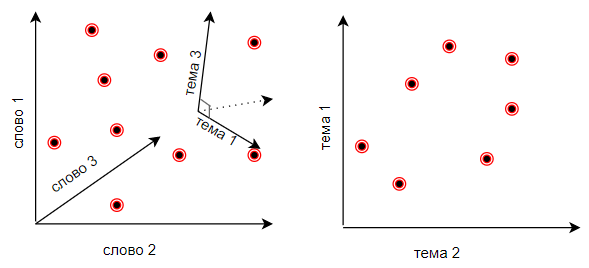
****

Рисунок 3.2 - Представление векторов слов в n-мерном пространстве (слева) и в двумерном (справа)

Для того, чтобы выполнить представить текстовый документ в понятном алгоритму виде, необходимо для начала составить частотные матрицы размером *m\*n*, где пересечением слова (строки *n*) и документа (столбца *m*) будет количество употреблений слова в документе. Такая матрица позволяет не только оценить схожесть документов (чем более подобны строки, тем более подобны столбцы), но и определить значение слова по документам, в которых оно встречается.

Однако для определения значения слов вместо матрицы слов и документов могут составить также матрицу слов и их контекста, размерность которой будет *n\*n*, а пересечение слова и другого слова (контекста) это количество раз, когда слово из строки встречается в том же документе, что и слово из столбца. Матрицу, в которой много слов не имеют соответствующего контекста, называют разреженной.

### Составление матрицы слово-документ

Создание матрицы соответствия слов и документов сочетает в себе предобработку текстов и составление индекса – использование такой структуры представления информации, чтобы последующие обращения к ней были наиболее эффективными. В наиболее упрощённом варианте это создание матрицы размером *m\*n*, где *m* – количество уникальных предобработанных слов по всем документам, *n* – количество документов, которые необходимо сравнить. Документом может быть как целая книга или статья, так и контекст – слова, окружающие какой-либо токен*.* Контекстом может служить предложение или n-грамма.

Инициализация матрицы слово-документ происходит с помощью bag-of-words – подхода, при котором для каждого предобработанного слова составляется частотный вектор, где количество употреблений слова соответствует документу, в котором оно содержится. Например, для предложений «Жизнь главного героя переворачивается с ног на голову» и «Герой спас ребёнка, перевернув каменный блок» можно составить такую матрицу bag-of-words, как в таблице 3.1.

Таблица 3.1 – Пример разбиения текстов на bag-of-words

|  |  |  |
| --- | --- | --- |
| Слова | Документ-1 | Документ-2 |
| жизнь | 1 | 0 |
| главный | 1 | 0 |
| герой | 1 | 1 |
| перевернуть | 1 | 1 |
| нога | 0 | 0 |
| голова | 0 | 0 |
| спасать | 0 | 1 |
| каменный | 0 | 1 |
| блок | 0 | 1 |

Значения матрицы слово-документ могут использоваться для определения связей между словами эффективно до тех пор, пока все исследуемые документы имеют одинаковый размер. В случае, когда некоторые тексты будут гораздо короче остальных, частота слов, встречающихся в них, также станет меньше. Для того, чтобы уравнять значимость слов в контексте документа используют меру TF-IDF (Term Frequency-Inverse Document Frequency) [54]. Вес слова является отношением частоты употребления слова в документе к частоте употребления слова во всех документах. Для ослабления эффекта IDF используют логарифм любого порядка.

В случае, когда слово встречается во всех документах, мера TF-IDF его игнорирует, что решает проблему удаления стоп‑слов.

По формуле (3.1) значением ячейки матрицы на пересечении слова *i* и документа *j* является вес слова *i* в документе *j*. Мера TF-IDF определяет значимость слова в контексте, соотнося количество раз, когда слово *i* встречается в документе *j* (TF) с количеством документов, в которых встречается слово *i* (IDF).

, (3.1)

где – число вхождений слова *i* в документ *j*;

– количество слов в документе *j;*

*N* – количество документов в коллекции;

– количество вхождений слова *i* в коллекцию документов.

Учёт инверсии частоты (IDF) уменьшает вес для слов, которые используются в большинстве документов и повышают вес малоупотребляемых слов. Основание логарифма может быть любым, т.к. его изменение лишь изменяет каждое значение веса на постоянную величину.

В качестве примера рассмотрим такой случай. Значения в таблице 3.2 – это частота появления слов. Допустим, каждый документ содержит 100 слов.

Таблица 3.2 – Исходные данных для примера TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
|  | Документ\_1 | Документ\_2 | Документ\_3 |
| условие | 3 | 8 | 0 |
| информация | 5 | 0 | 10 |
| значение | 18 | 3 | 10 |

В таблице 3.3 представлены результаты подсчёта веса каждого слова по трём документам.

Таблица 3.3 – Результат вычислений примера TF-IDF

|  |  |  |  |
| --- | --- | --- | --- |
|  | Документ\_1 | Документ\_2 | Документ\_3 |
| условие | 3/100 \* 3/2 = 0.045 | 8/100 \* 1.5 = 0.12 | 0 |
| информация | 5/100 \* 3/2 = 0.075 | 0 | 3/100 \* 3/2 = 0.15 |
| значение | 18/100 \* 3/3 = 0.18 | 10/100 \* 3/3 = 0.03 | 10/100 \* 3/3 = 0.1 |

Таким образом, в документе\_1 наибольший вес имеет слово «значение», в документе\_2 – «условие», в документе\_3 – «информация». Эти значнения уже можно использовать при вычислении меры схожести. Однако согласно алгоритму LSA, необходимо произвести сжатие матрицы, прежде чем приступать к обработке результатов.

### Сингулярное разложение матриц (SVD)

Уменьшение размерности данных необходимо по многим причинам: разреженность данных в матрице, которая может возникать из-за некорректных значений при её составлении, что особенно критично при нейросетевом обучении; слишком большое количество векторов слов и документов, что влияет на производительность системы.

Понижение размерности – преобразование данных большой размерности в такое пространство пониженной размерности, в которой не теряется информация, заключённая в исходных данных.

Существует несколько методов разложения матриц: LU, QR, разложение Холецкого и SVD [[55](https://royalsocietypublishing.org/doi/full/10.1098/rspa.2014.0585)]. LU-разложение представляет матрицу *A* как произведение верхней и нижней треугольных матриц, QR-разложение – как произведение ортогональной и верхнетреугольной матриц, а разложение Холецкого – как произведение нижнетреугольной матрицы на её же транспонированную матрицу.

Метод SVD позволяет разложить матрицу *A* на произведение трёх матриц по формуле (3.2).

, (3.2)

где *U* – ортогональная матрица размером *m\*k;*

*S* – диагональная матрица размером *k\*n*, элементы которой по главной диагонали являются сингулярными значениями *А*, расположенными в порядке убывания;

*V* – ортогональная матрица размером *k\*k*.

Преобразуем формулу через суммирование (3.3).

, (3.3)

где *n* – количество сингулярных значений (желаемый порядок результирующей матрицы).

Такое разложение с урезанием матриц *U* и *V* называется усечённым. Диагональная матрица *S* здесь представлена вектором. Столбцы матриц *U* и *V* называют левым и правым сингулярными векторами, которые соответствуют векторам слов и документов соответственно.

Рассмотрим алгоритм SVD на примере текстовых документов. Например, матрица TF-IDF имеет такие значения, как на рисунке Рисунок 3.3.



Рисунок 3.3 - Исходные значения матрицы слово-документ

Тогда матрицы будут разложены как на рисунке Рисунок 3.4.

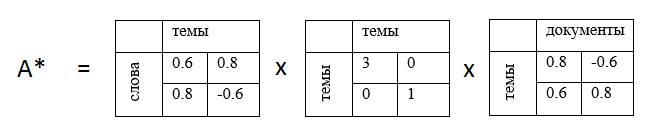


Рисунок 3.4 - Разложение матрицы А по алгоритму SVD

Результатом работы алгоритма будет матрица терм-документ порядка *k*. Дальше эту матрицу можно преобразовать в векторы документов и вычислить меру подобия каждой пары документов. В классическом LSA используют косинусную меру подобия.

### Определение подобия документов

Для того, чтобы оценить схожесть документов, необходимо применить меру сходства: евклидово расстояние, манхэттенское расстояние, расстояние Чебышёва, Cityblock (манхэттенское расстояние), коэффициент Танимото, коэффициент корреляции, косинусная мера и др. [[56](https://www.aclweb.org/anthology/W14-1503.pdf)]

Косинусная мера получила наибольшее распространение в системах, исследующих подобие документов. Её значения находятся в интервале [0, 1] и не зависят от абсолютных значений исходной матрицы.

Пусть *A* и *B* – векторы документов. Косинусная мера выражается формулой (3.4).

, (3.4)

где – значение ячейки в векторе первого документа;

– значение ячейки в векторе второго документа.

Однако алгоритм SVD, предшествующий нахождению косинусной меры между документами, может оставлять отрицательный значения в векторах, поэтому в некоторых системах косинусную меру нормализуют, например, меняют отрицательные значения матрицы на нули.

Косинусная мера чувствительна к изменению вектора слов, т.е. её значение может измениться в любую сторону при добавлении нового терма. Вследствие этого применение данной меры в системе рекомендаций приведёт к тому, что объекты, которые могли попадать в список рекомендованных для определённого пользователя, перестанут попадать в этот список при пересчёте индекса – операции, которую нужно проводить при добавлении новых объектов в систему.

## Разработка модификации алгоритма LSA

Для решения задачи определения степени схожести коротких текстов в оригинальный алгоритм LSA необходимо внести некоторые изменения. Это обусловлено тем, что короткие тексты могут состоять из всего одного короткого предложения, а значит, предоставлять мало информации для анализа или искажать его.

Учитывая специфику данных, с которыми необходимо работать, необходимо начать учитывать такой дополнительной параметр, как наличие или отсутствие термина в тексте. Эту задачу в некоторой степени решает мера взвешивания TF-IDF и дальнейшее разложение матриц SVD, что даёт нам ненулевые значения, которые отражают семантический вес слова в коллекции документов. Однако в коротких текстах, например, описаниях к фильмам, важную роль играет не просто семантическое подобие, сам факт наличия или отсутствия слова в сравниваемых документах, потому что специфичные слова (например, «супергерой» или «университет») могут указывать на наиболее вероятное подобие данных документов. Таким образом, предлагается изменить косинусную меру подобия так, чтобы учитывалось наличие и отсутствие терминов в двух документах [67].

Вследствие обработки документов алгоритмом TF-IDF получаем равноразмерные векторы документов. Для того, чтобы сравнить два документа A и B, нужно составить для них взвешенный вектор *p* наличия и отсутствия терминов, как описано в формуле (3.5).

(3.5)

Значения весов для вектора *p* выбраны экспериментальным путём. В случае, если оба значения имеют положительные значения, это свидетельствует о том, что оба документа содержат термин по индексу *j*.

Далее выполняется обработка матрицы, полученной вследствие работы алгоритма TF‑IDF, алгоритмом SVD, как и в оригинальном методе. Далее необходимо применить вместо стандартной косинусной меры её модификацию. Для каждого документа *D* вводится метрика  (см. формулу 3.6).

, (3.6)

где *i* – номер документа,

*j* – номер слова.

Формула (3.4) должна быть изменена так, чтобы повышать значимость тех слов, которые встречаются в обоих документах, на основе вектора . Модифицированная формула представлена в (3.7).

(3.7)

Кроме того, для упрощения обработки результатов значения векторов документов приводятся к нормальному виду, т.е. все отрицательные значения переводятся в 0.

Таким образом, формула (3.7) позволяет учитывать наличие термина в обоих документах для достижения более точных результатов при решении задачи поиска схожих документов по методу LSA.

## Выводы по разделу

В данном разделе подробно рассмотрен алгоритм LSA (Latent Semantic Analysis), выявляющий семантическое подобие текстов. Описаны этапы данного алгоритма и представлены методы, используемые в его классическом представлении.

Указана необходимость предварительной обработки текстов при автоматизации любых подходов к извлечению знаний, описана векторная модель документов как наиболее подходящая для работы с неформальными данными. Детально рассмотрен такой способ взвешивания документов как мера TF-IDF на основе частоты появления слов в документах. Далее описана косинусная мера как способ определения подобия векторов.

Также представлена модификация косинусной меры, которая учитывает не только значения векторов, но и наличие и отсутствие термина в документе.

# Проектирование рекомендательной системы AdviseMe

## Анализ требований к системе

Для того, чтобы метод определения схожести текстов показывал наилучшие результаты, необходимо проецировать его на предметную область. Предметной областью в данной работе является рекомендательная система фильмов, и, таким образом, можно составить список пользовательских историй (англ. «user stories»):

1. как пользователь я хочу получать рекомендации на основании моих действий в системе.
2. как пользователь я хочу иметь возможность оценивать фильмы по пятибальной шкале.
3. как пользователь я хочу иметь возможность однозначно оценить фильм как «понравился» или «не понравился».
4. как пользователь я хочу получать рекомендации даже в случае, если я не совершаю активных действий в системе.
5. как администратор системы я хочу осуществлять обновление базы данных фильмов автоматически.
6. как администратор системы я хочу получать анализ эффективности используемого алгоритма определения схожести текстов.

Кроме пользовательских историй необходимо также определить функциональные требования к системе:

1. система должна хранить информацию пользователя в базе данных.
2. система должна анализировать данные, полученные от пользователя вследствие использования им системы.
3. система должна собирать информацию о фильмах с внешнего источника, в данном случае – tmdb api, и сохранять её в базу данных.
4. система должна анализировать информацию о фильмах, сохранённых в базе данных.
5. система должна сопоставлять результаты анализа фильмов с результатами анализа профиля пользователя.
6. система должна выводить пользователю результаты анализа фильмов в сочетании с предпочтениями пользователя в виде списка.
7. система должна иметь графический интерфейс.
8. система должна производить обновление и анализ списка фильмов автоматически через определённые промежутки времени.

Для формализации и уточнения требований необходимо оформить их в виде вариантов использования. В качестве способа представления вариантов использования выбрана use-case диаграмма UML [5[7](https://www.informit.com/store/unified-modeling-language-user-guide-9780321267979)]. Данная диаграмма позволяет определить пользователей системы, внешние компоненты, не относящиеся к системе, и их взаимодействие. На рисунке Рисунок 4.1 представлена общая use-case диаграмма всей системы, на рисунке 4.2 – подробная use-case диаграмма варианта использования «Взаимодействие с фильмами», на рисунке 4.3 – подробная use-case диаграмма варианта «Оценка системы».

