УДК 004.6-024.11:004.

ГРАФ ЗНАНИЙ И МАШИННОЕ ОБУЧЕНИЕ КАК ИТ СРЕДА ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ ИНТЕРНЕТ ИСТОЧНИКОВ





М.П. Батура ***И.И. Пилецкий*** ***Н.А. Волорова*** ***П.А.Зорко*** ***А.О.Кулевич .***

Белорусский государственный университет информатики и радиоэлектроники, Республика Беларусь
E-mail: bmpbel@bsuir.by, ianmenski@gmail.com, volorova@bsuir.by, polina.zorko16@gmail.com, kulevich.01@gmail.com,

М.П. Батура
Заведующий лабораторией НИЛ 8.1 «Новые обучающие технологии» БГУИР, Доктор технических наук, профессор, академик «Международной академии наук высшей школы», заслуженный работник образования Республики Беларусь. Область научных исследований: Системный анализ, управление и обработка информации в технических и организационных системах. Опубликовано более 150 научных работ.

И.И. Пилецкий
Кандидат физико-математических наук, доцент БГУИР, имеет более 100 публикаций, сфера научных интересов – разработка проектов по обработке больших объёмов данных.

Н.А. Волорова
Заведующая кафедрой информатики БГУИР, кандидат технических наук, доцент. В сфере IT более 40 лет. Имеет более 140 публикаций, сфера научных интересов – модели сложных систем.

П.А. Зорко
Студент БГУИР специальности «Информатика и технологии программирования».

А.О. Кулевич
Студентка БГУИР специальности «Информатика и технологии программирования».

Аннотация. В статье приводится описание архитектурных и технологических решений совместного использования графовых БД, графов знаний и методов машинного обучения (ML) для построения прототипа компонента системы интеллектуального анализа данных интернет источников. Приводятся принятые решения по созданию платформы для решения задач с помощью совместного использования графовых БД и ML, демонстрируются полученные результаты.

Ключевые слова*: интернет-источники, Big Data, анализ, графовая БД, граф знаний, Neo4j, Embedding*

Введение.

В настоящее время в окружающем нас мире циркулируют огромные потоки информации. Повсеместная информатизация приводит к накоплению огромных объёмов данных в науке, производстве, бизнесе, транспорте, здравоохранении. Поэтому в любой организации, как большой, так и маленькой, возникает проблема такой организации управления данными, которая обеспечила бы наиболее эффективную работу. Возникающие при этом задачи анализа данных, прогнозирования, управления и принятия решений стали выполнятся с помощью машинного обучения.

Машинное обучение, по мнению аналитиков, является самым многообещающим технологическим трендом современности. Машинное обучение базируется на идее о том, что аналитические системы могут учиться выявлять закономерности и принимать решения с минимальным участием человека.

Широкое распространение получило использование машинного обучения в графовых базах данных. Графовые технологии - это основа для создания интеллектуальных приложений, позволяющая делать более точные прогнозы и быстрее принимать решения. Графы лежат в основе широкого спектра вариантов использования искусственного интеллекта (ИИ), начиная от разработки лекарств до рекомендаций по дружбе в социальных сетях.

Совместное применение методов и алгоритмов машинного обучения с графовыми технологиями позволяет получать скрытые зависимости и выполнять предиктивный анализ информации, получать ответы в режиме реального времени, реализовывать алгоритмы искусственного интеллекта.

Аналитическая обработка переопределила выполнение некоторых бизнес-процессов, поскольку расширенная аналитика в реальном времени становится неотъемлемой частью самого процесса, а не отдельной выполняемой операцией после факта [1].

Как правило при решении задачи перехода от неформальных данных к знаниям нужно решать две проблемы: представление знаний и обучение знаниям. Для представления знаний применяют различные методы моделирования информации. Так, граф знаний (KG – Knowledge graph) - используется для представления знаний. Графы знаний состоят из набора взаимосвязанных типизированных объектов и их атрибутов. С помощью комбинации графовых технологий и применения методов анализа графовых моделей и машинного обучения (ML) создается технологии глубокого анализа данных интернет источников. Для создания компонента системы анализа данных необходимо решить несколько непростых задач: построение графовой БД и графа знаний предметной области, определении существенных свойств из графовой модели для дальнейшего анализа данных, преобразовании выбранных свойств в векторное представление, выбор подходящего алгоритма ML и применении его для глубокого анализа данных.

Граф знаний, графовые алгоритмы и машинное обучение.

Граф знаний (KG – Knowledge graph) стал одним из самых мощных инструментов для моделирования отношений между субъектами в различных областях, от биотехнологий до электронной коммерции, от моделирования искусственного интеллекта до применения в различных сферах финансовых технологий. Современные KG используются в различных приложениях, включая поисковые системы, социальные сети, чат-боты, коммерческие системы рекомендаций покупок.

Gartner прогнозирует, что к 2025 году графовые технологии будут использоваться в 80% инноваций в области данных и аналитики по сравнению с 10% в 2021 году, что облегчит быстрое принятие решений по всей организации [2].

Графы знаний являются частным случаем более общей категории графов. Граф знаний — это как правило помеченный и ориентированный граф, который получается путем анализа и фильтрации исходных данных.

В общем случае, графы знаний представляют собой взаимосвязанные наборы некоторых данных, которые описывают реальные сущности и их отношения друг с другом в простом и понятном виде, в виде графовой модели.

Граф знаний позволяет собирать и объединять данные в информацию, используя отношения данных для получения новых знаний. При построении графовых БД сущности (узлы) и отношения как правило дополняются различными характеристиками, свойствами, атрибутами, что составляет контент модели предметной области. Данный контент используется в системах искусственного интеллекта и графовой аналитике чтобы быстро и точно дать ответ. Например, возьмем обычный граф, который представляет собой

географическую карту, содержащую названия и координаты городов. Тогда, чтобы получить простой граф знаний, можно просто посчитать расстояния между городами. Следовательно, вместо того, чтобы вычислять все расстояния при выполнении запроса, можно сразу спросить: каков кратчайший маршрут между точкой А и точкой В. Предварительный расчет расстояний — это простой шаг, но он значительно ускоряет географический анализ, позволяя также легко тестировать различные сценарии. Например, каков кратчайший путь между А и В, зная, что точка С внезапно недоступна?

