Федеральное государственное образовательное бюджетное учреждение
высшего профессионального образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)
Факультет «Факультет информационных технологий и анализа больших данных»
Кафедра «Теория сложных сетей в экономике»

Курсовая работа

на тему:

«Построение рекомендательной системы с помощью векторного представления графа знаний»

Вид исследуемых данных:

Набор данных плейлиста Spotify

Выполнила:
студентка группы ПМ19-4
Качуляк Маргарита
Григорьевна

Научный руководитель:
к.т.н., доцент
Быков Артем
Александрович

Москва 2022г.

Оглавление

1. ВВЕДЕНИЕ

Представленная курсовая работа заключается в построении рекомендательной системы с помощью векторного представления графа знаний.

Актуальность использования рекомендательных систем в современном мире очевидна. В последнее десятилетие наблюдается стремительный рост предложения на рынке товаров и услуг. Можно привести целый ряд примеров: отрывается все больше и больше ресторанов, которые, казалось бы, не отличаются разнообразием меню, но могут предложить авторскую подачу и рецептуру конкретных блюд, аутентичный интерьер и атмосферу; каждый год кинопрокат пополняется огромным количеством фильмов и сериалов, готовых удовлетворить любого киномана; растет число телеканалов, целенаправленных на трансляцию контента определенной страны производства, его конкретного вида или жанра (например, только новости, только юмористические передачи или программы исключительно российского и советского производства). Все это разнообразие предложения заставляет нас испытывать трудности при решении попробовать для себя что-то новое, при этом соответствующее сформированным ранее вкусам и предпочтениям. Однако благодаря современным технологиям и развитию науки сегодня учесть свои предпочтения и определиться с выбором нам позволяет именно рекомендательная система, способная предсказать, какие объекты (фильмы, музыка, книги, новости, веб-сайты) будут интересны клиенту, имея определенную информацию о его профиле.

Рекомендательные системы позволяют персонализировать рекламу и создать индивидуальные подборки для каждого пользователя некоторой сферы услуг или интернет-магазина. Поэтому все больше владельцев таких систем интересуются и стараются внедрять рекомендации в свои сервисы, что помогает существенно увеличить прибыль компаний, ведь от качества рекомендаций и их релевантности сильно зависит удовлетворённость

пользователей сервисом, а, следовательно, и количество времени, которые они будут на нём проводить.

Как результат, в последние годы интерес к разработке и улучшению существующих рекомендательных систем значительно вырос. На сегодняшний день существует большое количество способов реализации рекомендательных систем с помощью различных языков программирования и на основе разных типов данных: как структурированных (таблицы, XML/JSON файлы, графы), так и неструктурированных (текста).

В рамках моей курсовой работы я поставила цель построить рекомендательную систему с помощью векторного представления графа знаний. Такая структура данных является одной из самых удобных и понятных, среди других, предназначенных для хранения связных данных, то есть каким-то образом ссылающихся друг на друга. В качестве графа знаний я решила использовать наборы данных плейлиста Spotify. Один из них основан на подмножестве пользователей в наборе данных #nowplaying, которые публикуют свои твиты #nowplaying через Spotify, а второй характеристики треков. Результирующий набор данных содержит пользователей, плейлисты, которые они слушают, треки, содержащиеся в этих плейлистах, их жанр, популярность и имя исполнителя. В широком смысле эти данные представляет собой связь: «субъект => отношение => объект», благодаря чему мы можем самостоятельно сделать из них граф знаний.

Для достижения цели курсовой работы мною были поставлены следующие задачи:

- исследовать способы построения рекомендательных систем на основе графов знаний
- освоить методы преобразования знаний в графах в численные векторы
- изучить инструменты для работы с графами знаний
- подготовить базу знаний в виде отношений «субъект-предикат-объект»
- создать модель для преобразования данных в численные вектора и оценить ее качество

- разработать рекомендательную системы на примере данных музыкального сервиса Spotify

Подготовка к построению рекомендательной системы пройдет в несколько этапов. Сначала будет рассмотрена теоретическая справка по выбранной теме: информация о системах рекомендации и методах, используемых для ее построения. Затем изучены способы представления графа знаний и метрики для оценки их качества. Далее перейдем к преобразованию выбранных данных в граф знаний и его векторизации, чтобы в последствие создать собственную персонализированную рекомендательную систему, основанную наиболее эффективных методах обработки данных в машинном обучении.

Вся практическая часть будет реализована с помощью языка программирования Python.

2. ТЕОРЕТИЧЕСКАЯ СПРАВКА

2.1 Рекомендательная система на основе графа знаний

Рекомендательную систему можно считать одной из наиболее наукоемких частей сервисов, ведь свое применение в ней могут находить множество мощных технологий, базирующихся на машинном обучении: информационный поиск, обработка естественных языков, нейронные сети и другие.

Рекомендательные системы классифицируются в соответствии с методом, используемым в рекомендации. Различают большое количество различных методов рекомендаций: collaborative filtering (CF), content-based (CB), knowledge-based (KB), demographic-based (DB), utility-based (UB) and hybrid recommendation. В данной работе будет применяться именно knowledge-based метод.