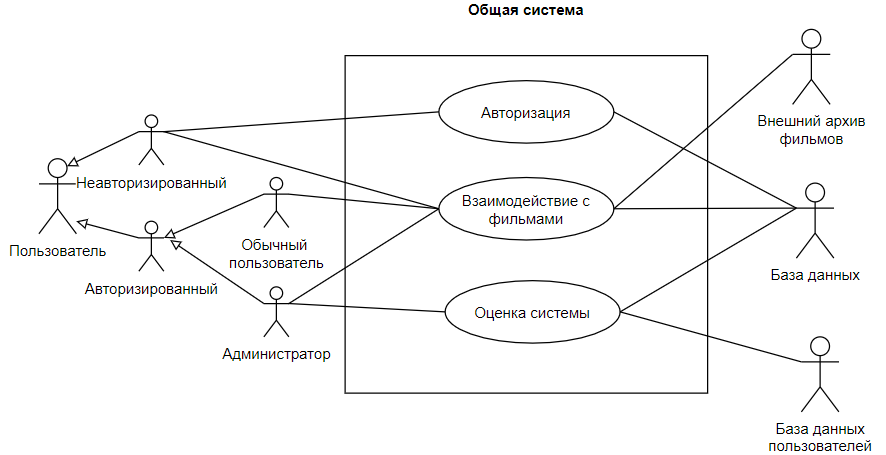


Рисунок 4.1 - Use-case диаграмма системы

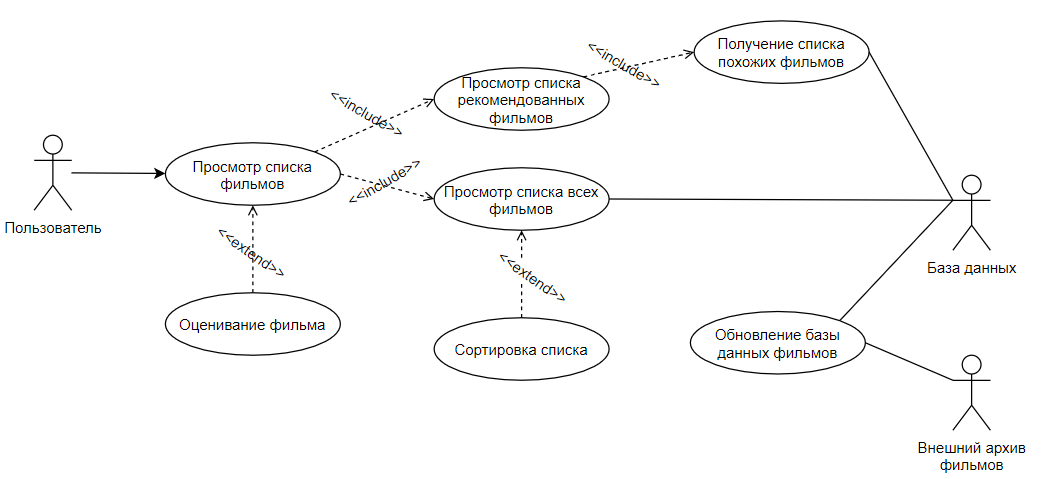


Рисунок 4.2 - Use-case диаграмма варианта использования «Взаимодействие с фильмами»

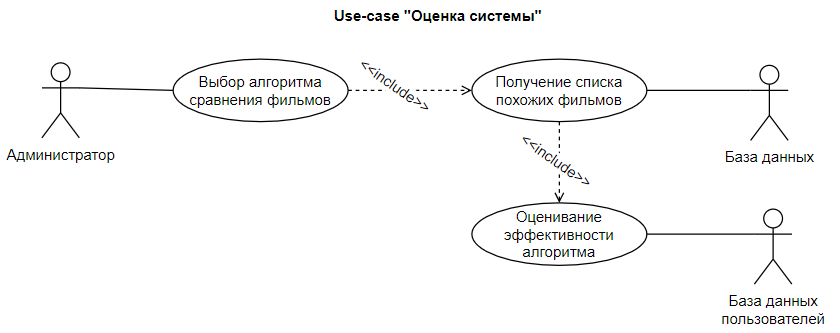


Рисунок 4.3 - Use-case диаграмма варианта использования «Оценка системы»

Роли пользователей различаются в правах доступа к статистической оценке эффективности рекомендательной системы. Администратор может выбрать алгоритм, который необходимо оценить, и запустить процесс оценивания. Для того, чтобы произвести оценивание, необходимо проанализировать поведение пользователей, но так как у разрабатываемой системы нет возможности провести долговременный сбор информации об использовании приложения многими пользователями, то решено использовать публичную базу данных GroupLens [[58](https://grouplens.org/datasets/movielens/)], которая связана с TMDb API. Эта база данных содержит информацию о пользователях и фильмах, которые они оценили на протяжении некоторого времени.

Актёр «База данных» позволяет хранить информацию о фильмах и пользователях для её дальнейшей обработки сервисами системы.

Актёр «Внешний архив фильмов» представлен в системе как интерфейс доступа к внешним API, которые предоставляют всю необходимую информацию о фильмах. В данном случае в качестве провайдера информации о фильмах выступает TMDb API [[59](https://developers.themoviedb.org/3)].

Рекомендательная система реализована на языке Java [62] в с использованием нереляционной базы данных MongoDB [63], поэтому системными требованиями для запуска программы являются установленная JVM и сформированная строка подключения к удалённой базе данных MongoDB.

## Проектирование базы данных

Учитывая объём данных, которые нужны для полноценной работы системы, необходимо организовать их хранение в базе данных. Вследствие анализа требований разработана модель базы данных как на рисунке Рисунок 4.4.

Таблица «Users» ассоциирует пользователя с фильмом, который он оценил. Таблица «Movies» описывает свойства фильмов, необходимые для анализа их схожести, а также для составления списка рекомендаций. Таблица «MoviesList» содержит информацию об оценке фильма пользователем. Таблица «CompareResult» содержит результаты работы алгоритмов LSA, ESA и n-gram, которые генерируются и используются в рекомендательной подсистеме.

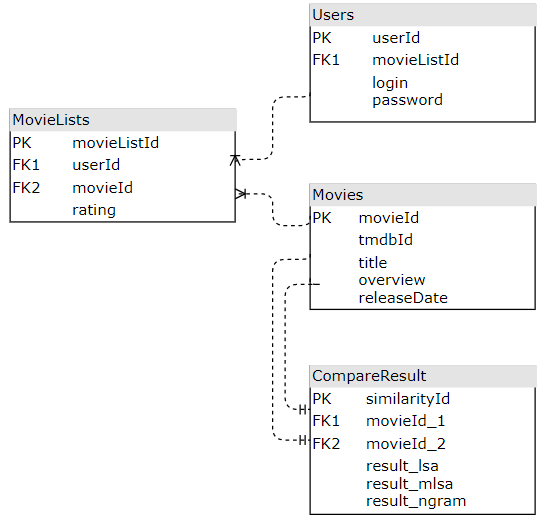


Рисунок 4.4 - Модель базы данных

Для разработки рекомендательной системы используется нереляционная база данных MongoDB, которая позволяет свободно хранить любые типы данных, не накладывая ограничений на структуры, а также мало зависит от внешних ключей, в отличие от реляционных баз данных таких как, например, MySql. Главным недостатком не только MongoDB, но самого принципа нереляционных БД является невозможность каскадного удаления и обновления, т.к. документы разных коллекций хранятся отдельно, что требует ручного управления каскадными операциями. Однако в разрабатываемой системе нет таких данных, которые необходимо удалять по ссылке, поэтому MongoDB выбрана как наиболее удобная база данных для хранения информации о фильмах.

## Проектирование приложения

В результате анализа требований и структуры базы данных можно выделить такие подсистемы (см. рис. Рисунок 4.5):

* модуль авторизации;
* пользовательский интерфейс;
* подсистему предоставления данных;
* подсистема выдачи рекомендаций;
* модуль оценивания системы.

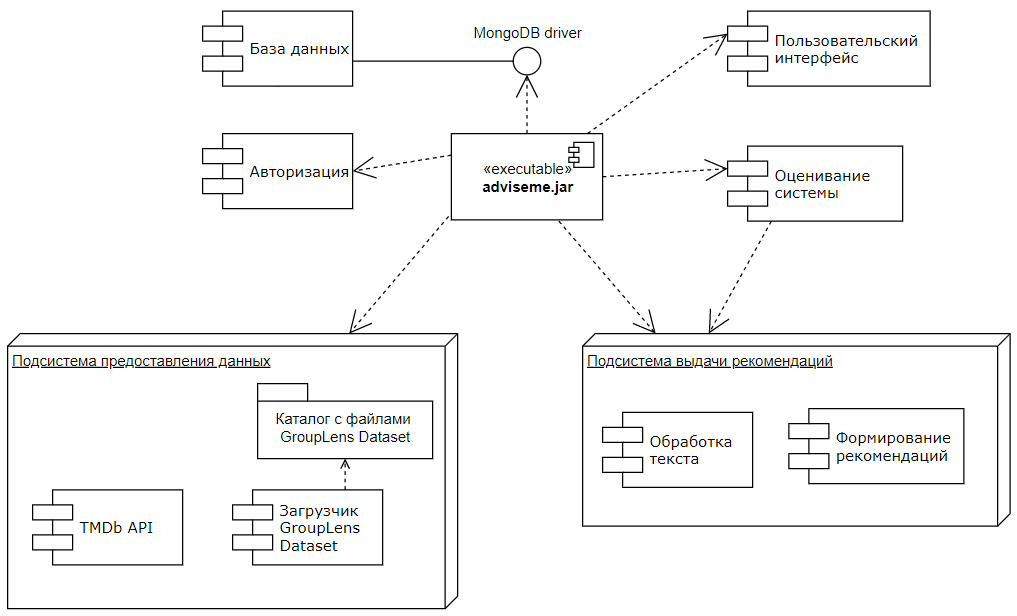


Рисунок 4.5 - Диаграмма компонентов системы

На основе модели базы данных разработана следующая архитектура той части рекомендательного модуля, которая реализует обработку текста (см. рис. 4.6). Данная диаграмма является упрощённой и не включает классы интерфейсов и приватные поля классов.

Класс DocumentAnalysis предоставляет интерфейс доступа к системе обработки текстов. Результатом работы подсистемы является список CompareResult, который содержит информацию о междокументной связи и/или подобии.

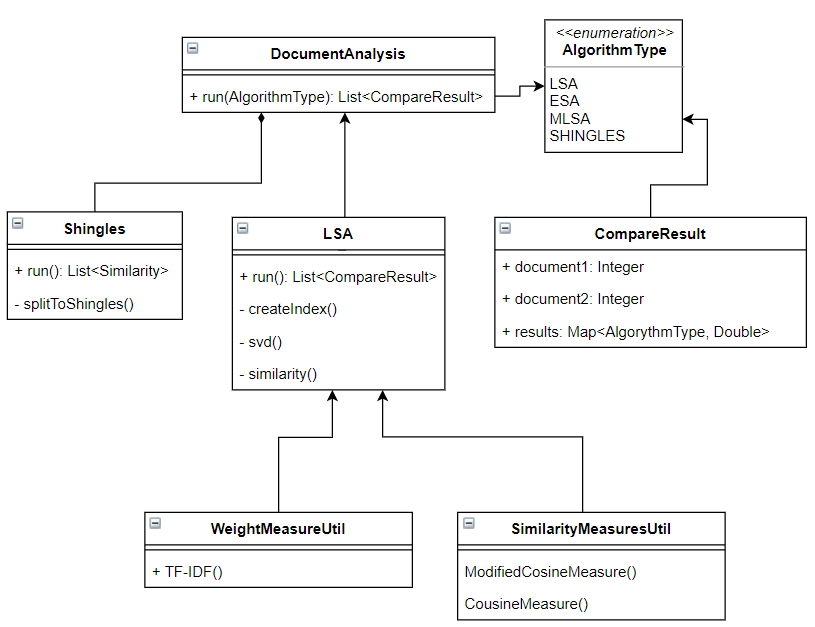


Рисунок 4.6 - Диаграмма классов модуля обработки естественного языка

Исполняемый файл adviseme.jar зависит от модулей, выполняющих логику системы, а также от базы данных. Для реализации приложения выбрана нереляционная база данных MongoDB. Подсистема предоставления данных включает интерфейс доступа к TMDb API, а также загрузчик данных о пользователях из заранее импортированных файлов. Модуль оценивания зависит от алгоритмов, реализованных в рекомендательном модуле. Пользовательский интерфейс позволяет управлять приложением и отображает результаты работы системы.

## Проектирование процесса выдачи рекомендаций фильмов

Анализ современных систем, главной функцией которых является предоставление списков объектов, которые предположительно могут понравиться пользователю, показал, что наибольшую эффективность показывают гибридные рекомендательные системы, которые подразумевают комбинацию подходов коллаборативной и контентной фильтрации [60], а также внедрение неперсональных рекомендаций для решения проблемы холодного старта.

При использовании в системе рекомендации фильмов контентного подхода пересчёт рекомендаций будет происходить редко: только при добавлении нового фильма в коллекцию. При коллаборативном подходе пересчёт рекомендаций должен происходить каждый раз, когда любой из пользователей будет взаимодействовать с объектами.

Поскольку пользователей в системе может быть больше, чем количество фильмов, и их взаимодействие с объектами может происходить регулярно – например, каждый пользователь выполнит несколько действий в течении одного часа. В этом случае нагрузка на систему может быть такой, что потребуется значительное увеличение вычислительных мощностей. Чтобы снизить затраты на покупку дополнительных мощностей, лучше использовать такой режим работы, при котором подсистема будет вычислять рекомендации в заданные промежутки времени, например, раз в день.

На рисунке Рисунок 4.7 изображена диаграмма деятельности, описывающая процесс обновлений данных о фильмах и об их подобии.

Вначале планировщик инициализирует запуск обновления БД. Данное действие предлагается выполнять не чаще раза в день, т.к. более частый запуск не улучшит результатов системы, а лишь даст большую нагрузку, а более редкий запуск не позволит содержать базу данных в актуальном состоянии.