Существует множество возможных вариантов использования мощной комбинации графов и ML. Применяя машинное обучение для графа знаний, например, можно выполнить классификацию узлов в графе (зеленые, белые, красные), кластеризацию узлов или предсказать отсутствующие соединения между узлами.

Для решения задач с помощью машинного обучения нужно преобразовать пространство, в котором находится граф, в другое пространство для машинного обучения – векторное пространство для которого применимы известные алгоритмы машинного обучения (на пример, node2vec или GraphSAGE). Данное преобразование выполняется с помощью не простой методологии выделения вектора свойств называемого включением (embedding).

Графы знаний являются основой Graph Data Science (GDS) и применяются для оптимизации различных рабочих процессов, для получения ответов на непростые интеллектуальные запросы.

Графовые алгоритмы, как правило, классифицированы следующим образом: Pathfinnding, Centrality и Community Detection. Графовые включения – это методология представление свойств сущностей (узлов) и свойств отношений в графе как вектор свойств некого пространства размерностью намного меньшей, чем их количество в графе, что позволяет более глубоко анализировать графовую модель, различными алгоритмами в ML.

Графовые технологии - это основа для создания интеллектуальных приложений, позволяющая делать более точные прогнозы и быстрее принимать решения. Граф знаний - одна из основных областей ИИ, которая позволяет понимать предписывающую аналитику и приложения ИИ).

Для решения различных задач может быть применена модель ML и многие алгоритмы с учителем и без учителя. Часть из этих алгоритмов можно применять для более глубокого анализа информации в графавых базах данных. Процесс глубокого обучения использует глубокие искусственные нейронные сети и ML в качестве моделей.

Гибридная транзакционная и аналитическая обработка может потенциально переопределить способ выполнения некоторых бизнес-процессов, поскольку расширенная аналитика в реальном времени становится неотъемлемой частью самого процесса, а не отдельной выполняемой операцией после факта. Гибридная транзакционная и аналитической обработки данных может быть применена в таких областях как: транзакции, маршрутизация, логистика, IoT, социальные сети, кибер-безопасность, обнаружение мошенничества в реальном времени, выдавать рекомендации в реальном времени в сфере услуг, мониторинг и управление различными физическими сложными сетями в реальном времени, предсказание поведения социальных групп и др.

Использование информации графовых моделей и ML, позволяют получать скрытые зависимости и выполнять предиктивный анализ информации, получать ответы в режиме реального времени, реализовывать алгоритмы искусственного интеллекта (ИИ), отслеживать решения ИИ.

Embedding, графовые включения.

В реальном мире большинство данных это неструктурированные данные или данные, содержащие неструктурированное содержимое (например, текст, изображения, аудио, песни). Чтобы использовать эти типы данных для задач машинного обучения, нужны

компактные векторные представления свойств этих типов данных с действительными значениями. Эти векторы свойств называются включениями (embedding).

Графовые включения – это представление узлов и отношений в графе как вектор свойств. В качестве значений вектора свойств могут быть выбраны некоторые атрибуты вершин и отношений. В зависимости от поставленной задачи эти свойства или атрибуты узлов и ребер, могут быть разными. Данные ML технологии могут быть комбинированы с другими методами для улучшения аналитических результатов и нахождения скрытых зависимостей.

Идея, лежащая в основе подходов к обучению представления, состоит в том, чтобы изучить отображение, которое представляет узлы или целые (под)графы, как точки в низкоразмерном векторном пространстве R^d. Цель состоит в том, чтобы оптимизировать это отображение так, чтобы геометрические отношения в этом изученном пространстве отображали структуру исходного графа. После оптимизации пространства отображения, изученные представления могут быть использованы в качестве входных данных для последующих задач машинного обучения.

Технология embedding использует алгоритм кодер-декодер, который состоит из функции кодер и декодер. Кодер отображает узел V^i в низкоразмерное векторное пространство z^i на основе положения узла в графе, его структуры локальных окрестностей и/или его атрибутов. Затем декодер извлекает указанную пользователем информацию из низкоразмерного пространства; это может быть информация о локальной окрестности графа V^i (например, идентификация его соседей) или метка классификации, связанная с V^i (например, метка сообщества). Совместно оптимизируя кодер и декодер, система учится сжимать информацию о структуре графа в низкоразмерное пространство представлений [3]. Схематически алгоритм кодер-декодер представлен на рисунке 1.

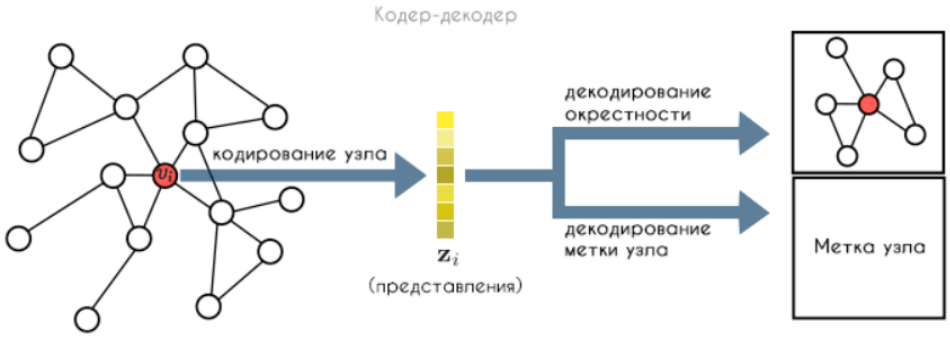


Рисунок 1. Алгоритм кодер-декодер

Для графовых БД применяются функции кодирования и восстановления списка ребер и вершин, их свойств по полученному представлению графа. Функция кодер позволяет выделить важные свойства графа, а функция декодер позволяет визуализировать полученный результат в графическом виде, а также делать предсказание. Например, для взаимодействия графов и нейросетей можно использовать методы, которые находятся в свободном доступе: DeepWalk (word2vec), Node2Vec, 2D CNN, Graph Convolutional Networks [4].

Среда для интеллектуального анализа данных Интернет источников.