Такой метод рекомендуют товары пользователям на основе знаний домена о том, как товары соответствуют предпочтениям пользователей. Системы рекомендаций, основанные на знаниях, должны использовать три типа знаний: знания о пользователях, знания о товарах и знания о соответствии между товаром и потребностями пользователя. В отличие от таких наиболее популярных методов рекомендации, как совместная фильтрации (CF) (анализируется только реакция пользователей на объекты: рейтинг/оценки, которые выставляют пользователи объектам) и основанная на контенте (CB) (вычисление сходства между элементами на основе характеристик), системы, основанные на знаниях, не сталкиваются с проблемами с новыми пользователями (когда новый пользователь, который недавно вошел в систему, не оценил ни один товар, следовательно, не может получать рекомендации из-за отсутствия пользовательских предпочтений) и новыми товарами (если новый товар имеет слишком мало оценок, следовательно, не может быть рекомендован) – проблемами холодного запуска. Это происходит, потому что KG рекомендации не привязаны к рейтингам, а вместо этого они используют знания домена, что позволяет дать более надежные рекомендации,

основанные на персональной информации. Поэтому методы, основанные на знаниях, подходят для гибридизации с другими методами рекомендаций.

2.2 Граф знаний и его представления

Базой данных, содержащую структурированную информацию о человеческом опыте и знаниях в некоторой предметной области и сервисы по работе с ней (поиск, правила вывода и т. д.), называют базой знаний (knowledge base, KB). Она была разработана для управления знаниями (метаданными), то есть сбором, поиском, выдачей и хранением знаний.

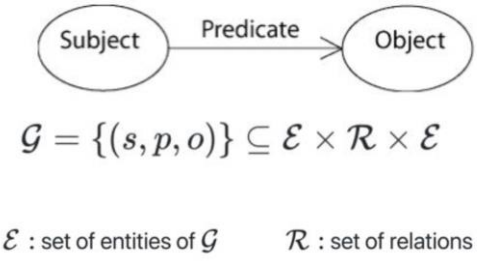
Граф знаний (Knowledge Graph, KG) – это машино-читаемая база знаний, организованная в виде семантической сети. Это означает, что она представляет знания в виде сети с ориентированными связями. В узлах сети находятся некоторые сущности – понятия (абстрактные или конкретные объекты). Отношения между понятиями определяют ориентированные типизированные связи. По сравнению с текстом на естественном языке такая информационная модель отличается большей формализацией и высокой гибкостью, позволяя достаточно содержательно описывать очень широкий диапазон реальных ситуаций и предметных областей, а также находить ёмкое решение для многих задач.

Сейчас в мире существует несколько крупномасштабных универсальных графов знаний: WikiData, BDpedia, Google’s Knowledge Graph, Викиданные (Wikidata) и другие.

Граф знаний включает собственное самоописание в виде онтологии – описания классов сущностей и типов взаимосвязей. Она содержится внутри самого же графа и это позволяет ему иметь гибкую структуру: при необходимости добавления нового типа или нового отношения достаточно ввести имя этого типа или отношения в этом самоописании и дать им определения.

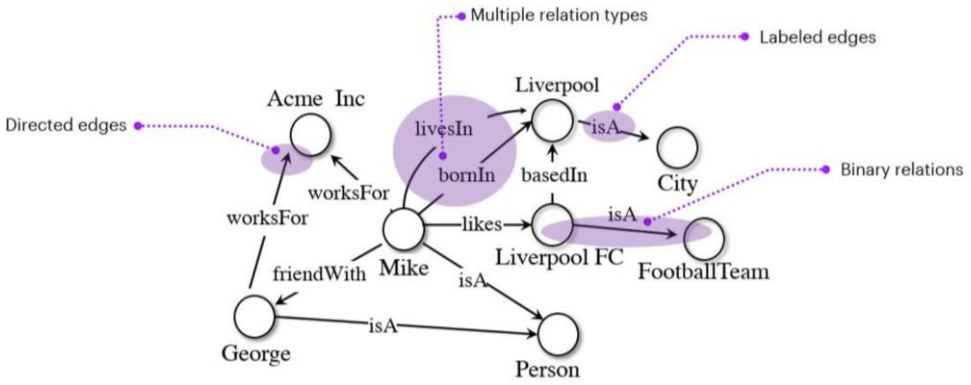
В графе знаний мы описываем семантическую сеть (информационную модель, имеющую вид ориентированного графа) по частям, выделяя каждую связь между узлами. Таким образом формируется структурированные

триплеты «subject – predicate – object» (SPO) (или «head – relation – tail» (HRT)), применимые для машинной обработки.



Структура отношений в графе знаний.

Между двумя вершинами графа знаний может быть несколько ориентированных связей, что позволяет говорить о том, что это мультиграф. Все сущности в нем именованы и имеют типы.



Пример представления графа знаний (knowledge graph).

KG имеет логическую структуру. Формальная семантика позволяет делать выводы – выводить новые факты и интегрировать их в исходный граф. Это способствует упрощению запросов для различных наборов данных и дает возможность выполнять валидацию и проверку консистентности данных, то есть выявить какие-либо внутренние противоречия в графе знаний.

Существует несколько задач машинного обучения на графах знаний:

- Задача идентификации сущностей в KG (entity matching) – позволяет устранять дубликаты, строить вопросо-ответные системы.
- Задача выявления сообществ в KG (сегментация сущностей).
- Задача предсказания ссылок (link prediction). Такие модели помогают пополнять KG, а также использоваться для построения рекомендаций и различных вопросо-ответных систем.

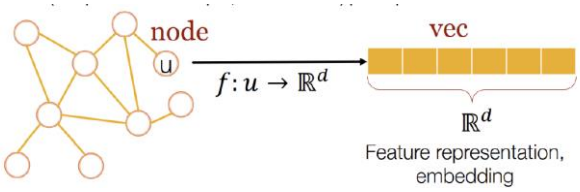
Выделяют два больших семейства представления графа знаний: символическое и векторное. Под Symbolic representation подразумеваются заданные триплеты «subject – predicate – object» (SPO) или в функциональном виде «predicate over subject and object» (p(s, o)). В Vector representation части, предикаты и сущности (SPO) заданы некоторым вектором размерности d (s, p, o $\in R^d$).