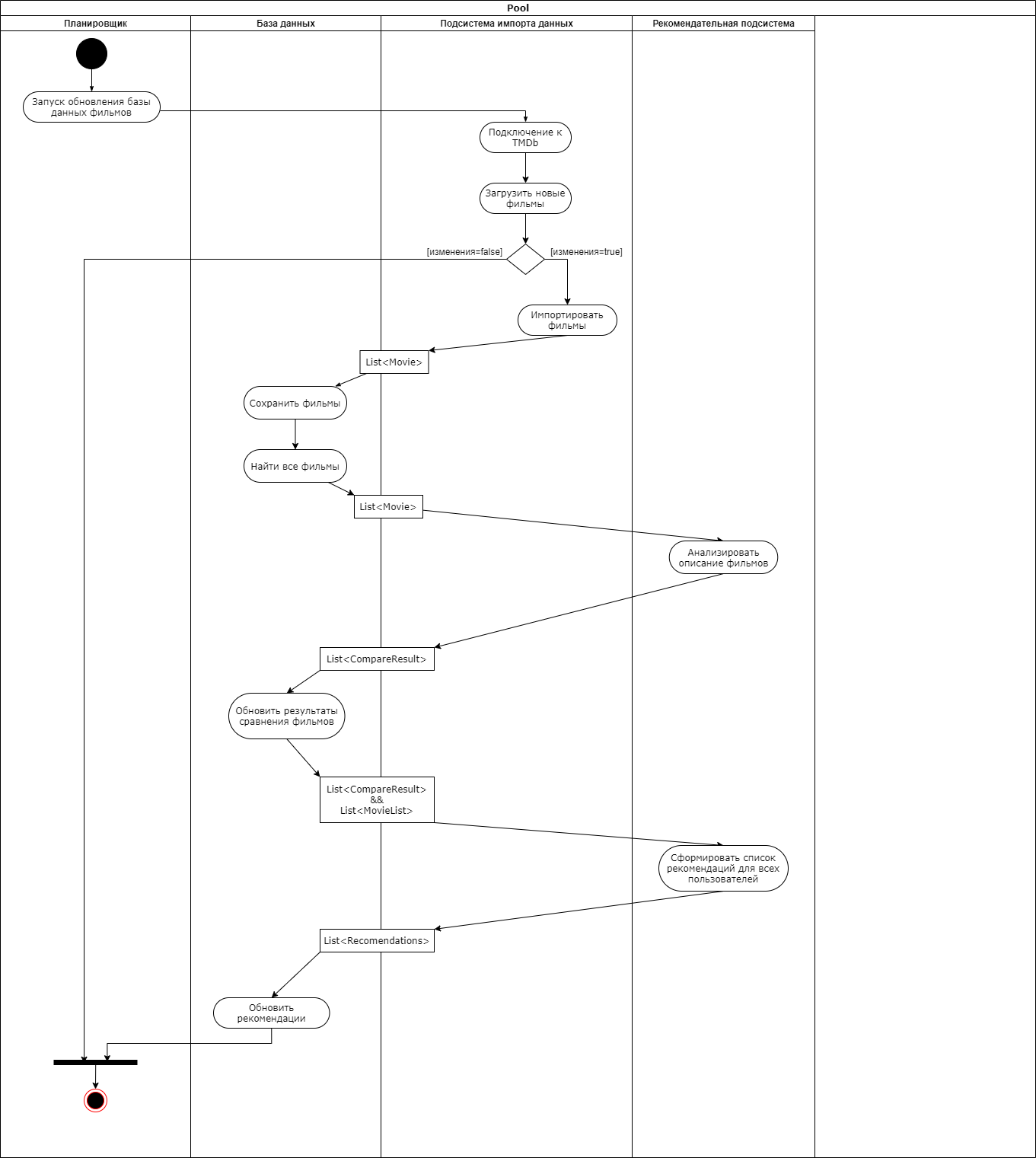


Рисунок 4.7 - Диаграмма деятельности для обработки описаний фильмов

При запуске обновления БД происходит обращение к сервису TMDb с целью поиска новых фильмов. Под новыми фильмами понимаются фильмы, добавленные в базу данных TMDb позже, чем дата последнего фильма, добавленного в систему AdviseMe. При обнаружении новых фильмов необходимо сохранить их в базу данных системы AdviseMe, после чего рекомендательный модуль может начать обработку новопоступивших фильмов.

Под обработкой фильмов рекомендательной системой подразумевается процесс определения подобия фильмов на основе их описания. Для этого система запускает один из алгоритмов (LSA, модифицированный LSA, n-gram) и дожидается обработки результатов.

Вследствие работы алгоритма определения подобия документов составляется список результатов, который сохраняется в базу данных. Далее для каждого пользователя происходит перестройка их списка рекомендаций.

Построение списка рекомендаций представлено на рисунке Рисунок 4.8. Граница в 70% установлена для того, чтобы сбалансировать выдаваемые результаты между объёмом данных и их точностью.

После составления списков рекомендованных фильмов для всех пользователей система обновляет объект пользователя в базу данных и при следующем запросе пользователь сможет получить индивидуальные рекомендации на основе своих действий.

Рассмотрен только случай, когда система получила от пользователя достаточно данных о предпочтениях, чтобы рекомендации получились персональными. Для того, чтобы составлять списки для просмотра для новых или неактивных пользователей, система реализована в режиме гибридизации.

Существует множество подходов к реализации гибридизации [61], таких как переключение, взвешивание, каскадный подход, комбинирование и усиление признаков.

Переключение – выбор рекомендаций по результатам работы разных алгоритмов в зависимости от обстоятельств [62]**.**

Взвешивание – назначение каждому из алгоритмов степени влияния на список рекомендованных объектов [[63](https://iopscience.iop.org/article/10.1088/1742-6596/930/1/012050/pdf)].

Каскадный подход – корректирование списка рекомендованных объектов, составленного одним алгоритмом, в следующем алгоритме.

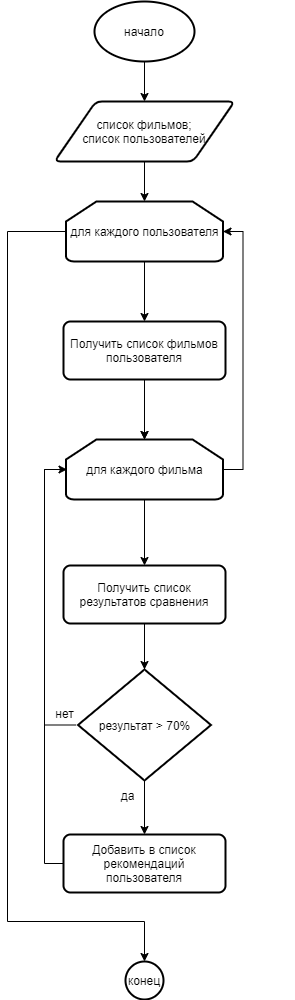


Рисунок 4.8 - Блок-схема алгоритма составления списков рекомендаций

Комбинирование признаков – несколько алгоритмов выдают независимые результаты, которые затем использует дополнительный алгоритм, выявляя скрытые признаки, не обнаруженные отдельными алгоритмами.

Усиление признаков – так же, как и каскадный подход, обрабатывает исходные данные последовательно, но при этом выходные данные одного алгоритма являются входными данными для следующего алгоритма.

Для достижения наибольше эффективности можно комбинировать описанные подходы. Предлагаемая архитектура (алгоритм) разрабатываемой подсистемы представлена на рисунке.

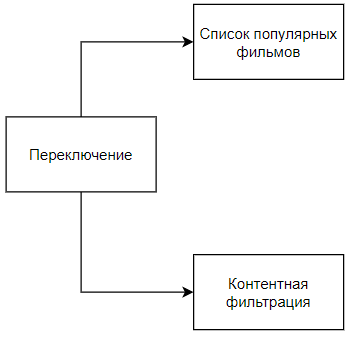


Рисунок 4.9 - Схема гибридизации рекомендательной системы

На первом этапе – переключении решается проблема холодного старта для пользователей. То есть пользователь, у которого слишком мало взаимодействий с объектами системы, будет получать неперсонализированный список рекомендаций, который в данной реализации представлен списком популярных фильмов. Этот список составляется ранжированием фильмов по убыванию рейтинга среди фильмов за последние два месяца. В этом списке содержатся 50фильмов, что повышает вероятность того, что пользователь оценит хотя бы один из них. Как только пользователь ставит положительную оценку хотя бы одному фильму, зарегистрированному в системе, он может начать получать рекомендации. Под положительной оценкой в данной работе понимается значение рейтинга выше или равное 3.5.

Пользователи, у которых можно извлечь данные об их предпочтениях, получают персонализированные рекомендации с помощью алгоритма контентной фильтрации.

В качестве основного алгоритма контентной фильтрации в данной работе используется семантический анализ описаний фильмов, то есть поиск фильмов по схожести их атрибутов – описаний. Таким образом, оценив всего один фильм о животных, пользователь получит список фильмов, в которых явно выражена сюжетная линия про животных. Полученный список ранжируется [[64](https://www.optimism.ru/wiki/%D0%A0%D0%B0%D0%BD%D0%B6%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5:_%D1%87%D1%82%D0%BE_%D1%8D%D1%82%D0%BE_%D1%82%D0%B0%D0%BA%D0%BE%D0%B5._%D0%9A%D0%B0%D0%BA_%D0%B2%D1%8B%D0%B3%D0%BB%D1%8F%D0%B4%D0%B8%D1%82_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_%D1%80%D0%B0%D0%BD%D0%B6%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)] по степени схожести и дате выхода. Больший приоритет имеют фильмы, вышедшие не позже, чем 6 месяцев назад, а также имеющие более высокий средний рейтинг. Из него будет взято не более 200 наиболее подходящих элементов. Данный подход решает проблему холодного старта для фильмов, т.е. даже новые фильмы, которые ещё не оценили пользователи системы, будет появляться в рекомендациях пользователей.

Хотя использование только лишь контентной фильтрации создаёт проблему одинаковых данных, т.е. в случае, если пользователь перестаёт оценивать фильмы, список рекомендаций существенно не изменяется. От этого теряется интерес пользователя к системе. Коллаборативная фильтрация позволяет избежать этой проблемы. Однако, учитывая специфику данной работы, рекомендательная система использует результаты только контентной фильтрации, потому как учёт данных о пользователях, т.е. применение коллаборативной фильтрации, усложняет анализ эффективности модифицированного алгоритма определения схожести текстов.

Часто существующие рекомендательные системы при всей сложности их алгоритмов не учитывают тот факт, что после взаимодействия с конкретным объектом пользователь теряет к нему интерес, если это не товар из категории таких товаров, которые покупаются регулярно. В связи с этим в данной рботе система рекомендаций будет учитывать недавние взаимодействия пользователей с объектами ещё и с той целью, чтобы исключать из списка рекомендаций те объекты, с которыми пользователь взаимодействовал в прошлые 90 дней. Также введены списки не понравившихся фильмов, с помощью которых пользователь может вручную фильтровать фильмы, которые он не хочет видеть в рекомендациях.

## Выводы по разделу

В разделе проанализированы пользовательские и функциональные требования к системе, на основе этих требований составлена модель базы данных и определены основные модули программной системы. По этим данным составлена диаграмма классов.

Графически отображён процесс обработки текстов для определения их подобия и описан процесс выдачи пользовательских рекомендаций, основанный на поиске подобия описаний фильмов.

# Оценка эффективности рекомендательной системы AdviseMe

## Анализ мер для оценивания рекомендательных систем

Качество применяемых рекомендательных алгоритмов можно определить по степени точности или полноте результатов. Выбор меры для оценивания зависит от способа обработки результатов работы алгоритма рекомендаций. Точностью в таких мерах считают отношение количества правильных рекомендованных объектов к общему количеству рекомендованных объектов. За полноту результатов принимают количество объектов, по которым система может дать свою оценку.

Для оценки точности рекомендательных систем применяют статистические меры и меры поддержки принятия решений [64]. Статистические меры (MAE, RMSE, корреляция) определяют точность рекомендательной системы, сравнивая предсказанные системой рейтинги объектов для каждого пользователя и реальные оценки пользователей в тестовой выборке. Меры поддержки принятия решений (Precision, Recall, ROC) выявляют эффективность системы по степени корректно рекомендованных объектов. Таким меры работают с рекомендациями однозначно – объект может быть либо рекомендован, либо не рекомендован.

Разрабатываемая система предназначена для того, чтобы рекомендовать фильмы однозначно, поэтому для оценки эффективности системы и в частности модифицированного алгоритма определения схожести текстов выбрана мера F1, которая использует Precision и Recall.

## Мера F1

F1-score используется для того, чтобы объединить меры Precision (точность) и Recall (полноту) в одну метрику. Результирующее значение упрощает сравнение эффективности алгоритмов, используемых в рекомендательной системе.

Precision (см. формулу 5.1) это отношение релевантных рекомендованных объектов, т.е. количества общих объектов в списках реальных понравившихся пользователям объектов и рекомендованных алгоритмом, к общему количеству рекомендованных объектов.

, (5.1)

где TP – количество объектов, которые понравились пользователю и были рекомендованы системой;

FP – количество объектов, которые не понравились пользователю, но были рекомендованы системой.

Recall (см. формулу 5.2) это отношение релевантных рекомендованных объектов к объему числу релевантных документов.

, (5.2)

где TP – количество объектов, которые понравились пользователю и были рекомендованы системой;

FN – количество объектов, которые понравились пользователю, но не были рекомендованы системой.

Значения Precision и Recall находятся в диапазоне [0, 1], соответственно, чем выше значение какой-либо метрики, тем лучше. Однако невозможно достичь наибольшей точности и при этом сохранить большой объём, поэтому для усреднения эффекта используют меру F1 [65].

В общем случае, когда нет необходимости усиливать точность или полноту, для вычисления меры F1 используют формулу 5.3. В разрабатываемой системе на данном этапе не нужно повышать полноту или точность результатов.

, (5.3)

Благодаря простоте интерпретации и возможности усилить эффект степени полноты мера F1 является наиболее широко применяемой в обработке текстов на естественном языке.

## Исследование эффективности разрабатываемой рекомендательной системы

Оценивание разработанной системы осуществляется с помощью обработки данных от реальных пользователей и результатов работы алгоритмов определения подобия текстов.

Тестовая выборка реальных данных пользователей была сформирована на основе набора данных сервиса Movielens [58], которая позволяет провести интеграцию с базой фильмов TMDb, используемой как основа в разрабатываемой рекомендательной системе.

Последовательность действий при выполнении автоматического оценивания представлена на рисунке 5.1. При создании тестовых списков фильмов реальных оцененных фильмов и рекомендованных алгоритмами системы фильмов используются таблицы TestUserMovie и EvaluationUserMovie соответственно.

Создание документов в коллекции TestUserMovie происходит с помощью импорта данных из файла ratings.csv, который содержит случайные ID пользователя, Movielens ID фильмов, рейтинг, который пользователь поставил фильму и дата оценки. Количество оценённых фильмов каждым пользователем варьируется примерно от 10 до 50. Для того, чтобы сопоставить Movielens ID фильма и TMDb ID фильма, используется файл links.csv, который содержить Movielens ID фильмов, TMDb ID фильмов, IMDb ID фильмов. Таким образом, можно рассчитать коэффициент подобия для фильмов, которые были оценены пользователями из тестовой выборки, и вычислить F1 меру по ним.

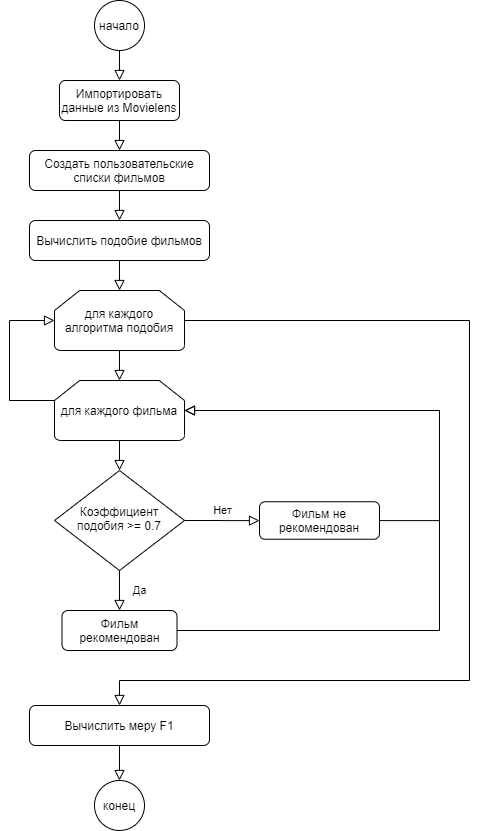


Рисунок . – Алгоритм оценки системы

При оценке эффективности разрабатываемой рекомендательной системы проводилась оценка алгоритмов LSA, модифицированного LSA и алгоритма шинглов. Вычисление меры F1 для каждого из этих алгоритмов дало такие результаты, как в таблице 5.1 и рисунке 5.2. Вычисление подобия проводилось на наборе из 450 фильмов, которые включали все фильмы из тестовой выборки.