В качестве ИТ платформы использовалась среда разработки Neo4j Desktop. С Neo4j Desktop можно создавать и управлять любым количеством локальных баз данных, которое может поддерживать компьютер. Базы данных Neo4j размещаются в экземпляре системы управления базами данных (СУБД), и, начиная с Neo4j 4.0, можно иметь одну или несколько баз данных в одном экземпляре СУБД. Поскольку Desktop может запускать все поддерживаемые в настоящее время версии базы данных Neo4j, можно создать один или

несколько экземпляров СУБД для поддержки разных версий Neo4j, разделить свои базы данных по типу данных, которые они содержат, или другим определённым образом для достижения конкретной желаемой конфигурации СУБД. С помощью Neo4j Desktop можно управлять конфигурацией СУБД, добавлять плагины, просматривать журналы, создавать резервные копии и восстанавливать данные, обновлять версии Neo4j и многое другое, чтобы получить полный жизненный цикл работы с Neo4j.

Graph Data Science Library – это библиотека, которая предоставляет эффективно реализованные параллельные версии общих графовых алгоритмов для Neo4j 3.x и Neo4j 4.x, представленных в виде функций Cypher.

Для анализа данных Интернет источников используются специальные плагины Graph Data Science Library (GDSL) и APOC, которые нужно установить заранее.

Библиотека содержит реализации для следующих типов алгоритмов [5]:

- Поиск пути (Path Finding) – эти алгоритмы помогают найти кратчайший путь или оценить доступность и качество маршрутов;
- Центральность (Centrality) – эти алгоритмы определяют важность отдельных узлов в сети;
- Обнаружение сообщества (Community Detection) – эти алгоритмы оценивают, как группа кластеризуется или секционируется, а также ее тенденцию к укреплению или распаду;
- Сходство(Similarity) – эти алгоритмы помогают вычислить сходство узлов;
- Предсказание топологических связей (Topological link prediction) – эти алгоритмы определяют близость пар узлов;
- Вложения узлов (Node Embeddings) – эти алгоритмы вычисляют векторные представления узлов в графе;
- Классификация узлов (Node Classification) – этот алгоритм использует машинное обучение для прогнозирования классификации узлов;
- Прогнозирование ссылок (Link prediction) – эти алгоритмы используют машинное обучение для прогнозирования новых связей между парами узлов.

APOC (Awesome Procedures on Cypher) – вспомогательный плагин для базы данных. Библиотека APOC считается самой большой и наиболее широко используемой библиотекой расширений для Neo4j. Она включает более 450 стандартных процедур, обеспечивающих функциональные возможности для утилит, преобразований, обновлений графов и многого другого. Они хорошо поддерживаются, и их очень легко запускать как отдельные функции или включать в запросы Cypher.

Примеры совместного применения графа знаний и машинного обучения.

В современном мире огромное влияние уделяется так называемым управляемым знаниям. Ставится задача создания «интеллектуальной фабрики знаний», обеспечивающей постоянную генерацию новых знаний, непрерывно анализирующей множество разрозненных источников данных. Решением являются графы знаний, имеющие в своем составе графовые хранилища семантических метаданных (или знаний, формализованных с помощью специальных формальных семантических языков) и онтологии, которые выступают в роли полуструктурировнной концептуальной схемы предметной области. Именно последнее качество принципиально отличает графы знаний от баз данных, позволяя решать трудноформализуемые интеллектуальные задачи, смещая фокус с задачи хранения данных в сторону связывания, повторного использования и согласованной циркуляции данных. В свою очередь перечисленные аспекты позволяют превратить данные компании в так называемые активные знания (active knowledge) за счет применения современных методов машинного обучения, специализированных для графовых моделей представления данных.

Гибкость и удобство использования графов знаний продемонстрируем на примерах.

В качестве первого примера построим граф знаний содержащий пищевые продукты, их ингредиенты, аллергены и некоторые другие данные (более 6 миллионов продуктов с ингредиентами [6]).

Общее представление полученного графа представлено на рисунке 2.

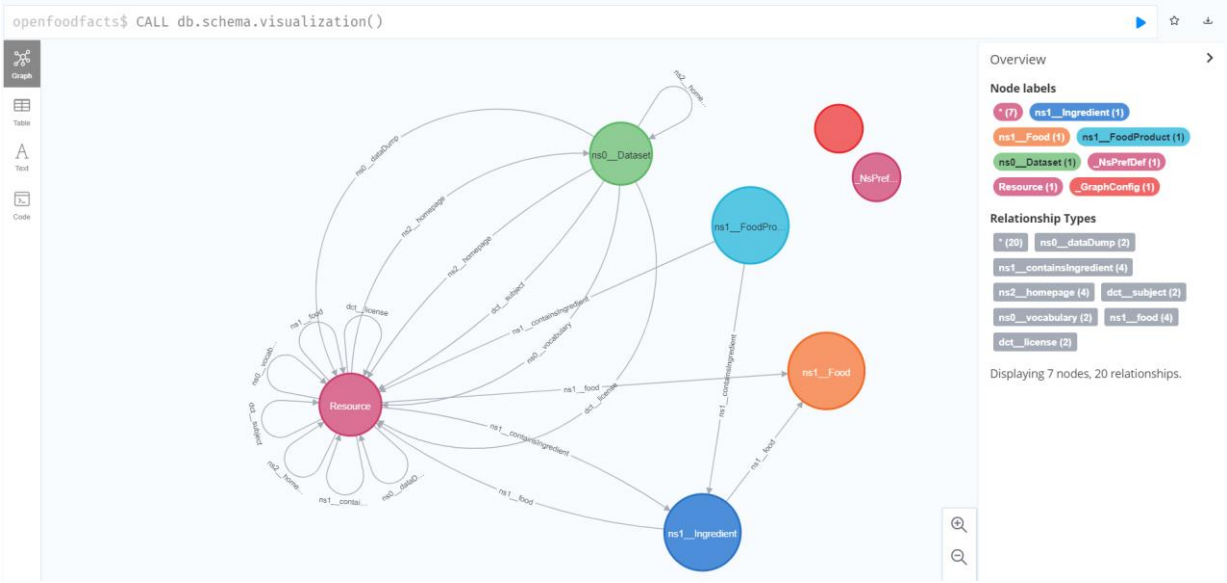


Рисунок 2. Общее представление схемы графовой базы данных

В данной БД основными типами узлов являются:

– *FoodProduct* – продукты;
– *Food* – ингредиенты, которые могут содержаться в продуктах;
– *Ingredient* – промежуточные узлы, соединяющие узлы *FoodProduct* и *Food*, в которых указывается, сколько ингредиента содержится в продукте.