Нельзя говорить о том, что какое-то из этих представлений лучше другого. Они используются одинаково активно зависимости от конкретной области работы. Например, символическое представление часто используется в системах управления базами данных, в обработке запросов, в то время как векторное применяется в компьютерном зрении и в целом в машинном обучении.

Реализация практической части данной курсовой работы осуществима только при помощи векторного представления графа знаний, что говорит о необходимости построения эмбеддинга.

2.3 Эмбеддинг графов знаний: определение и способы создания

Эмбеддинг (embedding) узла/графа – непрерывное отображение узла/графа в вектор в пространстве малой (по сравнению с матрицей смежности) размерности.



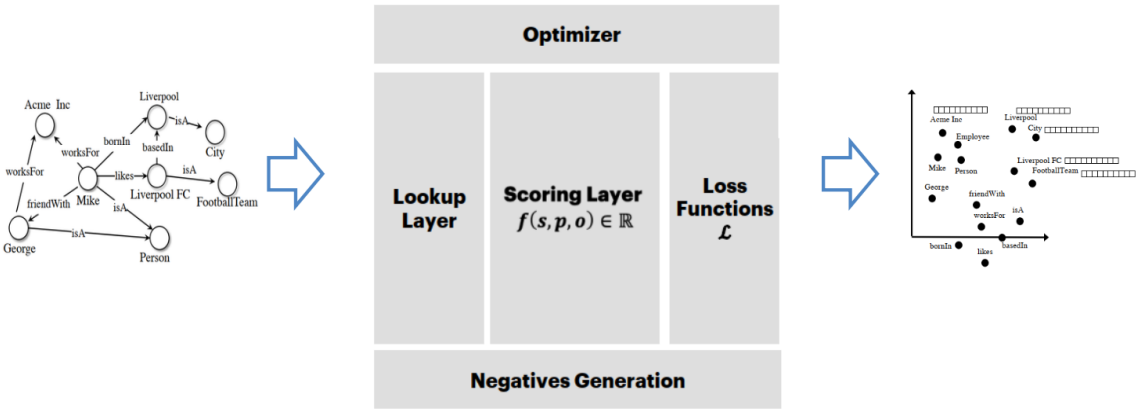
Общая схема построения эмбеддинга графа.

Knowledge Graph embedding (KGE) ставит в соответствие сущностям из графа знаний элементы из непрерывного пространства малой (по сравнению с количеством узлов) размерности.

В отличие от эмбеддинга простого графа эмбеддинг в KG обычно строится не только для узлов, но и для отношений (предикатов).

Эмбеддинг графа знаний включает в себя выявленные признаки. Это значит, что в целевой задаче нет необходимости конструировать вручную. Они закодированы и доступны автоматически.

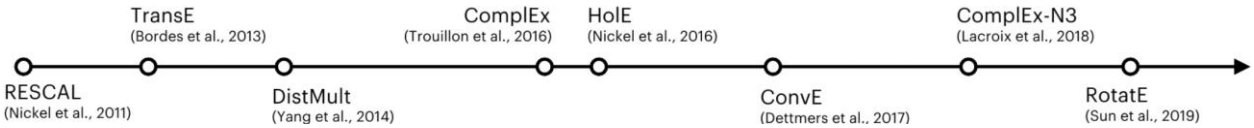
Рассмотрим общий фреймворк графа знаний.



Общая схема алгоритма построения KGE.

Любой классический алгоритм эмбеддинга состоит из нескольких ключевых блоков: Optimizer (сторонняя составляющая, не требующая генерации для конкретного KGE), Lookup Layer (стандартен для всех неглубоких эмбеддингов), Scoring Layer (функция скоринга, определяющая насколько триплет SPO удачно приближен), Loss Functions (функция потерь) и Negative Generation (генерирование искусственных отрицательных примеров для того, чтобы эффективно выстроить функцию потерь).

На сегодняшний день разработано большое количество моделей эмбеддинга графов знаний.



Некоторые модели KGE.

Каждый из них отличается собственным подходом к расчету той или иной составляющей алгоритма KGE и различной сложностью. Обратим внимание на отдельные из них.

В TransE (Translating Embeddings – классический алгоритм) отношения в графе знаний представлены как вектор от субъекта к объекту, такой что векторные вложения должны удовлетворять выражению: $s + p \approx o$.

Векторное представление каждой сущности и связи в графе знаний может быть вычислено путем обучения модели, которая минимизирует функцию скоринга, векторную норму различия между суммой «головной» сущности и отношения и «хвостовой» сущностью в пространстве вложения: $f_{TransE} = -||(e_s + r_p) - e_o||_n$.

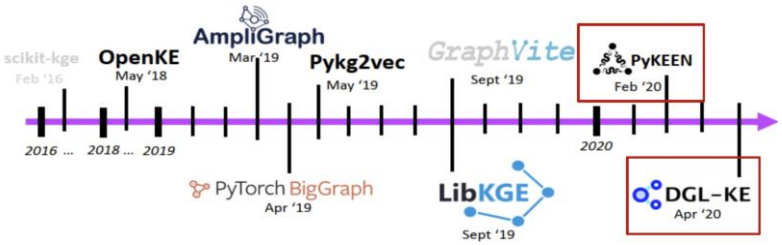
RESCAL модель факторизации низкого ранга с тензорным произведением. Она связывает каждую сущность с вектором для представления его скрытой семантики. Отношения представляются в виде матрицы, моделирующей попарные взаимодействия между латентными факторами. Функция скоринга: $f_{RECAL} = e_s^T \times W_r \times e_o$.