Таблица 5.1 – Результат оценки эффективности системы

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Количество тестовых фильмов | | |
| 25 | 100 | 1000 |
| LSA | 0.511 | 0.599 | 0.6 |
| Модифицированный LSA | 0.528 | 0.617 | 0.63 |
| Шинглы | 0.007 | 0.021 | 0.023 |

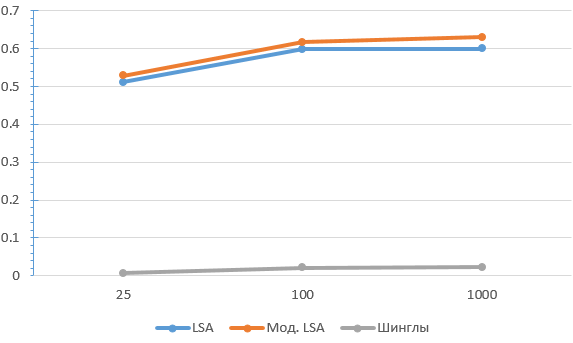


Рисунок 5.2 – Результат оценки эффективности системы

Точность и полнота всех алгоритмов повышается с увеличением размера тестовой выборки, однако очевидно, что строковые (лексические) методы, такие как n-gram (шинглы) не справляется с задачей сравнения коротких документов.

По результатам оценивания рекомендательной системы модифицированный алгоритм LSA показывает наилучшие результаты при усреднённом значении точности и полноты.

## Выводы по разделу

В разделе проведён анализ существующих мер определения эффективности рекомендательных систем, приведено обоснование выбора меры F1 как наиболее подходящей для оценивания разрабатываемой системы. Приведен алгоритм определения эффективности разрабатываемой системы с применением меры F1. Проведены исследования на разных исходных данных. Результаты выполнения алгоритма доказывают эффективность модифицированного алгоритма.

# Охрана труда

## Анализ характеристик и потенциальных опасностей и вредностей объекта исследования

Производственная (рабочая) среда включает в себя все, что окружает человека в процессе трудовой деятельности: техническое оснащение организации, особенности технологических процессов и про­изводства, состояние зданий, строений, сооружений и инженерных коммуникаций, санитарно-гигиеническую и эстетическую обстановку, взаимоотношения в трудовом коллективе, уровень профессионального риска исходя из идентифицированных опасных и вредных производственных факторов и пр.

К вредным факторам производственной среды относят:

* излучения разного вида при использовании видеомониторов на электроннолучевых трубках (мягкое рентгеновское излучение, ультрафиолетовое, видимое, инфракрасное излучения, низко и высокочастотное электромагнитное излучение), электростатические поля;
* механические шумы, связанные с работой принтера, вентиляторов системы охлаждения компьютера и кондиционера, приводов чтения СD-дисков;
* вибрация;
* ионизация воздуха;
* электромагнитное излучение мобильных телефонов;
* выделение в воздух рабочей зоны вредных химических веществ при печати и ксерокопировании документов;
* поглощение из воздуха рабочей зоны работающими кондиционерами отрицательно заряженных аэроионов, а также накопление в них вредных микроорганизмов (микробы, вирусы, грибы, бактерии), которые попадают в воздух рабочих помещений.

К психофизиологическим опасным и вредным производственным факторам относят физические нагрузки (статические и динамические) и нервно-психические пере­грузки (умственное перенапряжение, перенапряжение анализаторов, монотонность труда, эмоциональные перегрузки). Компьютер – это источник нескольких видов излучения.

Так как мониторы с электронно-лучевой трубкой редко используются, то этот вид вредного воздействия на организм человека рассматривать не будем. Вместо этих мониторов сейчас широкое распространение получили мониторы с TFT или LCD экраном, для которых лишь важно размещение в зависимости от оконных проемов для предотвращения бликов.

Электромагнитное поле, производимое блоками питания любых устройств, способствует оседанию пыли и аэрозольных частичек на лице, руках, шее, которые могут вызвать у людей очень чувствительных к подобному влиянию, отрицательные реакции – сухость и аллергию.

Электростатическое поле влияет на ионный состав воздуха, который ухудшает среду в помещении с компьютерами.

Инфракрасное излучение наиболее опасное, так как проникает в ткани и повышает их температуру. Это может вызвать покраснение глаз, раздражение и воспаление.

В помещениях, где используется современное компьютерное оборудование, часто присутствует шум. Источниками шума в таких помещениях могут стать вентиляционные установки, кондиционеры или периферийное оборудование для ПК. Продолжительное действие этих шумов отрицательно сказывается на эмоциональном состоянии персонала. Соответственно ГОСТ в помещениях вычислительных центров уровни звука и эквивалентные уровни звука не должны превышать 50 дб.

Недостаточность освещения приводит к напряжению зрения, ослабляет внимание, приводит к наступлению преждевременной утомленности. Чрезмерно яркое освещение вызывает ослепление, раздражение и резь в глазах. Неправильное направление света на рабочем месте может создавать резкие тени, отблески, дезориентировать работающего. Все эти причины могут привести к несчастному случаю на производстве или профзаболеваниям. Таким образом, перенапряжение зрительного нерва очень сильно зависит от освещения в компьютерных классах.

Кроме того, сидячая работа приводит к снижению мышечной активности – гиподинамии. За недостатком движений происходит снижение потребления кислорода клетками организма, замедляется обмен веществ. Это может привести к развитию атеросклероза, ожирению, а также стать причиной дистрофии миокарда, хронической головной боли, бессонницы, раздражения и др.

Особенность деятельности обучаемых с применением компьютерной техники предопределяет появление психологических опасных факторов – нервно-психической нагрузки, вследствие чего у обучаемого снижается функциональная активность центральной нервной системы, возникают разлады в ее деятельности. Все эти факторы отрицательно влияют на здоровье и способствуют возникновению профессиональных заболеваний:

* кистевой туннельный синдром – сжатие нерва в кистевом канале;
* синдром продолжительной статической нагрузки – мышцы находятся в состоянии постоянной усталости и слабеют;
* мышь-рука – атрофия мышц;
* компьютерная аллергия – аллергия на пыль и вещества, которые выделяются в процессе работы компьютера;
* радиоволновая болезнь – вегето-сосудистая дистония и расстройство центральной нервной системы.

## Выбор и обоснование мероприятий для создания нормальных и безопасных условий труда

Создание благоприятных условий работы и верное эстетическое оформление рабочих мест на производстве играет значительную роль не только для облегчения труда, но и для повышения его привлекательности, положительно влияющей на эффективность труда. Окраска помещений и мебели должна способствовать созданию приемлемых условий для визуального восприятия и хорошего настроения. В служебных помещениях, где выполняется монотонная умственная деятельность, требующая значительного нервного усилия и внимания, окраска должна быть сдержанных тонов - малонасыщенные оттенки холодного зеленого или голубого цветов. При разработке оптимальных условий труда программиста необходимо учитывать освещенность, шум и микроклимат.

Так как работа с компьютерными устройствами требует высокой зрительной сосредоточенности, необходимо обеспечить достаточную освещенность помещения, в котором она будет проводится. Запрещается размещение рабочих мест в подвальных помещениях. В комнате должно присутствовать естественное освещение через световые проемы. Для регулирования уровня освещения проемы должны оборудоваться соответствующими регуляторами (занавески, жалюзи). Для работы в вечернее время необходимо предусмотреть наличие искусственных источников освещения.

Освещение на рабочем месте программиста должно быть такой, чтобы работник мог без напряжения зрения выполнять свою работу. Расчет освещенности рабочего места сводится к выбору системы освещения, определению необходимого числа светильников, их типа и размещения. В помещении работающего на компьютере должно быть естественное и искусственное освещение.

Требования к освещению рабочего места:

* освещенность на поверхности стола в зоне размещения документа должна быть 300-500 л;
* местное освещение не должно создавать бликов на поверхности экрана и увеличивать освещенность экрана более 300 лк;
* яркость светящихся поверхностей (окна, светильники), находящихся в поле зрения, должна быть не более 200 кд/м2;
* яркость бликов на экране монитора не должна превышать 40 кд/м2;
* показатель ослеплённости (по СНиП 23-05-95) для источников общего искусственного освещения в помещениях должен быть не более 20.

Для обеспечения нормативных значений освещенности в помещениях следует проводить чистку стекол оконных проемов и светильников не реже двух раз в год и проводить своевременную замену перегоревших ламп.

Для поддержания оптимального микроклимата помещение должно быть оборудовано кондиционером, системой отопления и приточно-вытяжной вентиляцией. Параметры микроклимата могут меняться в широких пределах, в то время как необходимым условием жизнедеятельности человека является поддержание постоянства температуры тела благодаря свойству терморегуляции, т.е. способности организма регулировать отдачу тепла в окружающую среду.

Установлено, что шум ухудшает условия труда, оказывая вредное воздействие на организм человека. При длительном воздействии шума на человека происходят нежелательные явления: снижается острота зрения, слуха, повышается кровяное давление, понижается внимание. Сильный продолжительный шум может стать причиной функциональных изменений сердечно-сосудистой и нервной систем. Уменьшение шума, проникающего в помещение извне, достигается уплотнением по периметру притворов окон и дверей. Наиболее выраженными звукопоглощающими свойствами обладают волокнисто-пористые материалы: фибролитовые плиты, стекловолокно, минеральная вата, полиуретановый поропласт, пористый поливинилхлорид.

Общий контур заземления здания должен быть выведен через розетку на рабочее место. Заземленные конструкции, находящиеся в помещениях, где размещены рабочие места операторов (батареи отопления, водопроводные трубы, кабели с заземленным открытым экраном), должны быть надежно защищены диэлектрическими щитками или сетками с целью недопущения попадания человека под напряжение.

Площадь, отведенная на одно рабочее место должна составлять не менее 6 кв. м., а объем - не менее 20 куб. м. Конструкция рабочего места должна обеспечивать поддержание оптимальной рабочей позы (то есть такой, которая позволяет работнику выполнять работу с минимальным напряжением тела, и которая позволяет избежать переутомления в ходе и после окончания рабочего процесса).

Рекомендуемая влажность воздуха составляет 50-60%.

Приступая к работе, необходимо позаботиться о том, чтобы комната, в которой предстоит работать, в достаточной степени вентилировалась и была освещена. Время от времени нужно снимать напряжение мышц, для чего необходимо менять положение тела. Каждые 1-2 ч рекомендуется делать перерывы 10-15 мин. Нужно периодически во время работы на несколько секунд закрывать глаза и прикрывать их ладонями. Совершенно недопустимо курение во время работы. Большую роль в профилактике умственного утомления играют положительные эмоции.

Помещение, в котором проходила работа над магистерской диссертацией, соответствует требованиям охраны труда.

### Расчёт необходимой площади окон для бокового естественного освещения

Рассчитать необходимую площадь окон для бокового одностороннего естественного освещения для помещения программистов размерами *L \* B* = *10\*9 м,* высотой *3,8 м*. Высота от уровня рабочей поверхности до верха окна *h1* = *2,4 м*. Здание находится в IV световом поясе. Напротив участка нет затеняющих зданий. Окна ориентированы на запад. Характер зрительной работы соответствует работе IV разряда. Коэффициент отражения потолка равен , отражения стен , пола Расстояние расчётной точки от наружной стены 1 = 6 м, высота рабочей поверхности *hp* = 0,7 м.

Решение. Необходимая площадь окон вычисляется по формуле:

(6.1)

где – нормированное значение коэффициента естественного освещения КЕО;

– коэффициент запаса;

значение световой характеристики окон;

илощадь пола, м2;

– общий коэффициент светопропускания окон;

– коэффициент, который учитывает повьиление КЕО при боковом освещении благодаря свету, отраженному от поверхностей помещения.

коэффициент, учитывающий затенение окон противостоящими зданиями.

Нормированное значение КЕО определяем из выражения:

(6.2)

где – значение КЕО по СНиП II-4-79 для Ш светового пояса (принимаем по приложению 26 = 1,5%);

*m* – коэффициент светового климата;

*с* – коэффициент солнечности климата, зависящий от ориентации окон

по азимуту и вида организации естественного освещения (по приложению 19 принимаем *с* = 0,8).

По СНиП II-4-79 принимаем коэффициент запаса Кз = 1,3 (он определяет периодичность чистки стекол).

Находим отношение глубины помещения В к высоте от уровня рабочей поверхности до верха окна *h1*:

Отношение длины помещения к его глубине:

По СНиП II-4-79 световая характеристика окна

Определяем общий коэффициент светопропускания окон:

(6.3)

где – коэффициент светопропускания материала;

 – коэффициент, учитывающий потери света в переплетах;

 – то же, в несущих конструкциях покрытий;

 – коэффициент, учитывающий светопотери в солнцезащитных устройствах;

– то же, в защитной сетке, устанавливаемой под фонарями.

Поскольку окна изготовлены из двойных деревянных рам, в которые вставлено листовое стекло, то по приложению З1 = 0,8. Для двойных раздельных деревянных рам по приложению 34 = 0,6. Поскольку по условию у нас нет потерь света в несущих конструкциях . Поскольку окна не имеют светозащитных устройств = 1. Для бокового освещения = 1.

Подставляем значения в формулу:

Подсчитаем площадь потолка, стен, пола:

Определяем средневзвешенный коэффициент отражения:

(6.4)

Отношение расстояния расчётной точки от наружной стены к глубине помещения В:

По СНиП II-4-79 определяем коэффициент = 1,4. Поскольку затеняющие здания отсутствуют, коэффициент принимаем равным 1.

Подставляем значения в формулу (6.1) и определяем площадь окон:

Площадь окон 20 . При стандартном размере рамы 1,5\*1,7 м площадь одного окна составит 2,55 и количество окон равно

## Обеспечение пожарной безопасности

Пожарная безопасность на предприятии – это один из важных факторов сохранности рабочей территории и жизни персонала. Для обеспечения безопасности необходимо соблюдать следующие меры.

Запрещено:

* загромождать подступы к средствам пожаротушения;
* применять самодельные плавкие предохранители;
* использовать светильники со снятыми защитными колпаками;
* использовать неисправные электроприборы;
* использовать пожарные краны, огнетушители и другие средства пожаротушения не по прямому назначению.
* устраивать временные электросети;
* оставлять работающие электроприборы без присмотра;

Определение видов и количества огнетушителей следует проводить с учетом физико-химических и пожароопасных свойств горючих веществ, площадей и категории производственных помещений по взрывопожарной опасности, а также класса возможного пожара.

Рабочее помещение, в котором проходила производственная практика, имеет площадь 90 м2. Категория помещения по взрывопожарной и пожарной опасности – В, так как в комнате находятся твердые горючие материалы (бумага, мебель и т.д.). Класс возможного пожара – А. Учитывая минимальную порчу компьютерной техники при тушении пожара, данное помещение желательно оснастить четырьмя углекислотными огнетушителями типа ОУ - 5.

Заключение

В ходе выполнения работы исследованы наиболее популярные подходы к формированию индивидуальных пользовательских рекомендаций и к обработке текстовых объектов. Проанализированы литературные источники с целью нахождения подходов выдачи рекомендации и определения подобия документов, соответствующих задачам разрабатываемой системы. Исследованы различные подходы к получению весовых коэффициентов и определения степени подобия для методов поиска подобия документов. Предложена модификация существующей меры подобия, которая основана на информации о том, что сравниваемые документы имеют небольшую длину.