Основными типами связей являются:

– *containsIngredient* – соединяет продукт *FoodProduct* и промежуточный узел *Ingredient*;
– *food* – соединяет промежуточный узел *Ingredient* и ингредиент *Food*.

Используя данный граф знаний, мы легко можем продемонстрировать 15 самых популярных ингредиентов, которые используются в продуктах. Для этого нам понадобится рассмотреть только узлы FoodProduct, Ingredient, Food и связи между ними.

Следующий запрос используется для подсчета использования каждого ингредиента и выводит топ 15 из них:

```
MATCH (:ns2__FoodProduct)-[:ns2__containsIngredient]->()-[:ns2__food]-
(i:ns2__Food)
RETURN i.ns2__name as name, count(*) as Popularity
ORDER BY Popularity DESC
LIMIT 15
```

Результат запроса продемонстрирован в формате таблицы на рисунке 3.

	name	Popularity
1	"Salt"	483566
2	"Sugar"	353868
3	"Water"	320053
4	"Citric acid"	139108
5	"Natural flavor"	134465
6	"Wheat flour"	125080
7	"Spices"	95992

Рисунок 3. Результат работы алгоритма по подсчету популярности ингредиентов

Сохраним полученные данные в файл формата csv, чтобы в дальнейшем их использовать.

Построим гистограмму популярности ингредиентов. Для решения данной задачи будем использовать возможности языка Python. Следующий код позволяет нам получить данные из файла формата csv и представить их в виде гистограммы:

```
import matplotlib.pyplot as plt
import csv
with open('food.csv', newline='') as csvfile:
    food_reader = csv.reader(csvfile, delimiter=',')
    food_reader = list(food_reader)[1:]
    names = [ingr[0] for ingr in food_reader]
    values = [int(ingr[1]) for ingr in food_reader]
    print(values)
    index = [i for i in range(len(values))]
    plt.bar(index, values)
    plt.xticks(index, names, size='small')
    plt.show()
```

Полученная гистограмма представлена на рисунке 4.

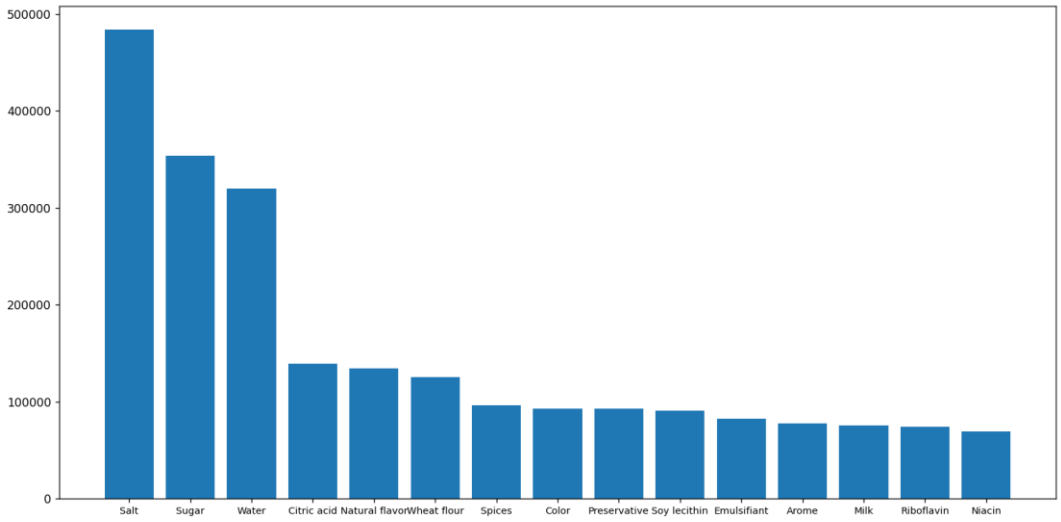


Рисунок 4. Гистограмма популярности ингредиентов

Таким образом, благодаря удобной структуре графа знаний, с помощью небольшого запроса мы смогли определить топ 15 ингредиентов и представили их популярность в наглядном виде. Проанализировав полученную гистограмму легко можем заметить, что самым популярным ингредиентами являются соль, сахар, вода, усилители вкуса, мука. Далее можно выполнить анализ какие продукты полезны, а какие нет исходя из рекомендаций ВОЗ, на пример, можем найти все продукты, которые не содержат пальмовое масло и содержания соли которых на 100 г меньше 0,5 г.

Для следующего примера будем использовать набор данных европейских дорог. Набор данных содержит 894 города и 1250 дорог, соединяющих их. Также есть страна, которой принадлежит город (см. Neo4j Desktop [5]).

Схема полученной базы данных приведена на диаграмме ниже (рисунок 5):

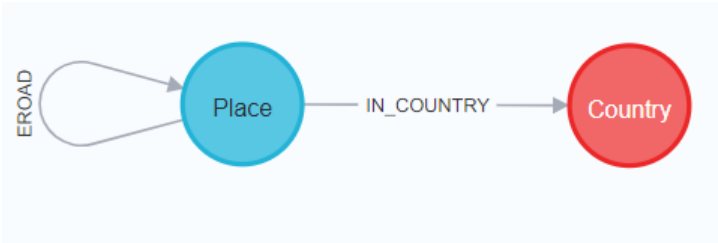


Рисунок 5. Схема графа Европейских дорог

Продемонстрируем в виде гистограммы общую продолжительность магистральных и автомобильных дорог каждой страны.