Complex embeddings (ComplEx) – билинейная диагональная модель. Композиция ComplEx embeddings может обрабатывать большое разнообразие бинарных отношений, среди которых симметричные и антисимметричные отношения. По сравнению с остальными современными моделями, такой подход несколько проще, поскольку он использует только эрмитово точечное произведение, комплексный аналог стандартного точечного произведения между вещественными векторами. Этот метод масштабируется для больших наборов данных, поскольку он остается линейным как в пространстве, так и во времени, при этом неизменно превосходя альтернативные подходы по стандартным критериям прогнозирования связей. Функция скоринга основана на трехлинейном эрмитовом точечном произведении: $f_{ComplEx} = Re(\langle r_p, e_s, e_o \rangle)$.

Согласно исследованиям, эта модель обладает самой современной прогностической способностью, в связи с чем именно ее было принято использовать для реализации практической части данной курсовой работы.

2.4 Инструменты для работы с графами знаний

Существует большое количество пакетов, предназначенных для обучения и оценки моделей эмбеддинга графов знаний.



Основные библиотеки python для работы с KGE.

Каждая из них отличается количеством встроенных методов и функций, специализацией (реализуется на разных языках программирования), форматом входных данных и графикой.

В данной курсовой работе практическая часть будет реализована с помощью библиотеки Ampligraph. Такой выбор обусловлен тем, что этот пакет уже достаточно долгое время применяется программистами (с 2019 года), имеет несколько обновлений и большое количество учебных пособий в сети (Slack/Colab), а значит есть возможность изучить уже реализованные методы эмбеддинга, узнать о наиболее производительных параметрах для обучения модели. Также эта библиотека поддерживает необходимый в текущей практической работе формат – csv, имеет встроенные метрики для оценки модели, понятный интуитивно понятный API-интерфейс, а также основан на TensorFlow, что позволяет легко и быстро развертывать сложные численные вычисления с большим объемом данных.

2.5 Метрики оценки результата эмбеддинга

После обучения модели для эмбеддинга данных графа знаний необходимо оценить, насколько качественно и точно она способна преобразовывать данные в векторы. В библиотеке ampligraph для этого можно воспользоваться несколькими встроенными метриками: mr_score, mrr_score и hits_at_n_score. Все они оценивают средний ранг (расположение) неизведанных, заведомо правдивых, триплетов по сравнению с искусственно-сгенерированными моделью. Для нахождения этих рангов используется функция evaluate_performance. Она получает на вход все данные графа (тренировочные триплеты, на которых модель была обучена, и тестовые), для

каждого из них генерирует «ложные» тройки, а затем возвращает ранг каждой тройки из тестовой выборки в получившемся наборе.

- mr_score (Mean Rank): вычисляет среднее значение вектора ранжирования рангов

$$MR = \frac{1}{|Q|}\sum_{i=1}^{|Q|} rank_{(s,p,o)_i}$$

- mrr_score (Mean Reciprocal Rank): вычисляет среднее значение, обратное элементам вектора ранжирования рангов

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_{(s,p,o)_i}}$$

- hits_at_n_score (Hits@N): вычисляет сколько элементов вектора рейтинговых рангов попадают на верхние n позиций

$$Hits@N = \frac{1}{|Q|}\sum_{i=1}^{|Q|} 1\ if\ rank_{(s,p,o)_i} \geq N$$

3. ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Создание графа знаний

Тематикой рекомендательной системы данной курсовой работы стало предложение треков пользователям стриминг-сервиса Spotify, позволяющего легально прослушивать музыкальные композиции, аудиокниги и подкасты, не скачивая их на устройство.

На популярной публичной платформе данных Kaggle представлен датасет «Spotify Playlists», содержащий информацию о пользователях, их плейлистах и треках, находящихся в этих плейлистах. Такой набор позволяет создавать связи «subject => predicate => object» только с 5 видами отношений («track writtenBy artist», «artist performs track», «user listenTo track», «track belongsTo playlist», «playlist contains track»), 2 из которых являются симметричными, чего недостаточно для построения качественной RS. Поэтому было принято решение добавить еще пару признаков для треков из датасета «Spotify Tracks DB», содержащего подробную характеристику музыкальных композиций (danceability, loudness и тд). По ключам «artist_name» и «track_name» первому датасету были присвоены такие признаки треков как жанр («genre») и популярность («popularity»), в результате чего датасет может быть преобразован в записи с 7 видами отношений:

- track writtenBy artist
- user listenTo track
- artist performs track
- track hasGenre genre
- track hasPopularity popularity
- track belongsTo playlist
- playlist contains track

Однако, прежде чем создавать триплеты отношений, необходимо сформировать новый ключ для треков в виде «трек + артист», поскольку существует вероятность наличия треков с одинаковыми названиями.

Теперь, когда все ключи готовы, с помощью встроенной функции библиотеки Ampligraph приступаем к комплектации триплетов. «Dataframe_to_triples» принимает на вход первичный датасет и список кортежей (subject, relation_name, object), где subject и object находятся в заголовках фрейма данных. Получившиеся массивы соединяются в единый и сохраняются в новый DataFrame, применяя удаление дубликатов.