Список использованных источников

1. Aggarwal, C.C. Recommender Systems: The Textbook. / C.C Aggarwal. – Switzerland: Springer International Publishing, 2016. – 518 p.
2. Singh, R. A Survey on the Generation of Recommender Systems / R. Singh // Information Engineering and Electronic Business. – 2017. – Vol. 3 – P. 26‑35.
3. Sinha, B.B. Evolution of recommender paradigm optimization over time / B.B. Sinha, R.Dhanalakshmi // Journal of King Saud University. – 2019.
4. Hassan M., Hamada M. Genetic Algorithm Approaches for Improving Prediction Accuracy of Multi-criteria Recommender Systems / M. Hassan, M. Hamada // International Journal of Computational Intelligence Systems. – 2018. – Vol. 11 – P. 146-162.

Schafer, B. Collaborative Filtering Recommender Systems. / B. Schafer [et al] // Lecture Notes in Computer Science – 2007. – Vol. 4321 – P. 292-324

1. Zhelezniak, V. Correlation Coefficients and Semantic Textual Similarity. / V. Zhelezniak [et al] // Human Language Technologies. – Vol. 1 – 2019. – P. 951‑962.
2. Recommender system: A Computer Science Comprehensive Exercise [Электронный ресурс] – Режим доступа: <http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/index.html> – (19.05.2020).
3. Arpita, N. Review based on data clustering algorithms / N. Arpita [et al] // Conference: Information & Communication Technologies. – 2013. – P. 298-303.
4. Claypool, M. Combining Content-Based and Collaborative Filters in an Online Newspaper / M. Claypool [et al] // Workshop Recommender Systems: Algorithms and Evaluation. – 1999.
5. Billsus, D. User Modeling for Adaptive News Access / D. Billsus, M. Pazzani // User Modeling and User-Adapted Interaction. – 2000. – Vol. 10 – P. 147‑180.
6. Tran, T. Hybrid Recommender Systems for Electronic Commerce / T. Tran, R. Cohen // Knowledge-Based Electronic Markets, Papers from the AAAI Workshop, Technical Report. – 2000.
7. Pazzani, M. A Framework for Collaborative, Content-Based, and Demographic Filtering, Artificial Intelligence / M. Pazzani // Artificial Intelligence Review. – 1999. – Vol.13 – P. 393-408.
8. Middleton, S.E. Ontological User Profiling in Recommender Systems / S.E. Middleton [et al] // ACM Trans. Information Systems. – Vol. 22 – 2004. – P. 54‑88.
9. Schein, A.I. Methods and Metrics for Cold-Start Recommendations / A.I. Schein [et al] // ACM SIGIR Conf. – 2002.
10. Анатомия рекомендательных систем [Электронный ресурс] – Режим доступа: <https://habr.com/ru/company/lanit/blog/420499/> – (19.05.2020).
11. Ghazanfar, M. A. Fulfilling the Needs of Gray-Sheep Users in Recommender Systems / M. Ghazanfar [et al] // A Clustering Solution – 2011.
12. Gomez-Uribe, C.A. The Netflix recommender system: Algorithms, business value, and innovation. / C.A. Gomez-Uribe, N. Hunt // ACM Trans. Manage. Inf. Syst. – 2015. – Vol.6 – p. 19.
13. Blei, D.M. Latent Dirichlet Allocation / D. M. Blei [et al] // Journal of Machine Learning Research. – 2003. – Vol.3 – P. 993-1022
14. Koren, Y. Matrix factorization techniques for recommender systems. / Y. Koren, R. Bell, C. Volinsky // Computer – 2009. – Vol42 – P. 42-49.
15. Paterek, A. Improving regularized singular value decomposition for collaborative filtering. / A. Paterek // KDDCup. – 2007. – Vol. 7 – P. 39-42.
16. Mnih, A. Probabilistic matrix factorization. / A. Mnih, R. Salakhutdinov // In Advances in Neural Information Processing Systems. – 2007. – P. 1257–1264
17. Гума, С. Н. Анализ методов Data Mining с целью применения в рекомендательной системе / С. Н. Гума, И. А. Коломойцева // Материалы VI Международной научно-технической конференции «Современные информационные технологии в образовании и научных исследованиях» (СИТОНИ-2019). – Донецк: ДонНТУ, 2019. – С. 299-303.
18. String Metric [Электронный ресурс] – Режим доступа: <https://en.wikipedia.org/wiki/String_metric> (19.05.2020).
19. Евклидово пространство [Электронный ресурс] – Режим доступа: shorturl.at/cm015 (19.05.2020).
20. Расстояние городских кварталов [Электронный ресурс] – Режим доступа: shorturl.at/gpILM (дата обращения: 19.05.2020).
21. Расстояние Чебышёва [Электронный ресурс] – Режим доступа: shorturl.at/otR56 (19.05.2020).
22. Chen, H. String Metrics and Word Similarity applied to Information Retrieval. / H. Chen // University of Eastern Finland. – Finland, 2012. – p. 60.
23. Soloshenko, A. Establishing Semantic Similarity of the Cluster Documents and Extracting Key Entities in the Problem of the Semantic Analysis of News Texts. / A. Soloshenko [et al] // Modern Applied Science. – vol. 9. – 2015. – P. 246-268.
24. Word2Vec Tutorial - The Skip-Gram Model [Электронный ресурс] – Режим доступа: <http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/> (19.05.2020).
25. Li, B. Scaling Word2Vec on Big Corpus / B. Li [et al] // Data Science and Engineering. – 2019. – Vol.4 – P. 157-175.
26. Landauer, T. K. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge / Landauer T. K., Dumais S. T // Psychological Review. – 1997. – Vol.104 – P. 211–240.
27. Turney, P.D. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. / P.D. Turney // [Machine Learning: ECML – 2001](https://link.springer.com/book/10.1007/3-540-44795-4). – P. 491-502.
28. Budiu, R. Modeling Information Scent: A Comparison of LSA, PMI and GLSA Similarity Measures on Common Tests and Corpora / R. Budiu, C. Royer, P. Pirolli // Computer-Assisted Information Retrieval Conf. – 2007.
29. Gabrilovich, E. Computing semantic relatedness using wikipedia-based explicit semantic analysis / E. Gabrilovich, S. Markovitch. // IJcAI. – 2007. – vol. 7 – P. 1606-1611.
30. Panchenko, A. Comparison of the Baseline Knowledge-, Corpus-, and Web-based Similarity Measures for Semantic Relations Extraction. / A. Panchenko // Workshop on Geometrical Models of Natural Language Semantics – 2011. – P. 11‑21.
31. Medical Subject Headings [Электронный ресурс] – Режим доступа: <https://www.nlm.nih.gov/mesh/meshhome.html> – (19.05.2020).
32. WordNet A Lexical Database for English [Электронный ресурс] – Режим доступа: <https://wordnet.princeton.edu/> – (19.05.2020).
33. Hyponymy and hypernymy [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Hyponymy\_and\_hypernymy – (11.06.2020)
34. Hirst, G. Lexical Chains as Representations of Context for the Detection and Correction of Malapropisms / G. Hirst, D. St-Onge. // WordNet: An Electronic Lexical Database. – 1995. – 305 p.
35. Leacock, C. Combining Local Context and WordNet Similarity for Word Sense Identification / C. Leacock, M. Chodorow // WordNet: An electronic lexical database – 1998. – Vol.49 – P. 265-283.
36. Resnik, P. Using Information Content to Evaluate Semantic Similarity in a Taxonomy / P.Resnik // International Joint Conference for Artificial Intelligence (IJCAI-95) – 1995. – P. 448-453.
37. Altszyler, E. Corpus specificity in LSA and Word2vec: the role of out-of-domain documents / E. Altszyler [et al] – 2007.
38. Altszyler, E. Comparative study of LSA vs Word2vec embeddings in small corpora: a case study in dreams database / E. Altszyler, M. Sigman, S. Riberio, D. F. Slezak. – 2017.
39. Liu, C. On the Connections between Explicit Semantic Analysis and Latent Semantic Analysis. / C. Liu, Yi-Min Wang // CIKM. – 2012.
40. Stefanovič, P. The N-Grams Based Text Similarity Detection Approach Using Self-Organizing Maps and Similarity Measures / P. Stefanovič, O. Kurasova, R. Štrimaitis. // Applied Sciences – 2019.
41. Gomaa, W. A Survey of Text Similarity Approaches / W. Gomaa, A. Fahmy // International Journal of Computer Applications. – 2013. – Vol.68. – P.13‑18.
42. Соловьев А. Н. Использование латентно‑семантического анализа в исследованиях и моделировании когнитивного развития детей / А. Соловьёв – 2013.
43. Deerwester, S. Indexing by Latent Semantic Analysis. / S. Deerwester [et al] // [Journal of the American Society for Information Science](https://asistdl.onlinelibrary.wiley.com/journal/10974571). – 1990. – vol.41.
44. Suriati, M. Weighted hybrid technique for recommender system / M. Suriati, T. Dwiastuti // International Conference on Information and Communication Technology. – 2017.
45. Bullinaria, John & Levy, Joseph. Extracting semantic representations from word co-occurrence statistics: A computational study. Behavior research methods. 2007
46. Eckart, C., Young, G. The approximation of one matrix by another of lower rank. Psychometrika 1, 211–218 (1936).
47. Кузнецов А.В. Применения инструментов text mining для анализа средневековых латиноязычных текстов: предварительная обработка текстов. / А.В. Кузнецов // [Международная научная конференция ЕНО](https://esa-conference.ru/journal/58ya-mezhdunarodnaya-nauchnaya-konferentsiya-eno/). – 2019. – № 11. – С.68-71
48. Касевич В.Б. Элементы общей лингвистики. — М.: Наука, 1977. – 183 c.
49. Jurafsky, D. Speech and Language Processing / D. Jurafsky, J.H.Martin // – Prentice Hall, 2000. – 1032 p.
50. Jones, K. S. A statistical interpretation of term specificity and its application in retrieval / K. S. Jones // Journal of Documentation – 1972. – Vol.4 – P. 493-502
51. Townsend, A. Continuous analogues of matrix factorizations / A. Townsend, L. N. Trefethen // Proc. – 2015.
52. Kiela, D. A Systematic Study of Semantic Vector Space Model Parameters / D. Kiela, S. Clark // Workshop on Continuous Vector Space Models and their Compositionality. – 2014. – P.21-30.
53. Booch, G. Unified Modeling Language User Guide / G. Booch [et al] Addison-Wesley Professional, 2005. – 512 p.
54. Harper, F.M. The MovieLens Datasets: History and Context / F.M. Harper, J. A. Konstan // ACM Transactions on Interactive Intelligent Systems (TiiS). – 2015. – Vol.5. – 19 p.
55. Fang, Z. Hybrid recommender system based on personal behavior mining / Z. Fang, L. Zhang, K. Chen – 2016. – 15 p.
56. Burke R. Hybrid web recommender systems // The adaptive web / Lecture Notes In Computer Science. – 2007. – Vol. 4321. – P. 377-408
57. Tejeda-Lorente Á. A quality based recommender system to disseminate information in a university digital library / Tejeda-Lorente Á., Porcel C., Peis E., Sanz R., Herrera-Viedma E // Inf. Sci. – 2014. – Vol. 261. – P. 52-69.
58. Java | Oracle [Электронный ресурс] – Режим доступа: https://www.java.com/en/ – (11.06.2020)
59. The database for modern applications [Электронный ресурс] – Режим доступа: https://www.mongodb.com/ – (11.06.2020)
60. Castellano, G. Web Personalization in Intelligent Environments / G. Castellano, L. C. Jain, A. M. Fanelli – Berlin: Springer. – 2009. – 141 p.
61. Chinchor, N. MUC-4 Evaluation metrics / N. Chinchor // Proc. Fourth Messages Understanding Conf. – 1992. – P. 22-29.
62. Интернет-сайт для рекомендации фильмов [Электронный ресурс] – Режим доступа: http://masters.donntu.org/2019/fknt/guma/library/article1.htm – (11.06.2020)
63. Гума, С. Н. Модификация алгоритма латентно-семантического анализа для сравнения коротких текстов / С. Н. Гума, И. А. Коломойцева. //Материалы студенческой секции X Международной научно-технической конференции «Информатика, управляющие системы, математическое и компьютерное моделирование» (ИУСМКМ-2020). – Донецк: ДонНТУ, 2020.

Приложение А  
Программный код

//LatentSemanticAnalysis.java

public class LatentSemanticAnalysis extends NaturalLanguageProcessing {

SVD svdService;

ThreadPoolTaskExecutor threadPoolExecutor;

private final CosineSimilarity cosineSimilarity;

private final ModifiedCosineSimilarity modifiedCosineSimilarity;

protected LatentSemanticAnalysis(MovieToNLPConverter movieToNlpConverter, DocumentStatsToArrayConverter weightStructureConverter, BagOfWords wordOccurrenceMatrix, WeightMeasure weightMeasureService, @Qualifier("lsaCalculationsThreadPool") ThreadPoolTaskExecutor threadPoolExecutor, CosineSimilarity cosineSimilarity, ModifiedCosineSimilarity modifiedCosineSimilarity, CompareResultRepository compareResultRepository) {

super(movieToNlpConverter, weightStructureConverter, wordOccurrenceMatrix, weightMeasureService, compareResultRepository);

this.threadPoolExecutor = threadPoolExecutor;

this.cosineSimilarity = cosineSimilarity;

this.modifiedCosineSimilarity = modifiedCosineSimilarity;

this.svdService = new SVD();

}

@Async("algorithmsThreadPool")

@Override

public CompletableFuture<Set<CompareResult>> run() {

if(stemmedWords.isEmpty()) {

this.init(new HashSet<>());

}

AtomicLong timer = new AtomicLong(new Date().getTime());

Map<String, List<BagOfWords.WordFrequency>> bagOfWords = wordOccurrenceMatrix.get(stemmedWords);

stemmedWords.clear();

// calculate tf-idf matrix for words-documents

List<DocumentStats> wordDocumentMatrix = this.weightMeasureService.calculateWeight(bagOfWords);

bagOfWords.clear();

// sort by docID

wordDocumentMatrix.sort(Comparator.comparingInt(DocumentStats::getDocumentId));

// group by word => create word vectors

Map<String, List<DocumentStats>> groupByWord = wordDocumentMatrix.stream().collect(Collectors.groupingBy(DocumentStats::getWord));

log.info("Step1 -- " + (new Date().getTime() - timer.get()));

timer.set(new Date().getTime());

// perform svd

double[][] tfIdfArray = this.weightStructureConverter.convert(groupByWord);

int rank = Math.min(tfIdfArray[0].length, 55); // rank cannot be bigger than amount of documents

rank = Math.min(rank, tfIdfArray.length);

double[][] solved = svdService.solve(tfIdfArray, rank);

log.info("Step2 -- " + (new Date().getTime() - timer.get()));

timer.set(new Date().getTime());

// convert again to get values for corresponding documents

List<DocumentStats> wordVector;