Для подсчета общей продолжительности дорог каждой страны используется приведенный ниже запрос:

```
MATCH (c:Country)<-[:IN_COUNTRY]-(:Place)-[r:EROAD]->(:Place)
RETURN c.code as code, sum(r.distance) as Total_Length
ORDER BY Total_Length DESC
```

Для построения гистограммы используется следующий код на языке Python:

```
import matplotlib.pyplot as plt
import csv
with open('roads.csv', newline='') as csvfile:
    road_reader = csv.reader(csvfile, delimiter=',')
    road_reader = list(road_reader)[1:]
    country_codes = [road[0] for road in road_reader]
    values = [int(road[1]) for road in road_reader]
    print(values)
    index = [i for i in range(len(values))]
    plt.bar(index, values)
    plt.xticks(index, country_codes, size='small')
    plt.show()
```

Результат работы данного кода представлен на рисунке 6:

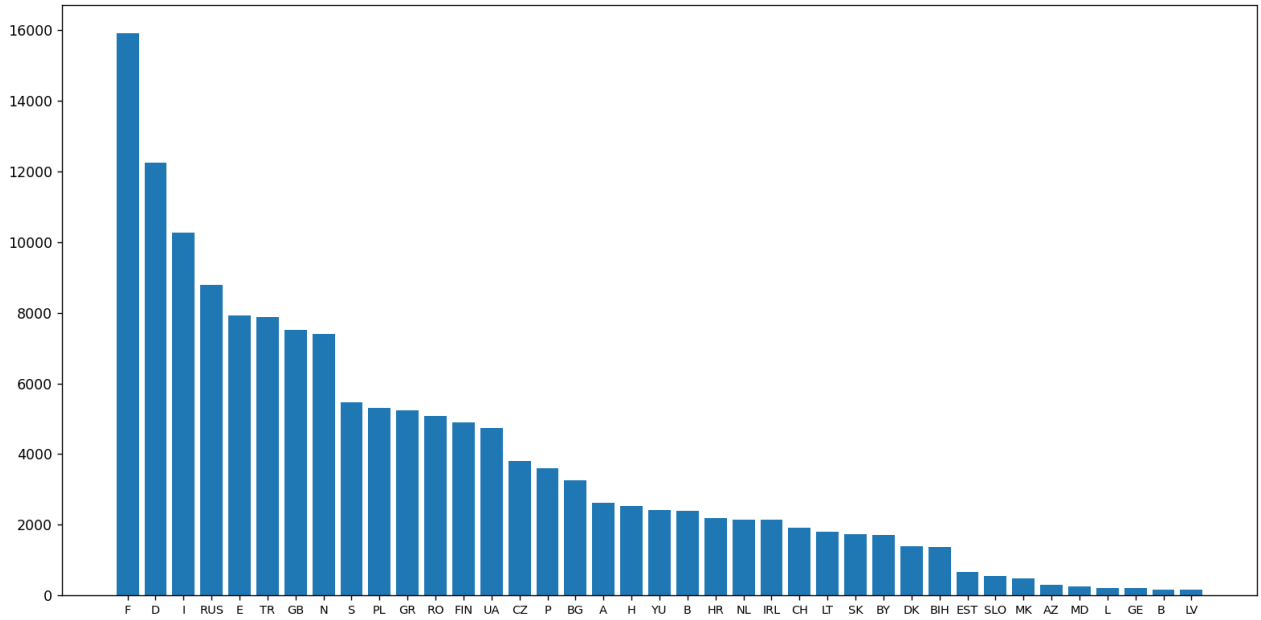


Рисунок 6. Гистограмма продолжительности дорог по странам

Проанализировав гистограмму можно заметить, что Беларусь занимает 28 место по продолжительности магистральных и автомобильных дорог.

Следующий пример также будет строится на базе данных автомобильных дорог.

Продемонстрируем работу алгоритма graph embedding. Используем алгоритм node2vec, который является одним из 2 алгоритмов встраивания (embedding), доступных в библиотеке Graph Data Science. Node2vec создает вложения на основе смещенных случайных блужданий окрестности узла.

Сначала запустим потоковую версию этой процедуры, которая возвращает поток идентификаторов узлов и вложений. Алгоритм имеет следующие обязательные параметры:

nodeProjection – метки узлов, которые будут использоваться для проецируемого графа;

relationshipProjection – типы отношений, которые будут использоваться для проецируемого графа;

embeddingDimension – размер вектора/списка чисел, создаваемых для каждого узла;

iterations – количество итераций для выполнения.

Также уточним *walkLength*, что определяет количество шагов в каждом случайном блуждании. Значение по умолчанию равно 80, что кажется слишком большим для такого маленького графа.

Запустим алгоритм с помощью следующего запроса:

```
CALL gds.beta.node2vec.stream({
  nodeProjection: "Place",
  relationshipProjection: {
    eroad: {
      type: "EROAD",
      orientation: "UNDIRECTED"
    }
  },
  embeddingDimension: 10,
  iterations: 10,
  walkLength: 10
})
```

```
YIELD nodeId, embedding
RETURN gds.util.asNode(nodeId).name AS place, embedding
LIMIT 5;
```

В relationshipProjection, укажем orientation: "UNDIRECTED", чтобы направление связи типа EROAD игнорировалось на проецируемом графе, с которым работает алгоритм.

Если выполним запрос, он вернет следующий результат, приведенный в таблице 1:

Таблица 1. Результаты запроса

place	**embedding**
"Larne"	[0.9289653301239014, 0.7726005911827087, 0.08407653123140335, -1.3916348218917847, 1.163403868675232, -2.1654694080352783, 3.3044044971466064, 0.5551645159721375, 0.2188272476196289, -0.1902276873588562]
"Belfast"	[1.4064077138900757, 1.123956561088562, 0.2082871049642563, -1.5443018674850464, 1.2832289934158325, -2.380089521408081, 3.623403549194336, 0.9978418946266174, 0.49655747413635254, -0.31815648078918457]
"Dublin"	[1.6048322916030884, 0.87335205078125, 0.7916964888572693, -0.6670299768447876, 1.1776026487350464, -2.270751714706421, 2.982720136642456, 1.2146323919296265, 1.127911925315857, -0.8818912506103516]
"Wexford"	[1.998420000076294, 1.7016336917877197, 1.0729856491088867, -0.24083521962165833, 0.708989143371582, -1.944329857826233, 2.500248670578003, 1.5418587923049927, 1.3439892530441284, -0.41184690594673157]
"Rosslare"	[2.0829923152923584, 2.1498963832855225, 1.286923885345459, 0.12702038884162903, 0.5306026935577393, -1.8138670921325684, 2.3255913257598877, 1.8290250301361084, 1.0674598217010498, -0.38719895482063293]

Числа в таблице подставляют собой координаты векторов. Эта процедура недетерминирована, поэтому при каждом ее выполнении будем получать разные результаты.