3.2 Модель эмбеддинга графа знаний и ее производительность

Переходим к обучению модели эмбеддинга на полученных в предыдущем пункте тройках.

Разделяем датасет с триплетами на тестовую (для оценки производительности эмбеддинга) и обучающую выборки (для обучения модели), применяя вспомогательную функцию Amligraph «train_test_split_no_unseen». Её особенность заключается в том, что наши точки данных представляют собой две сущности, объединенные некоторой связью, и нам нужно позаботиться о том, чтобы все сущности были представлены в наборах обучения и проверки по крайней мере одной тройкой.

Выбор гиперпараметров в модели ComplEx основан на наилучших результатах, которые были реализованы и изучены до сих пор для данной модели, применяемой к некоторым контрольным наборам данных:

- batches_count = 100 (количество пакетов, на которые разбивается обучающий набор во время цикла обучения)
- epochs = 70 (количество эпох для обучения модели)
- k = 100 (размерность пространства вложения)
- eta = 10 (количество отрицательных или ложных троек, которые должны быть сгенерированы во время выполнения обучения для каждой положительной или истинной тройки)
- остальные параметры по умолчанию

Теперь оценим модель эмбеддинга с помощью метрик, о которых было упомянуто в пункте 2.5. Применяем функцию evaluate_performance и считаем mr_score, mrr_score и hits_at_n_score для n = {1;3;10}.

```
MR to min: 1370.96
MRR to max: 0.52
Hits@1 to max: 0.44
Hits@3 to max: 0.57
Hits@10 to max: 0.68
```

Таблица 1. Результаты оценки модели эмбеддинга.

Полученные результаты заявляют, что качество модели достаточно хорошее (лучших показателей можно добиться при помощи увеличения эпох или уменьшения количества пакетов в обучении модели, однако это требует большей производительности компьютера). MR_score выше 0,5, а значит в среднем мы угадывали правильный предмет или объект в 52% случаев. Модель будет ранжировать правильную сущность в топ-10 в 68% случаев, в топ-3 в 57% случаев и в топ-1 в 44% случаев. А средний рейтинг положительных троек составляет 1370,96.

3.3 Построение рекомендательной системы

Обученная модель эмбеддинга позволяет создавать векторы данных, а значит объяснять близость объектов в некотором пространстве на основе их отношений с другими объектами.

Используемый в работе датасет содержит информацию о пользователях и их треках. Эмбеддинг уникальных юзеров будет представлять собой набор векторов, расстояние между которыми обратно пропорционально некоторой величине схожести музыкальных предпочтений пользователей (то есть чем больше расстояние, тем менее схожие вкусы людей). Аналогично при помощи векторов уникальных треков можно оценить и их подобие.

Таким образом, используя эмбеддинг, рекомендательная система, цель которой – предложить пользователю неизвестные ему треки, может иметь два подхода:

1. Искать рекомендации среди треков других пользователей, имеющих похожие вкусы
2. Рекомендовать треки, подобные тем, что пользователь уже слушает

Реализуем оба подхода. Каждый из них будет основываться на поиске ближайших «соседей».

Широко используемым алгоритмом для их нахождения является алгоритм ближайших соседей k-nearest neighbors algorithm (k-NN), содержащийся в пакете Scikit-learn (sklearn). Чтобы найти точки данных, наиболее близкие к заданной точке, функция «NearestNeighbors» класса sklearn.neighbors использует математическое определение расстояния, называемое евклидовым. На вход необходимо подать количество соседей для поиска (k), тогда метод «kneighbors» вернет два массива: индексы ближайших точек и дистанции до них, размерности (len(X), k), где X – точки, на которых модель была обучена.

Библиотека Ampligraph уже имеет встроенную функцию «find_nearest_neighbours», в алгоритме которой участвует KNN. Она работает в пространстве эмбеддинга и находит желаемое количество соседних вложений, принимая на вход обученную модель эмбеддинга, список объектов, для которых необходимо вычислить «соседей», количество вычисляемых «соседей» и список объектов, из которых необходимо вычислить соседей. При этом возвращает также массив «соседей» и массив соответствующих расстояний до них.

```
def find_nearest_neighbours(kge_model, entities, n_neighbors=10, entities_subset=None, metric="euclidean"):
    assert kge_model.is_fitted, "KGE model is not fit!"
    assert isinstance(entities, (list, np.ndarray)), \
        "Invalid type for entities! Must be a list or np.array"

    if entities_subset is not None:
        assert isinstance(entities_subset, (list, np.ndarray)), \
            "Invalid type for entities_subset! Must be a list or np.array"

        all_neighbors_emb = kge_model.get_embeddings(entities_subset)
        all_neighbors = entities_subset
    else:
        all_neighbors_emb = kge_model.trained_model_params[0]
        all_neighbors = list(kge_model.ent_to_idx.keys())

    assert n_neighbors < len(all_neighbors), 'n_neighbors must be less than the number of entities being fit!'
    knn_model = NearestNeighbors(n_neighbors=n_neighbors, metric=metric).fit(all_neighbors_emb)

    test_entities_emb = kge_model.get_embeddings(entities)
    distances, indices = knn_model.kneighbors(test_entities_emb)
    out_neighbors = []
    for neighbor_idx_list in indices:
        out_neighbors.append([])
        for neighbor_idx in neighbor_idx_list:
            out_neighbors[-1].append(all_neighbors[neighbor_idx])

    return np.array(out_neighbors), np.array(distances)
```

Документация библиотеки Ampligraph. Функция «find_nearest_neighbours»

Таким образом, применяя «find_nearest_neighbours», напишем алгоритм нахождения рекомендаций среди треков других пользователей, имеющих похожие вкусы:

1) Создаем функцию, аргументом которой будет выбранный нами случайно пользователь [USER] среди всех уникальных.