List<List<DocumentStats>> matrix = new ArrayList<>(groupByWord.values());

for (int i = 0; i < groupByWord.size(); i++) { // word vector

wordVector = matrix.get(i);

for (int j = 0; j < wordVector.size(); j++) {

wordVector.get(j).setValue(solved[i][j]);

}

}

groupByWord.clear();

// calculate similarity matrix with cosine measure

Map<Integer, List<DocumentStats>> groupByDocument = matrix.stream().flatMap(List::stream).collect(Collectors.groupingBy(DocumentStats::getDocumentId));

ArrayList<List<DocumentStats>> documents = new ArrayList<>(groupByDocument.values());

matrix.clear();

wordDocumentMatrix.clear();

groupByDocument.clear();

List<DocumentStats> documentVector1;

List<CompletableFuture<List<CompareResultHelper>>> compareResult = new ArrayList<>();

for (int i = 0; i < documents.size(); i++) {

documentVector1 = documents.get(i);

compareResult.add(this.getCompareResultsForDocument(documentVector1, documents, i));

}

Set<CompareResult> results = compareResult.stream()

.map(CompletableFuture::join)

.flatMap(List::stream)

.map(crh -> new CompareResult(crh.getMovieId\_1(), crh.getMovieId\_2(), crh.getResult\_lsa(), crh.getResult\_mlsa(), null))

.collect(Collectors.toSet());

log.info("Step3 -- " + (new Date().getTime() - timer.get()));

compareResult.clear();

return CompletableFuture.completedFuture(results);

}

private CompletableFuture<List<CompareResultHelper>> getCompareResultsForDocument(List<DocumentStats> documentVector1, ArrayList<List<DocumentStats>> documents, int i) {

//log.debug("document iteration " + i);

return CompletableFuture.supplyAsync(() -> {

List<CompareResultHelper> result = new ArrayList<>();

List<DocumentStats> documentVector2;

Integer documentId2;

Integer documentId1 = documentVector1.get(0).getDocumentId();

//double[] presenceVector = new double[presenceMatrix.length];

for (int j = 0; j < documents.size(); j++) {

if (i != j) {

documentVector2 = documents.get(j);

documentId2 = documentVector2.get(0).getDocumentId();

Double simLsa = this.cosineSimilarity.findSimilarity(documentVector1, documentVector2);

Double simMlsa = this.modifiedCosineSimilarity.findSimilarity(documentVector1, documentVector2);

result.add(new CompareResultHelper(documentId1, documentId2, simLsa, simMlsa));

}

}

return result;

}, threadPoolExecutor);

}

}

//Shingles.java

public class Shingles extends NaturalLanguageProcessing {

private final int NGRAM\_SIZE = 4;

JaccardSimilarity jaccardSimilarity;

protected Shingles(MovieToNLPConverter movieToNlpConverter, DocumentStatsToArrayConverter weightStructureConverter, WordOccurrenceMatrix wordOccurrenceMatrix, WeightMeasure weightMeasureService, CompareResultRepository compareResultRepository, JaccardSimilarity jaccardSimilarity) {

super(movieToNlpConverter, weightStructureConverter, wordOccurrenceMatrix, weightMeasureService, compareResultRepository);

this.jaccardSimilarity = jaccardSimilarity;

}

@Async("algorithmsThreadPool")

@Override

public CompletableFuture<Set<CompareResult>> run() {

if (this.stemmedWords == null || stemmedWords.isEmpty()) {

this.init(new HashSet<>());

}

Map<Integer, List<String>> documentsByNgrams = new HashMap<>();

for (Map.Entry<Integer, List<String>> document : this.stemmedWords.entrySet()) {

ArrayList<String> ngrams = Ngrams.ngrams(new ArrayList<>(document.getValue()), NGRAM\_SIZE);

documentsByNgrams.put(document.getKey(), ngrams);

}

this.stemmedWords.clear();

Set<CompareResult> results = this.compareDocuments(documentsByNgrams);

return CompletableFuture.completedFuture(results);

}

private Set<CompareResult> compareDocuments(Map<Integer, List<String>> documentsByNgrams) {

Set<Map.Entry<Integer, List<String>>> entries = documentsByNgrams.entrySet();

Map<CompareResult.CompareId, CompareResult> compareResults = new HashMap<>();

for (Map.Entry<Integer, List<String>> entry1 : entries) {

Integer documentId = entry1.getKey();

for (Map.Entry<Integer, List<String>> entry2 : entries) {

if (entry1 != entry2) {

CompareResult.CompareId compareId = new CompareResult.CompareId(documentId, entry2.getKey());

if (!compareResults.containsKey(compareId)) {

double sim = this.getSimilarity(entry1.getValue(), entry2.getValue());

Map<AlgorithmType, Double> result = new HashMap<>();

result.put(AlgorithmType.NGRAM, sim);

compareResults.put(compareId, new CompareResult(compareId, result));

}

}

}

}

return new HashSet<>(compareResults.values());

}

private double getSimilarity(final List<String> document1, final List<String> document2) {

List<String> tempDoc1 = new ArrayList<>(document1);

List<String> tempDoc2 = new ArrayList<>(document2);

int all = tempDoc1.size() + tempDoc2.size();

tempDoc1.retainAll(tempDoc2);

int common = tempDoc1.size();

return this.jaccardSimilarity.findSimilarity(common, all);

}

}

//Evaluation.java

public class Evaluation {

private final Double SIMILARITY\_LIMIT = 0.7;

TestUserMovieRepository evaluationRepository;

EvaluationUserMovieRepository evaluationUserMovieRepository;

MovielensImporter movielensImporter;

CompareResultRepository compareResultRepository;

public Evaluation(TestUserMovieRepository evaluationRepository, EvaluationUserMovieRepository evaluationUserMovieRepository, MovielensImporter movielensImporter, CompareResultRepository compareResultRepository) {

this.evaluationRepository = evaluationRepository;

this.evaluationUserMovieRepository = evaluationUserMovieRepository;

this.movielensImporter = movielensImporter;

this.compareResultRepository = compareResultRepository;

}

List<TestUserMovie> realUserMovies;

List<EvaluationUserMovie> evaluationMovies;

public void init() {

realUserMovies = this.importEvaluationMovies();

evaluationMovies = this.selectTestMovies(realUserMovies);

}

public EvaluationResult compare() {

log.info("Comparing...");

Double f1Score\_lsa = this.getF1Score(AlgorithmType.LSA);

Double f1Score\_mlsa = this.getF1Score(AlgorithmType.MLSA);

Double f1Score\_ngram = this.getF1Score(AlgorithmType.NGRAM);

return new EvaluationResult(f1Score\_lsa, f1Score\_mlsa, f1Score\_ngram);

}

private List<EvaluationUserMovie> selectTestMovies(List<TestUserMovie> evaluationMovies) {

List<EvaluationUserMovie> result = this.evaluationUserMovieRepository.findAll();

if(result.isEmpty()) {

log.info("Calculating test dataset");

//key - movieId, value - users that rated this movie

Map<Integer, Set<Integer>> movies = evaluationMovies.stream().collect(Collectors.groupingBy(TestUserMovie::getMovieId, mapping(TestUserMovie::getUserId, toSet())));

List<CompareResult> byMovieIds = this.compareResultRepository.findByMovieIds(movies.keySet());

for (CompareResult compareResult : byMovieIds) {

Integer movieId1 = compareResult.getIdPair().getMovieId1();

Set<Integer> userIds = movies.get(movieId1);

if(userIds != null) {

for (Integer userId : userIds) {

//lsa

Double simResult = compareResult.getResults().get(AlgorithmType.LSA);

UserMovieStatus status = simResult >= SIMILARITY\_LIMIT ? UserMovieStatus.RECOMMENDED : UserMovieStatus.NOT\_RECOMMENDED;

result.add(new EvaluationUserMovie(userId, movieId1, status, AlgorithmType.LSA));

//mlsa

simResult = compareResult.getResults().get(AlgorithmType.MLSA);

status = simResult >= SIMILARITY\_LIMIT ? UserMovieStatus.RECOMMENDED : UserMovieStatus.NOT\_RECOMMENDED;

result.add(new EvaluationUserMovie(userId, movieId1, status, AlgorithmType.MLSA));

}

}

Integer movieId2 = compareResult.getIdPair().getMovieId2();

userIds = movies.get(movieId2);

if( userIds != null) {

for (Integer userId : userIds) {

//lsa

Double simResult = compareResult.getResults().get(AlgorithmType.LSA);

UserMovieStatus status = simResult >= SIMILARITY\_LIMIT ? UserMovieStatus.RECOMMENDED : UserMovieStatus.NOT\_RECOMMENDED;

result.add(new EvaluationUserMovie(userId, movieId2, status, AlgorithmType.LSA));

//mlsa

simResult = compareResult.getResults().get(AlgorithmType.MLSA);

status = simResult >= SIMILARITY\_LIMIT ? UserMovieStatus.RECOMMENDED : UserMovieStatus.NOT\_RECOMMENDED;

result.add(new EvaluationUserMovie(userId, movieId2, status, AlgorithmType.MLSA));

}

}

}

this.evaluationUserMovieRepository.saveAll(result);

}

return result;

}

private List<TestUserMovie> importEvaluationMovies() {

List<TestUserMovie> result = this.evaluationRepository.findAll();

if(result.isEmpty()) {

log.info("Importing movielens dataset");

this.movielensImporter.importTestData("movielens");

return this.evaluationRepository.findAll();

}

return result;

}

private Double getF1Score(AlgorithmType algorithmType) {

List<EvaluationUserMovie> algorithmList = new ArrayList<>(evaluationMovies);

algorithmList.removeIf(tum -> tum.getAlgorithmType() != algorithmType);

Set<Integer> recommendedIds = algorithmList.stream().filter(tum -> tum.getStatus() == UserMovieStatus.RECOMMENDED).map(EvaluationUserMovie::getMovieId).collect(Collectors.toSet());

Set<Integer> notRecommendedIds = algorithmList.stream().filter(tum -> tum.getStatus() == UserMovieStatus.NOT\_RECOMMENDED).map(EvaluationUserMovie::getMovieId).collect(Collectors.toSet());;

Set<Integer> likedIds = this.realUserMovies.stream().filter(eum -> eum.getStatus() == UserMovieStatus.LIKED).map(TestUserMovie::getMovieId).collect(toSet());

recommendedIds.retainAll(likedIds);

notRecommendedIds.retainAll(likedIds);

double recommendedCount = recommendedIds.size();

double recommendedLikedCount = recommendedIds.size();

double notRecommendedLikedCount = notRecommendedIds.size();

if(recommendedLikedCount == 0.0 && notRecommendedLikedCount == 0.0) {

throw new IllegalStateException("Fix your lists!");

}

Double precision = recommendedLikedCount / recommendedCount;

Double recall = recommendedLikedCount / (recommendedLikedCount + notRecommendedLikedCount);

Double result = (2 \* (precision \* recall)) / (precision + recall);

return result;

}

}

//ModifiedCosineSimilarity.java

public class ModifiedCosineSimilarity {

private final double WEIGHT = 1.6;

public Double findSimilarity(List<DocumentStats> document1, List<DocumentStats> document2) {

Set<String> wordsDocument1 = document1.stream().map(DocumentStats::getWord).collect(Collectors.toSet());

Set<String> wordsDocument2 = document2.stream().map(DocumentStats::getWord).collect(Collectors.toSet());

Set<String> commonWords = this.getCommonWords(

wordsDocument1,

wordsDocument2

);

final double dotProduct = dot(

document1.stream().collect(Collectors.toMap(DocumentStats::getWord, DocumentStats::getValue)),

document2.stream().collect(Collectors.toMap(DocumentStats::getWord, DocumentStats::getValue)),

commonWords);

List<Double> valuesDocument1 = document1.stream().map(DocumentStats::getValue).collect(Collectors.toList());

List<Double> valuesDocument2 = document2.stream().map(DocumentStats::getValue).collect(Collectors.toList());

double d1 = 0.0d;

double d2 = 0.0d;

double vectorValue1;

double vectorValue2;

for (int i = 0; i < valuesDocument1.size(); i++) {

vectorValue1 = valuesDocument1.get(i);

vectorValue2 = valuesDocument2.get(i);

d1 += Math.pow(vectorValue1, 2);

d2 += Math.pow(vectorValue2, 2);

}

double cosineSimilarity;

if(d1 <= 0 || d2 <= 0) {

return 0.0;

}

double v = Math.sqrt(d1) \* Math.sqrt(d2) \* Math.sqrt(WEIGHT);

cosineSimilarity = dotProduct / v;

return cosineSimilarity;

}

/\*\*

\* Computes the dot product of two vectors. It ignores remaining elements. It means

\* that if a vector is longer than other, then a smaller part of it will be used to compute

\* the dot product.

\*/

private double dot(Map<String, Double> valuesFromDocument1,

Map<String, Double> valuesFromDocument2,

Set<String> commonWords) {

double dotProduct = 0.0;

for (String commonWord : commonWords) {

Double aDouble = valuesFromDocument1.get(commonWord);

Double aDouble1 = valuesFromDocument2.get(commonWord);

if (aDouble < 0) {

aDouble = 0.0;

}

if (aDouble1 < 0) {

aDouble1 = 0.0;

}

double weight = 0.7;

if(aDouble > 0 && aDouble1 > 0) {

weight = WEIGHT;

}

dotProduct += aDouble \* aDouble1 \* weight;

}

return dotProduct;

}

private Set<String> getCommonWords(Set<String> wordsFromDocument1,

Set<String> wordsFromDocument2) {

return wordsFromDocument1.stream()

.distinct()

.filter(wordsFromDocument2::contains)

.collect(Collectors.toSet());