Этот же результат продемонстрированный в виде точек на плоскости (Limit 1000) (смотрите рисунок 7):

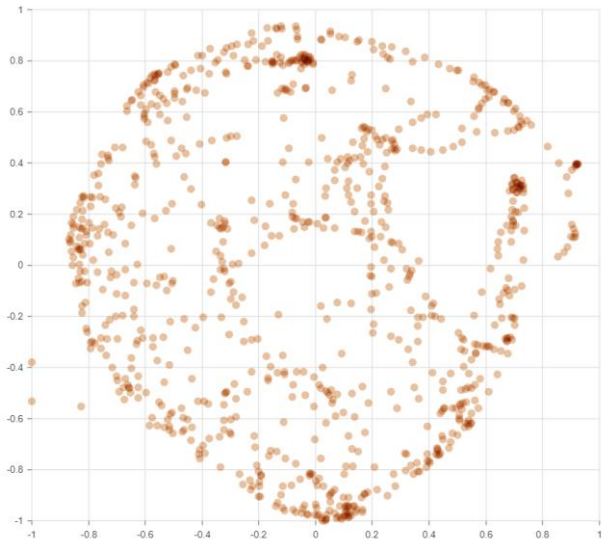


Рисунок 7. Результат работы алгоритма graph embedded для 1000 элементов

Дальнейшее исследование будет проще, если сохраним вложения в Neo4j Desktop в базу данных neo4j, поэтому сделаем это, используя версию процедуры для записи.

Теперь рассмотрим graph embeddings с использованием языка программирования Python, драйвера Python Neo4j и некоторых популярных библиотек Data Science. Создадим диаграмму рассеяния встраивания и посмотрим, можно ли определить, к какой стране относится город, посмотрев на ее встраивание.

Необходимые библиотеки можно установить, выполнив следующую команду:

```
pip install neo4j sklearn altair
```

Создадим файл с именем roads.py и вставим следующие инструкции:

```
from neo4j import GraphDatabase
from sklearn.manifold import TSNE
import numpy as np
import altair as alt
import pandas as pd
driver = GraphDatabase.driver("bolt://localhost:7687(your_host)", auth=("neo4j",
"1234567890(your_password)"))
```

Первые несколько строк импортируют необходимые библиотеки, а последняя строка создает соединение с базой данных Neo4j. Нужно будет изменить Bolt URL-адрес и учетные данные в соответствии с нашей собственной базой данных.

Используем драйвер для выполнения запроса Cypher, который возвращает вложение для городов в таких странах, как Испания, Великобритания, Франция, Турция, Италия, Германия и Греция. Ограничение числа стран облегчит обнаружение любых закономерностей. После выполнения запроса преобразуем результаты в фрейм данных Pandas:

```
with driver.session(database="neo4j") as session:
   result = session.run("""
   MATCH (p:Place)-[:IN_COUNTRY]->(country)
   WHERE country.code IN $countries
   RETURN p.name AS place, p.embeddingNode2vec AS embedding, country.code AS
country
   """, {"countries": ["E", "GB", "F", "TR", "I", "D", "GR"]})
   X = pd.DataFrame([dict(record) for record in result])
```

Теперь приступим к анализу данных.

На данный момент наши вложения имеют размерность 10, но нам нужно, чтобы они были размером 2, чтобы мы могли визуализировать их в 2 измерениях. Алгоритм t-SNE – это метод уменьшения размерности, который уменьшает объекты высокой размерности до 2 или 3 измерений, чтобы их можно было лучше визуализировать. Используем его для создания координат x и y для каждого вложения.

Следующий фрагмент кода применяет t-SNE к вложениям, а затем создает фрейм данных, содержащий каждое место, его страну, а также координаты x и y.

```
X_embedded = TSNE(n_components=2,
random_state=6).fit_transform(list(X.embedding))

places = X.place
df = pd.DataFrame(data = {
   "place": places,
   "country": X.country,
   "x": [value[0] for value in X_embedded],
   "y": [value[1] for value in X_embedded]
})
```

Запустим следующий код для создания диаграммы рассеяния вложений:

```
chart = alt.Chart(df).mark_circle(size=60).encode(
    x='x',
    y='y',
    color='country',
    tooltip=['place', 'country']
).properties(width=700, height=400)
altair_viewer.show(chart)
```

Результат выполнения запроса представлен на рисунке 8:

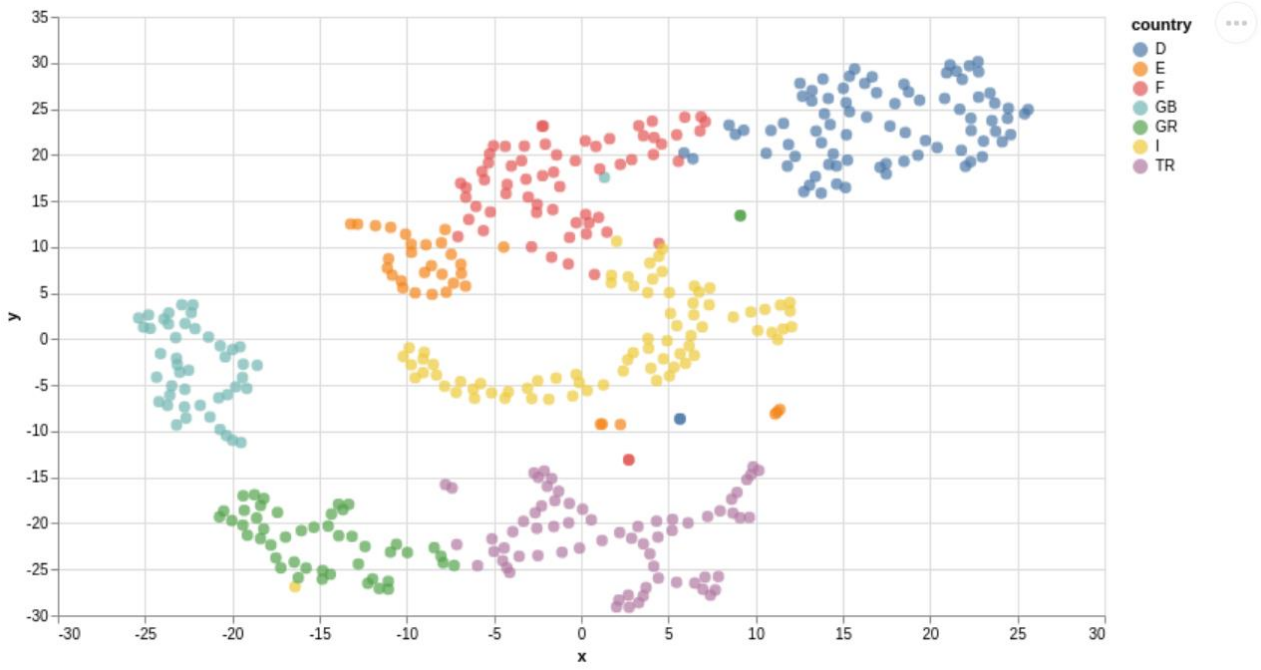


Рисунок 8. Диаграмма рассеяния вложений

При беглом визуальном осмотре этой диаграммы видим, что вложения, сгруппированы по странам.