2) Находим для него трех «соседей» с помощью «find_nearest_neighbours» (аргументы: model, entities=[USER], n_neighbors=4 (ищем четыре, так как первым будет являться сам «user»), entities_subset=все уникальные пользователи)
3) Создаем список песен, которые есть у «соседей», но отсутствуют у «USER» (ищем треки в исходном датафрейме по фиксированным id пользователей-соседей). Здесь же для каждого трека находим его популярность, чтобы в дальнейшем предложить пользователю отсортированный по общему рейтингу список рекомендаций.
4) Формируем итоговый датафрейм, где первый столбец – рекомендуемый трек, второй столбец – его популярность, сортируем по последнему и выводим на экран ТОП-10.

Для пользователя с id «05db2293372a2e4a425b3bbfd8961fbe» система RS-1 предложила следующие треки (см. код в приложении 6.2 «RS-1»):

	Tracks	Popularity
2	All of Me (by artist John Legend)	85
8	Demons (by artist Imagine Dragons)	74
10	Work REMIX (by artist A$AP Ferg)	73
0	The Days (by artist Avicii)	69
3	Welcome Home, Son (by artist Radical Face)	68
1	Hold Back The River (by artist James Bay)	67
4	Retrograde (by artist James Blake)	63
5	Berlin (by artist RY X)	60
7	1234 (by artist Feist)	60
9	22 (by artist Taylor Swift)	60

Результат RS-1 для USER = «05db2293372a2e4a425b3bbfd8961fbe».

Теперь реализуем систему, которая будет рекомендовать треки, подобные тем, что пользователь уже слушает:

1) Создаем функцию с аргументом USER (случайный пользователь среди всех уникальных)
2) Находим треки, которые он уже знает (user_tracks) и, исключая их из списка уникальных треков (all_tracks), получаем список тех композиций, что он точно еще не слушал (unknown_tracks)
3) Для каждого трека в плейлисте пользователя с помощью «find_nearest_neighbours» находим еще один, максимально похожий

(аргументы: model, entities=[track in user_tracks], n_neighbors=2 (ищем два, поскольку первым будет являться сам «track»), entities_subset=unknown_tracks)

4) Создаем список песен-соседей, найденных в предыдущем пункте, а также список расстояний. Здесь же для каждого трека находим его популярность в исходном датафрейме по фиксированным трекам-соседям, чтобы в дальнейшем предложить пользователю отсортированный по общему рейтингу и дистанциям список рекомендаций.
5) Формируем итоговый датафрейм, где первый столбец – рекомендуемый трек, второй столбец – его популярность, удаляем дубликаты строк и сортируем по последнему столбцу и выводим на экран ТОП-10.

Для пользователя с id «05db2293372a2e4a425b3bbfd8961fbe» система RS-2 предложила следующие треки (см. код в приложении 6.2 «RS-2»):

	Tracks	Popularity
0	Bitch Better Have My Money (by artist Rihanna)	77
1	Maps (by artist Maroon 5)	77
2	Clocks (by artist Coldplay)	75
3	Kiss Me (by artist Ed Sheeran)	75
4	Tenerife Sea (by artist Ed Sheeran)	74
5	Hall of Fame (by artist The Script)	73
6	Partition (by artist Beyoncé)	71
7	Beautiful Now (by artist Zedd)	71
8	Mess Is Mine (by artist Vance Joy)	71
9	If I Were a Boy (by artist Beyoncé)	70

Результат RS-1 для USER = «05db2293372a2e4a425b3bbfd8961fbe».

Таким образом, для одного и того же пользователея музыкальной платформы мы получаем два списка рекомендованных к прослушиванию треков. Это происходит, потому что рекомендательные системы имеют различную логику поиска интересных для пользователя песен. Какая из рекомендаций является более точной и действительно понравится клиенту, вероятно, можно узнать только протестировав каждую из них на конкретных пользователях и оценив их отклики.

4. ЗАКЛЮЧЕНИЕ

В результате данной курсовой работы с помощью векторного представления графа знаний были построены две рекомендательные системы, предлагающие для пользователей сервиса Spotify неизвестные для них треки.

Все поставленные задачи решены:

- изучена необходимая теория
- исследованы методы разработки рекомендательных систем с помощью графов знаний
- из датасета сформирована база знаний в виде отношений «субъект-предикат-объект»
- освоены необходимые для векторизации знаний в графе алгоритмы
- создана, обучена и оценена модель эмбеддинга
- для случайно-выбранного пользователя рекомендованы треки к прослушиванию

Удалось понять, что рекомендательные системы на основе векторного представления графа знаний могут иметь разнообразные подходы поиска продуктов или услуг, которые могли бы быть интересны пользователю. В результате чего получилось придумать две реализации цели курсового проекта: первая находит рекомендации среди треков пользователей, имеющих схожие с целевым клиентом вкусы, а вторая рекомендует треки, подобные тем, что клиент уже слушает.

Оценить преимущество одной из них над второй, вероятно, можно лишь использовав их в реальной жизни и проанализировав отзывы пользователей или исследовав какое количество треков попадало в плейлист пользователя после предложенных ему композиций из той или иной рекомендации.

В любом случае, практическая значимость полученных результатов каждой системы рекомендации высока. Оптимизируя выбор и экономя время, клиент получает удовольствие от использования платформы. Она начинает привлекать больше внимания к себе. Следовательно, популярность и рейтинг сервиса растут.