}

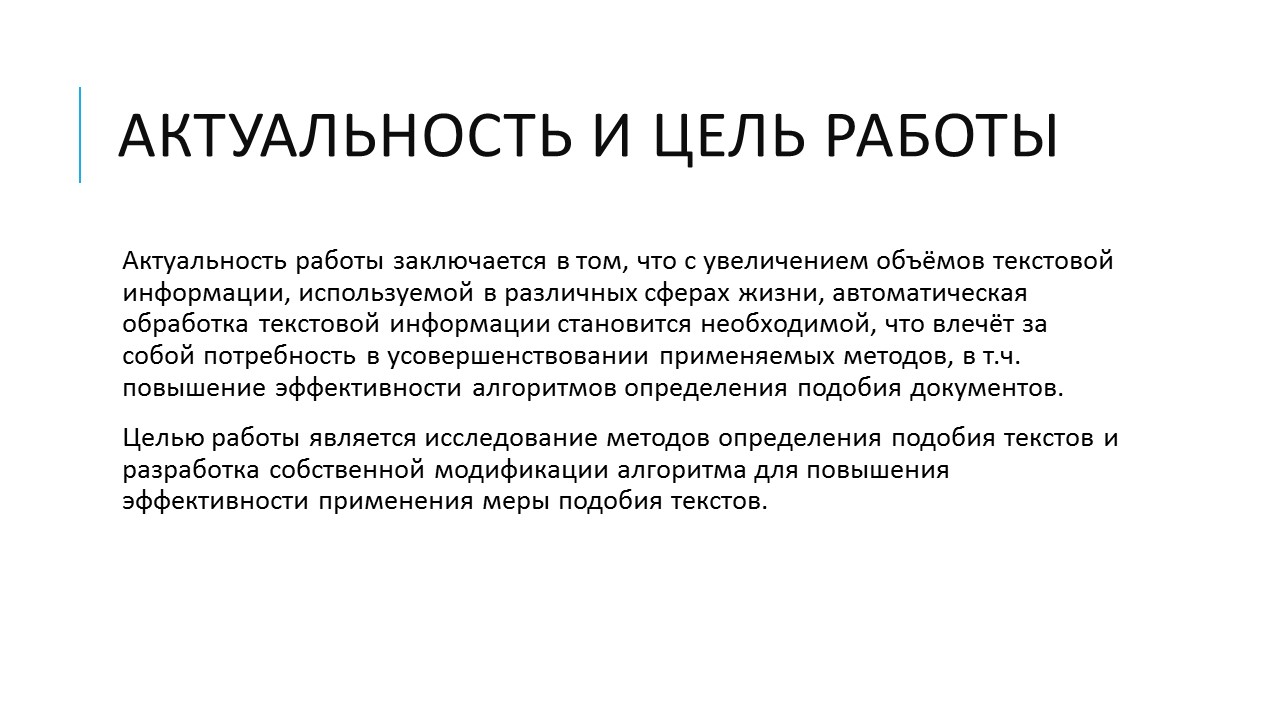
}

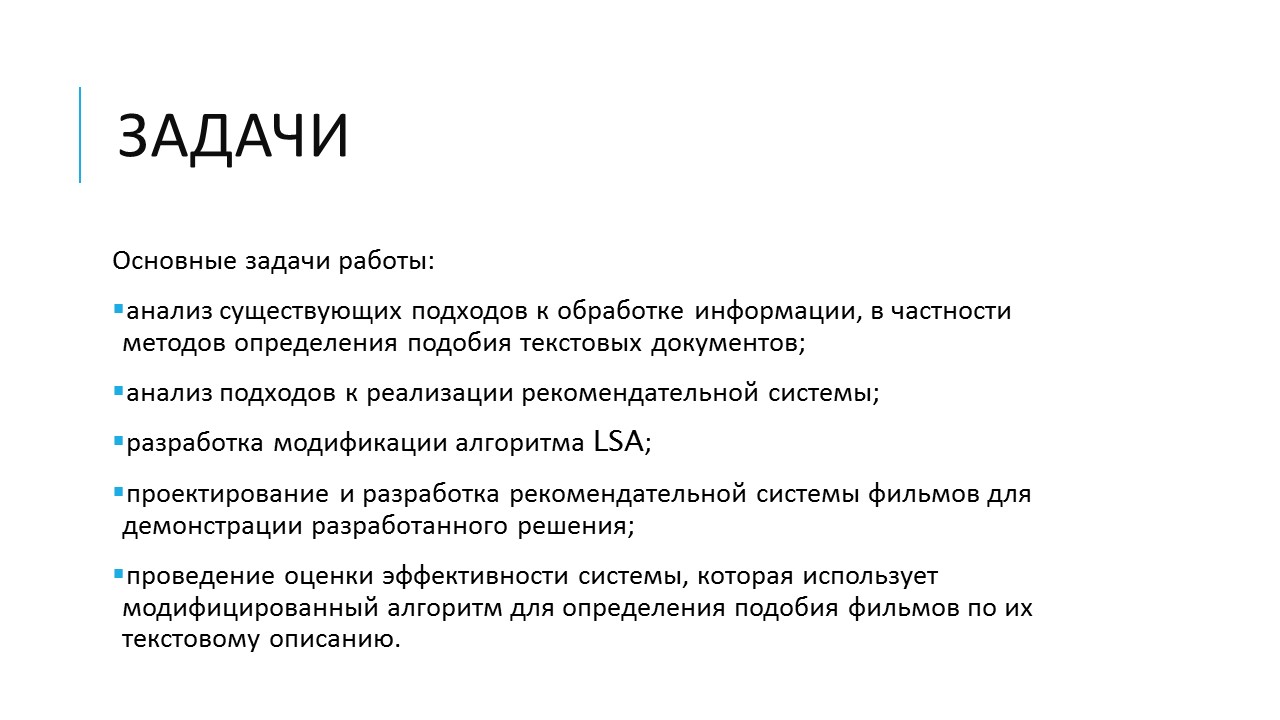
//UserService.java

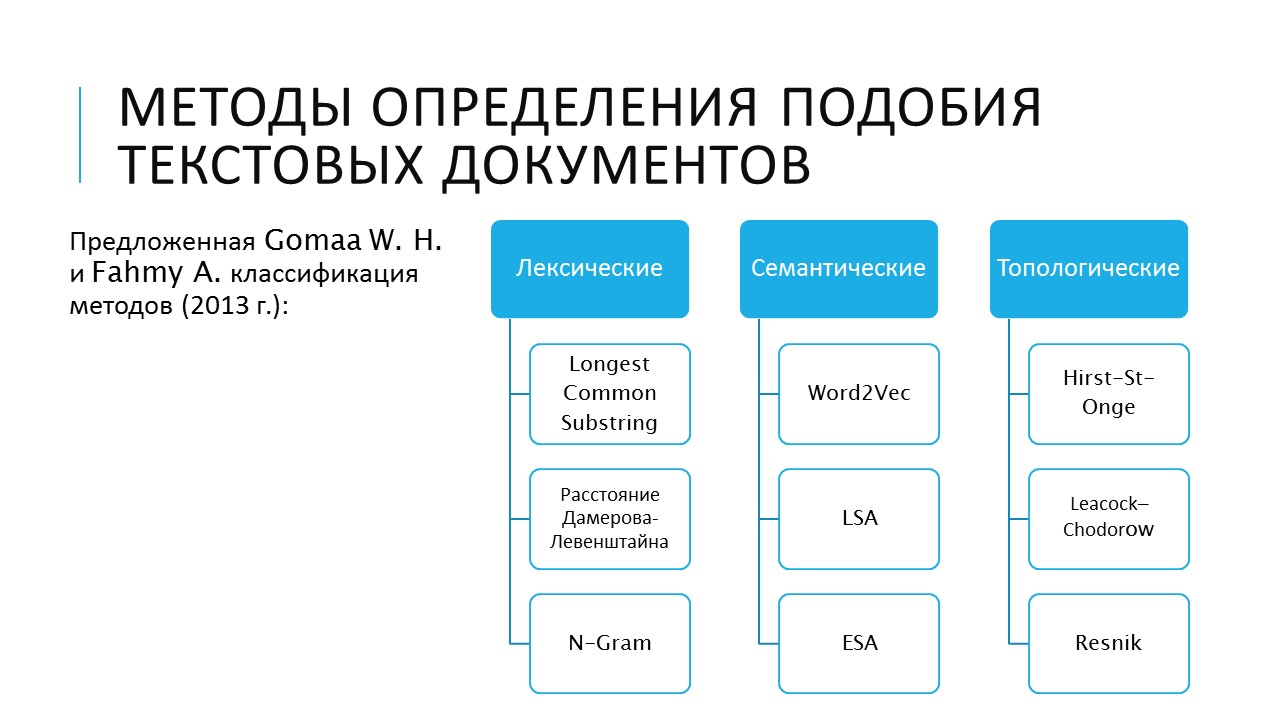
**public class** UserService {  
  
 @Value(**"${rating.limit}"**)  
 Double ratingLimit;  
  
 UserRepository userRepository;  
 UserMovieRepository userMovieRepository;  
 MovieService movieService;  
  
 **public** UserService(UserRepository userRepository, UserMovieRepository userMovieRepository, MovieService movieService) {  
  
 **this**.userRepository = userRepository;  
 **this**.userMovieRepository = userMovieRepository;  
 **this**.movieService = movieService;  
 }  
  
 **public** User getUser(String id) {  
 **return this**.userRepository.findById(id).orElseThrow(() -> **new** EntityNotFoundException(**"User"**));  
 }  
  
 **void** addMovieRating(String userId, String movieId, Double rating) {  
  
 UserMovieStatus type;  
 **if**(rating > ratingLimit) {  
 type = UserMovieStatus.LIKED;  
 } **else** {  
 type = UserMovieStatus.DISLIKED;  
 }  
  
 UserMovie userMovie = **new** UserMovie(userId, movieId, type, rating, Instant.now().toEpochMilli());  
 **this**.userMovieRepository.save(userMovie);  
  
 User user = **this**.userRepository.findById(userId).orElseThrow(() -> **new** EntityNotFoundException(**"User"**));  
 user.getMovies().add(userMovie);  
 **this**.userRepository.save(user);  
 }  
  
 List<UserMovieDtoOut> getMovies(UserMovieStatus userMovieStatus, String userId) {  
  
 User user = **this**.userRepository.findById(userId).orElseThrow(() -> **new** EntityNotFoundException(**"User"**));  
  
 Map<String, UserMovie> userMovies = user.getMovies().stream().filter(m -> m.getStatus() == userMovieStatus).collect(Collectors.toMap(UserMovie::getMovieId, Function.identity()));  
  
 List<Movie> movies = **this**.movieService.getMovies(userMovies.keySet());  
  
 List<UserMovieDtoOut> result = **new** ArrayList<>();  
 NumberFormat nf = NumberFormat.getInstance();  
 **for** (Movie movie : movies) {  
 Double userRating = userMovies.get(movie.getId()).getRating();  
 Long userRatingDateLong = userMovies.get(movie.getId()).getDate();  
 LocalDate userRatingDate = Instant.ofEpochMilli(userRatingDateLong).atZone(ZoneId.of(**"UTC"**)).toLocalDate();  
 nf.setMaximumFractionDigits(1);  
 String formattedAvgRating = nf.format(movie.getVoteAverage().doubleValue() / 2);  
 result.add(**new** UserMovieDtoOut(movie.getId(), movie.getTitle(), movie.getOverview(), formattedAvgRating, userRating, userRatingDate));  
 }  
  