Для следующего примера будем использовать базу данных содержащую научные статьи, цитаты и их авторов. В ней содержится 629814 научных статей и 632752 цитаты. Нашей задачей является нахождение наиболее важных публикаций и их авторов на заданную тему. Схема графа приведена ниже (рисунок 9):

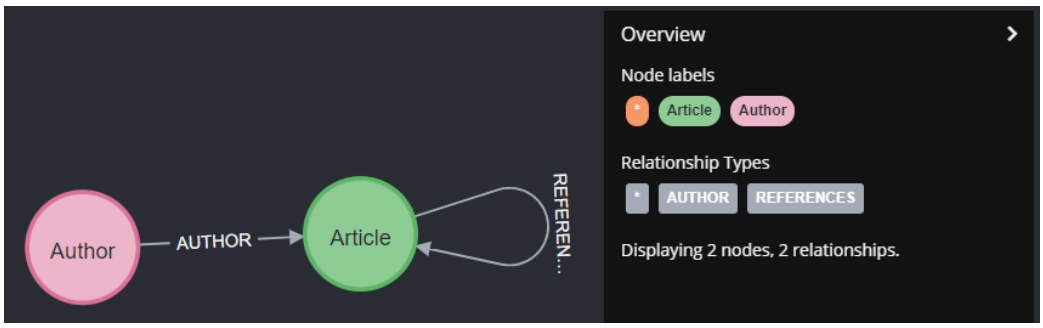


Рисунок 9. Схема графа статей и их авторов

Из схемы видно, что граф состоит из вершин двух типов: авторы и их статьи. Вершины «Авторы», содержащие в себе имя автора в свойстве name, не имеют входящих дуг и ссылаются на свои статьи с помощью дуг «AUTHOR». Вершины «Статьи», хранящие название статьи в свойстве title и аннотацию к ней в свойстве abstract, связаны между собой дугами «REFERENCES». Стоит отметить, что в данном тестовом наборе не все статьи имеют аннотации.

Прежде чем определять наиболее важные публикации, необходимо подготовить граф для применения алгоритмов из библиотеки Graph Data Science. Классический алгоритм PageRank не принимает в расчёт авторов публикаций, поэтому создадим граф только из вершин типов «Article» и дуг типа «REFERENCES». Для того, чтобы указать, с какими вершинами и дугами следует работать библиотеке Graph Data Science, воспользуемся процедурой:

```
CALL gds.graph.create("AGraph", "Article", "REFERENCES")
```

Теперь граф доступен по имени «AGraph».

Применить алгоритм PageRank на графе можно вызовом соответствующей процедуры. Используя вызов версии «write», можно записать вычисленные значения ранга напрямую в вершины графа. Пример запроса вызова процедуры:

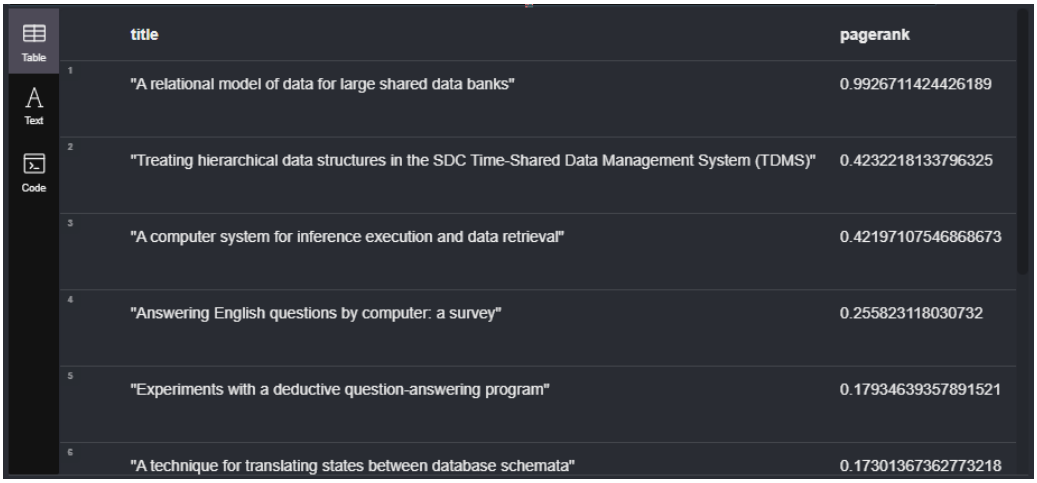
```
CALL gds.pageRank.write("AGraph", {
   writeProperty: "pagerank",
   maxIterations: 20,
   dampingFactor: 0.85,
   scaler: "L1Norm"})
```

В данном примере вычисленное значение нормализируется и записывается в свойство pagerank вершины.

Попробуем найти наиболее важные работы по отношению к фразе «relational database». Для этого применим модификацию RageRank – Personalized PageRank. В качестве исходных статей выберем все статьи, описываемые ключевой фразой «relational database». Выполним следующий запрос:

```
MATCH (:Keyword {value: "relational database"})-[:DESCRIBES]->()<-[:HAS_ANNOTATED_TEXT]-(a:Article)
WITH collect(a) AS nodes
CALL gds.pageRank.stream("AGraph", {
   maxIterations: 20,
   dampingFactor: 0.85,
   sourceNodes: nodes
}) YIELD nodeId, score
WHERE score > 0
RETURN gds.util.asNode(nodeId).title AS title, score AS pagerank
ORDER BY pagerank DESC, title ASC
LIMIT 10
```

По результатам запроса можно сделать вывод, что статья с заголовком «A relational model of data for large shared data banks» внесла наибольший вклад в развитие темы «relational database» (рисунок 10).