Таким образом, использование рекомендательной системы положительно влияет как на интернет-платформу, увеличивая прибыль компании, которой она принадлежит, так и на самого пользователя, радуя его новыми интересными предложениями.

5. СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Online community of data scientists and machine learning practitioners Kaggle, Spotify Playlists [Электронный ресурс]. URL: https://www.kaggle.com/datasets/andrewmvd/spotify-playlists (дата обращения: 3.05.2022)
2. Online community of data scientists and machine learning practitioners Kaggle, Spotify Tracks DB [Электронный ресурс]. URL: https://www.kaggle.com/datasets/zaheenhamidani/ultimate-spotify-tracks-db (дата обращения: 20.04.2022)
3. Zhang, Fuzheng, et al. “Collaborative knowledge base embedding for recommender systems.” Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, 2016.
4. Макрушин, С.В. Лекции «GPB. Курс по сетям» [Электронный ресурс]. URL: https://disk.yandex.ru/d/WSg0J44xscVOxw (дата обращения: 22.04.2022)
5. Документация библиотеки для работы с графами знаний Ampligraph [Электронный ресурс]. URL: https://github.com/Accenture/AmpliGraph (дата обращения: 20.04.2022)
6. Библиотека Python с открытым исходным кодом, которая предсказывает связи между понятиями в графе знаний. (Ampligraph 1.4.0) [Электронный ресурс]. URL: https://docs.ampligraph.org/en/1.3.2/index.html (дата обращения: 29.04.2022)

6. ПРИЛОЖЕНИЕ

6.1 Характеристики компьютера

Тип процессора – Apple M1 (8 ядер, 4 производительных и 4 энергоэффективных)

Тактовая частота – 3.20 ГГц

Частота системной шины – 1066 МГц

Объем кэш-памяти второго уровня (L2) – 16 MB

6.2 Коды программ

Установка пакетов и библиотек:

```
%pip install "tensorflow-gpu>=1.15.2,<2.0"
import tensorflow as tf
%pip install ampligraph
import pandas as pd
import numpy as np
import time
import random
from random import randint
from random import choices
from ampligraph.latent_features import ComplEx
from ampligraph.discovery import find_nearest_neighbours
from ampligraph.evaluation import evaluate_performance
from ampligraph.evaluation import mr_score, mrr_score, hits_at_n_score
from ampligraph.evaluation import train_test_split_no_unseen
from ampligraph.utils.model_utils import dataframe_to_triples

from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
```

Обработка датасета:

```
df1 =
pd.read_csv('/content/drive/MyDrive/Курсовая2022/spotify_dataset.csv',
dtype=str, on_bad_lines='skip', skiprows=1, names=['user_id',
'artist', 'track', 'playlist'])
df2 =
pd.read_csv('/content/drive/MyDrive/Курсовая2022/spotify_features.csv'
, on_bad_lines='skip')
df2 = df2.drop(columns=['track_id', 'danceability', 'energy',
'liveness', 'acousticness', 'duration_ms', 'instrumentalness', 'key',
'loudness', 'mode', 'speechiness', 'tempo', 'time_signature',
'valence'])
```

```
df2 = df2.drop_duplicates(subset=['artist_name',
'track_name']).reset_index(drop=True)
df = pd.merge(df1, df2, left_on=['artist','track'], right_on
=['artist_name', 'track_name'], how='inner')
df = result.drop(columns=['artist_name', 'track_name'])
df = result.sort_values('user_id')
```

Преобразование исходного датасета в триплеты:

```
df["track_id"] = df.track.astype(str) + ' (by artist ' +
df.artist.astype(str)+ ')'
df["user_id"] = df.user_id.values.astype(str)
df["artist_id"] = df.artist.astype(str)
df["playlist_id"] = df.playlist.astype(str)
df["genre"] = df.genre.astype(str)
df = df.drop(columns=['artist', 'playlist', 'track'])
schema1 = [['user_id', 'listenTo', 'track_id']]
schema2 = [['track_id', 'writtenBy', 'artist_id']]
schema3 = [['track_id', 'belongsTo', 'playlist_id']]
schema4 = [['track_id', 'hasGenre', 'genre']]
schema5 = [['track_id', 'hasPopularity', 'popularity']]
schema6 = [['artist_id', 'performs', 'track_id']]
schema7 = [['playlist_id', 'contains', 'track_id']]

tr1 = dataframe_to_triples(df, schema1)
tr2 = dataframe_to_triples(df, schema2)
tr3 = dataframe_to_triples(df, schema3)
tr4 = dataframe_to_triples(df, schema4)
tr5 = dataframe_to_triples(df, schema5)
tr6 = dataframe_to_triples(df, schema6)
tr7 = dataframe_to_triples(df, schema7)

output = np.vstack([tr1, tr2, tr3, tr4, tr5, tr6, tr7])
tr_df = pd.DataFrame(output, columns=['subject', 'relate', 'object'])
tr_df = tr_df.sort_values(['subject', 'object'])
tr_df = tr_df.drop_duplicates().reset_index(drop=True)
```

Embedding графа знаний:

```
x_train, x_test = train_test_split_no_unseen(tr_df.to_numpy(),
test_size=10881)
model = ComplEx(batches_count=100,
                epochs=70, #100
                k=100,
                eta=10,
                verbose=True)
model.fit(x_train)
```