 **return** result;  
 }  
}

Приложение Б  
Графическая часть

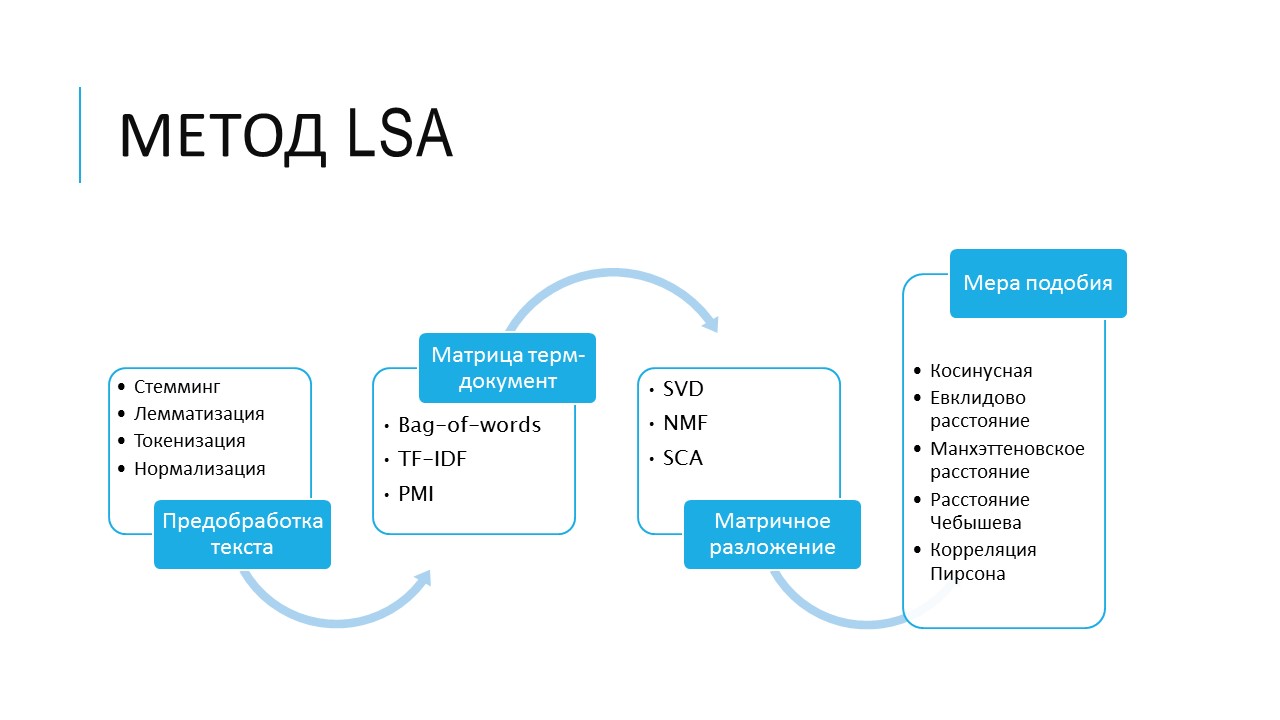


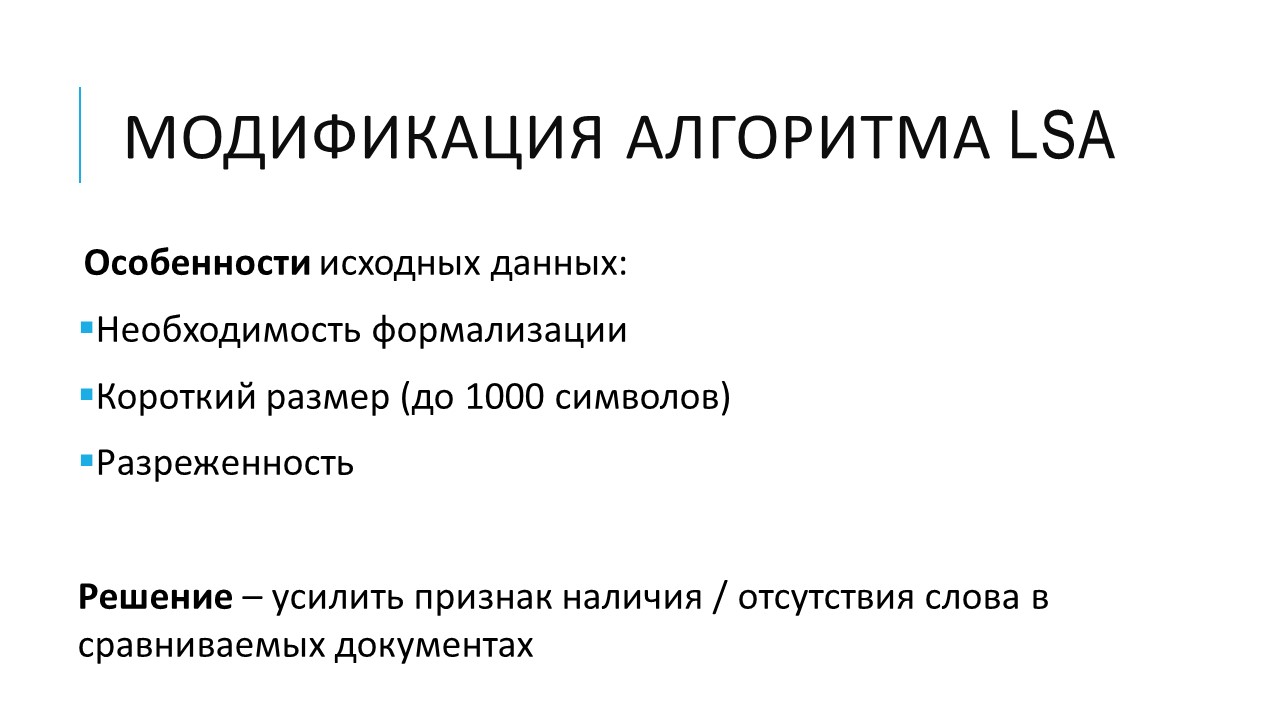


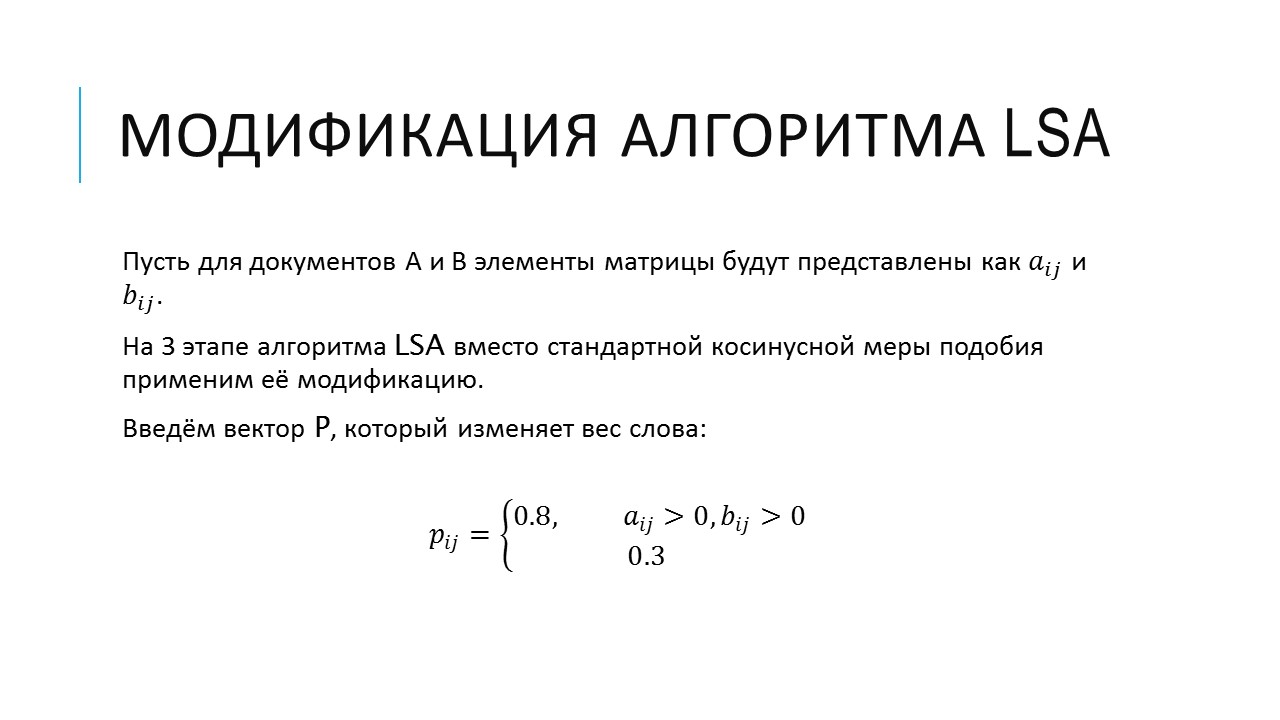


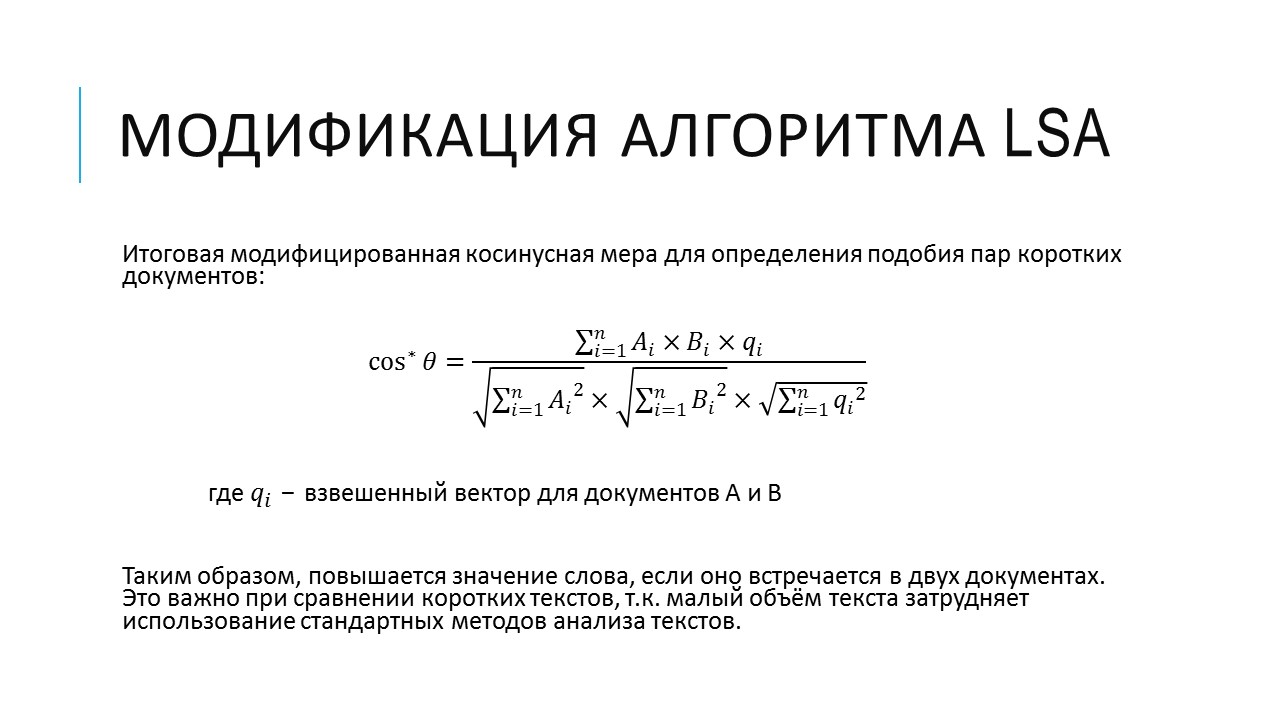


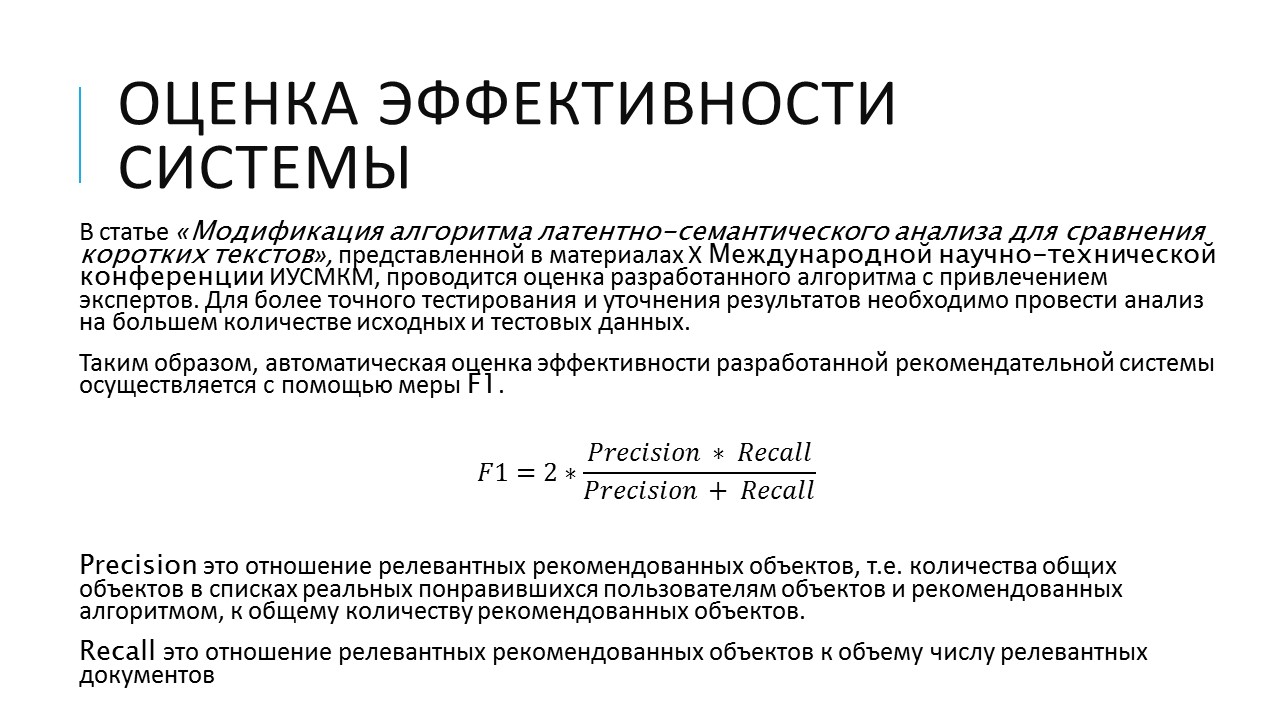


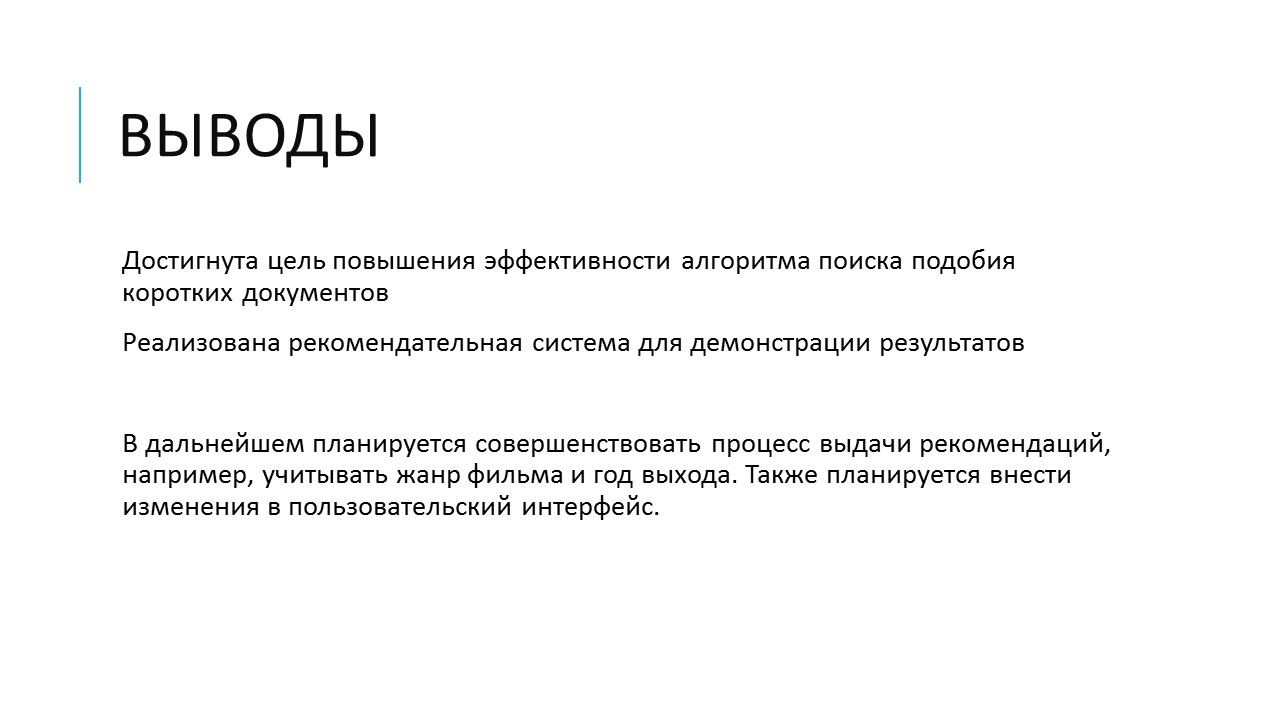
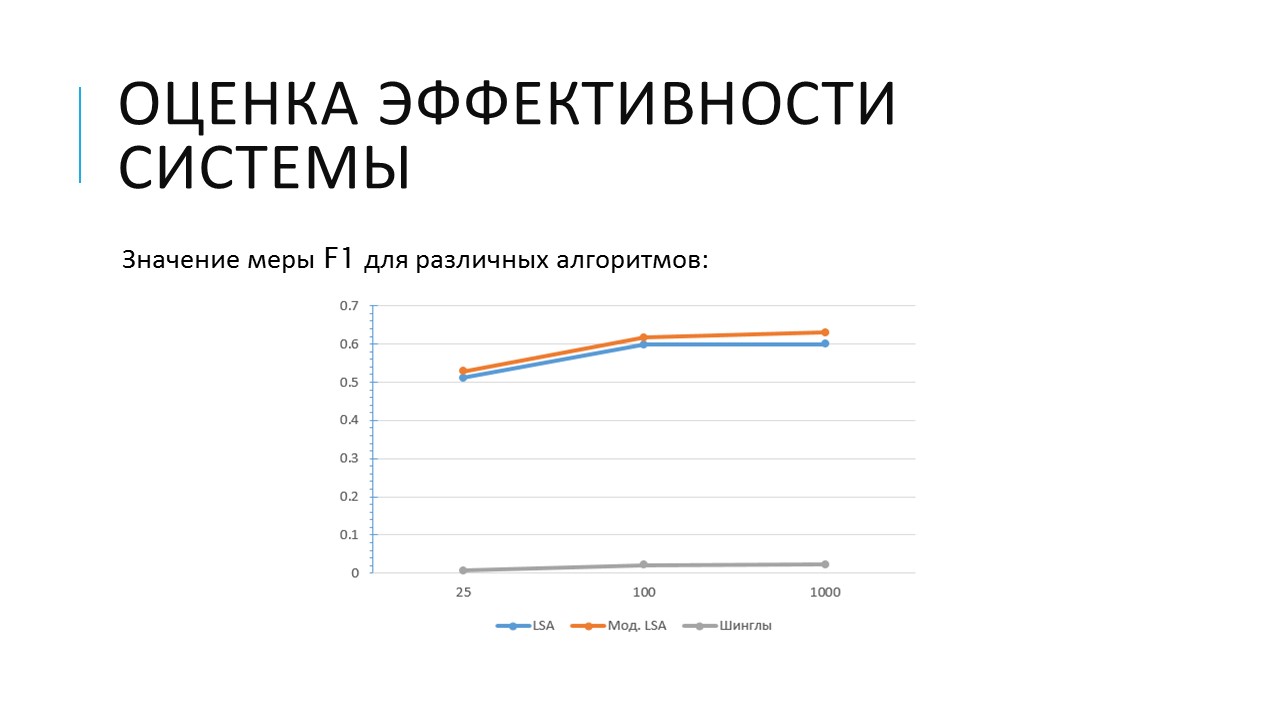
****

****

****

****

****

****