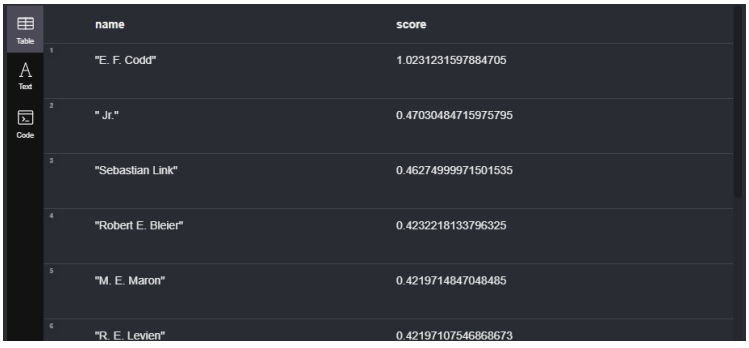
	title	pagerank
1	"A relational model of data for large shared data banks"	0.9926711424426189
2	"Treating hierarchical data structures in the SDC Time-Shared Data Management System (TDMS)"	0.4232218133796325
3	"A computer system for inference execution and data retrieval"	0.42197107546868673
4	"Answering English questions by computer: a survey"	0.255823118030732
5	"Experiments with a deductive question-answering program"	0.17934639357891521
6	"A technique for translating states between database schemata"	0.17301367362773218

Рисунок 10. Наиболее важные статьи на тему «relational database»

Теперь найдём авторов, внёсших наибольший вклад на ту же тему. Так как алгоритм RageRank не предусматривает наличие авторов у статей, определим важность автора как сумму pagerank всех написанных им статей относительно выбранной темы. Выполним следующий запрос:

```
MATCH (:Keyword {value: "relational database"})-[:DESCRIBES]->()<-
[:HAS_ANNOTATED_TEXT]-(a:Article)
WITH collect(a) AS nodes
CALL gds.pageRank.stream("AGraph", {
maxIterations: 20,
dampingFactor: 0.85,
sourceNodes: nodes
}) YIELD nodeId, score
WHERE score > 0
WITH gds.util.asNode(nodeId) AS a, score AS pagerank
MATCH (a)<-[:AUTHOR]-(b:Author)
RETURN b.name AS name, sum(pagerank) AS score
ORDER BY score DESC, name ASC
LIMIT 10
```

Список авторитетов на тему «relational database» показан на рисунке 11. На первом месте оказался E. F. Codd – автор книги, занявшей первое место в предыдущем рейтинге. К слову, Эдгар Франк Кодд – британский учёный, работы которого заложили основы теории реляционных баз данных. Этот факт доказывает работоспособность полученной системы.



	name	score
1	"E. F. Codd"	1.0231231597884705
2	" Jr."	0.47030484715975795
3	"Sebastian Link"	0.46274999971501535
4	"Robert E. Bleier"	0.4232218133796325
5	"M. E. Maron"	0.4219714847048485
6	"R. E. Levien"	0.42197107546868673

Рисунок 11. Авторитеты (эксперты) на тему «relational database»

Заключение.

Результаты приведенные в статье представляют собой инновационный научно-образовательный проект БГУИР. Результаты выполнения проекта используются при обучении студентов и магистрантов по тематике «Обработка больших объемов информации».

В данной статье проанализировано довольно сложное современное направление – представление данных в виде графа знаний и их дальнейшее использование совместно с ML. Продемонстрированы примеры, которые отражают важность графов знаний и совместное применение ML для решения различных практических задач.

Список использованных источников

[1] Gartner [Электронный ресурс] / Режим доступа: https://www.gartner.com/en /research/methodologies / Дата доступа: 23.02.22.

[2] Gartner [Электронный ресурс] / Режим доступа: https://www.gartner.com/en/newsroom/press-releases/2021-03-16-gartner-identifies-top-10-data-and-analytics-technologies-trends-for-2021 / Дата доступа: 20.02.22.

[3] Habr [Электронный ресурс] / Режим доступа: https://habr.com/ru/post/487138/ Дата доступа: 19.02.22

[4] DEEP LEARNING [Электронный ресурс] / Режим доступа: https://jasonyanglu.github.io/files/lecture_notes/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0_2020/Lecture%2011%20Deep%20Learning%20on%20Graphs.pdf / Дата доступа: 20.02.22.

[5] Neo4j [Электронный ресурс] / Режим доступа: https://neo4j.com/developer/graph-data-science/graph-algorithms/ Дата доступа: 10.02.22

[6] Open Food Facts [Электронный ресурс] / Режим доступа: https://world.openfoodfacts.org /Дата доступа: 25.02.22.

ARCHITECTURAL SOLUTIONS FOR QUICK CONSTRUCTION OF A GRAPHIC DB WEB-SITE AND ANALYSIS OF ITS PROPERTIES

M.P. BATURA I.I. PILETSKI H.A.VOLOROVA P.A. ZORKA A.O. KULEVICH

M.P. Batura

Head of the Laboratory of Research Laboratory 8.1 "New Educational Technologies" BSUIR, Doctor of Technical Sciences, Professor, Academician of the "International Academy of Sciences of Higher Education", Honored Worker of Education of the Republic of Belarus. Research area: System analysis, management and information processing in technical and organizational systems.

I.I. Piletski

PhD, Associate Professor of BSUIR.

H.A.Volorova

Head of the Department of Informatics at BSUIR, PhD, Associate Professor.

P.A. Zorka

Student of BSUIR Faculty of Computer Science and Programming Technologies..

A.O. Kulevich

Student of BSUIR Faculty of Computer Science and Programming Technologies.

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus
E-mail: bmpbel@bsuir.by, ianmenski@gmail.com, volorova@bsuir.by, polina.zorko16@gmail.com, kulevich.01@gmail.com,

Abstract. The article provides a description of architectural and technological solutions for the joint use of graph databases, knowledge graphs and machine learning (ML) methods for building a prototype of the data mining component of Internet sources. The decisions made on creating a platform for solving problems using the joint use of graph databases and ML are given, and the results obtained are demonstrated.

Keywords: Internet sources, Big Data, analysis, graph database, knowledge graph, Neo4j, Embedding