Таблица 1. Результаты оценки модели эмбеддинга:

```
filter_triples = np.concatenate((x_train, x_test))
```

```
ranks1 = evaluate_performance(x_test,
                              model=model,
                              filter_triples=filter_triples,
                              use_default_protocol=True,
                              verbose=True)
print("MR to min: %.2f" % (mr_score(ranks1)))
print("MRR to max: %.2f" % (mrr_score(ranks1)))
print("Hits@1 to max: %.2f" % (hits_at_n_score(ranks1, n=1)))
print("Hits@3 to max: %.2f" % (hits_at_n_score(ranks1, n=3)))
print("Hits@10 to max: %.2f" % (hits_at_n_score(ranks1, n=10)))
```

3D визуализация данных:

```
def make_3d_visualization(objects, model, color):
  embs = model.get_embeddings(objects)
  embs_3d = PCA(n_components=3).fit_transform(np.array([i for i in
embs]))
  pdf = pd.DataFrame({"x": embs_3d[:,0],
                      "y": embs_3d[:,1],
                      "z": embs_3d[:,2]})
  fig = plt.figure(figsize = (8, 6))
  ax = Axes3D(fig)
  ax.scatter(pdf.x, pdf.y, pdf.z, color=color)
  plt.show()
uni_tracks = list(df.track_id.unique())
uni_users = list(df.user_id.unique())
make_3d_visualization(uni_users, model, color='blue')
make_3d_visualization(uni_tracks, model, color='green')
embst = model.get_embeddings(uni_tracks)
embs_3dt = PCA(n_components=3).fit_transform(np.array([i for i in
embst]))
embsu = model.get_embeddings(uni_users)
embs_3du = PCA(n_components=3).fit_transform(np.array([i for i in
embsu]))

fig = plt.figure(figsize = (10, 7))
ax = Axes3D(fig)
ax.scatter3D(embs_3dt[:,0], embs_3dt[:,1], embs_3dt[:,2], s=50,
color='green')
ax.scatter3D(embs_3du[:,0], embs_3du[:,1], embs_3du[:,2], s=50,
color='blue')
plt.show()
```

Выбор пользователя для рекомендации:

```
USER = random.choice(uni_users)
user_tracks = list(df[df.user_id==USER].track_id.values)
all_tracks = list(df.track_id.unique())
unknown_tracks = list(set(all_tracks) - set(user_tracks))
print('Пользователь --', USER)
```

```
print('Количество треков в прейлисте пользователя:', len(user_tracks))
print('Количество неизвестных пользователю треков:', len(all_tracks))
```

RS-1 (Рекомендательная система по 3-ем "соседям" пользователя):

```
def RS_fnn(user):
  neighbors, dist = find_nearest_neighbours(model,
                                            entities=[USER],
                                            n_neighbors=4,
                                            entities_subset=uni_users)
  neighbors = list(neighbors[0][1:])
  neighbors_tracs =
list(tr_df[(tr_df.subject==neighbors[0])|(tr_df.subject==neighbors[1])
|(tr_df.subject==neighbors[2])].object.unique())
  tracks_to_rec = list(set(neighbors_tracs) - set(user_tracks))
  pops = []
  for track in tracks_to_rec:
    pops.append(list(df[df.track_id==track].popularity.unique())[0])
  rec_df = pd.DataFrame({'Tracks':tracks_to_rec, 'Popularity':pops})
  rec_df = rec_df.sort_values(by='Popularity', ascending=False)
  return rec_df.head(10)
```

RS-2 (топ-10 треков, максимально похожих на те, что слушает пользователь):

```
def REC_2(user):
  user_tracks = list(df[df.user_id==user].track_id.values)
  all_tracks = list(df.track_id.unique())
  unknown_tracks = list(set(all_tracks) - set(user_tracks))
  rec_tracks, pops = [], []
  for track in user_tracks:
    near_track, nt_dist = find_nearest_neighbours(model,
                                                  entities=[track],
                                                  n_neighbors=1,

entities_subset=unknown_tracks)
    for t, d in zip(near_track[0], nt_dist[0]):
      rec_tracks.append(t)
      pops.append(list(df[df.track_id==t].popularity.unique())[0])
      unknown_tracks = list(set(unknown_tracks) - set(t))
  rec_df = pd.DataFrame({'Tracks': rec_tracks, 'Popularity':
pops}).reset_index(drop=True)
  rec_df = rec_df.sort_values(by=['Popularity'],
ascending=False).reset_index(drop=True)
  return
rec_df.drop_duplicates(subset=['Tracks']).reset_index(drop=True).head(
10)
```

6.3 Список файлов

Kachulyak_ПМ19-4_RS-VKG.ipynb – файл JupiterNotebook, техническая реализация практической части работы.

Курсовая_2022_Качуляк_МГ_ПМ19-4.pdf – Печатная работа.

spotify_dataset.csv – музыкальная база данных платформы Kaggle с информацией о пользователях и их музыкальных предпочтениях.

spotify_features.csv – музыкальная база данных платформы Kaggle с информацией о характеристиках треков (жанр, популярность).

6.4 Время работы программ

Программа	Время работы, с.
Установка пакетов и библиотек	78.47641086578369
Обработка датасета	45.135735750198364
Преобразование исходного датасета в триплеты	5.619055271148682
Embedding графа знаний	2061.595826625824
Таблица 1. Результаты оценки модели эмбеддинга	1444.5133047103882
Выбор пользователя для рекомендации	0.0516657829284668
RS-1 (по 3-ем "соседям" пользователя: (find_nearest_neighbours))	0.5389599800109863
RS-2 (топ-10 треков, максимально похожих на те, что слушает пользователь)	1.6082019805908203