ПРОЕКТ

«Разработка полносвязных графов в машинном обучении»

Выполнила:
Смирнова Ника Сергеевна,
ученица 10 “В” класса,
МОБУ “СОШ№6” г. Всеволожска

Научный руководитель:
Ерёменко Максим Викторович

Ленинградская область
2024-2025

Оглавление

Глава 1: Введение:

1.1 Актуальность:

В последние годы наблюдается значительный рост интереса к полносвязным графам как альтернативе традиционным деревьям контекстов в области машинного обучения [1]. Данная тенденция объясняется несколькими ключевыми моментами, которые подчеркивают преимущества графовых структур в сравнении с иерархическими моделями.

Во-первых, полносвязные графы обладают высокой гибкостью. Они способны эффективно представлять сложные взаимосвязи между данными, что особенно важно в многослойных системах. В отличие от деревьев, которые имеют фиксированную иерархическую структуру, графы могут легко адаптироваться к изменениям, обеспечивая более точное отражение реальной природы взаимосвязей [2].

Во-вторых, использование полносвязных графов может значительно улучшить производительность моделей машинного обучения. Графы позволяют более глубоко анализировать взаимосвязи между элементами данных, что особенно актуально в задачах обработки естественного языка [3]. Например, графовые модели способны лучше захватывать семантические связи [4] между словами и фразами, что приводит к более точным предсказаниям и улучшенному пониманию контекста.

Кроме того, полносвязные графы демонстрируют высокую масштабируемость. Это свойство становится критически важным при работе с большими объемами данных, где традиционные деревья могут сталкиваться с ограничениями по производительности и сложности. Графы обеспечивают возможность обработки больших сетей взаимосвязей, что открывает новые горизонты для реализации сложных задач в области анализа данных [5].

Таким образом, переход от деревьев контекстов к полносвязным графам не только актуален, но и необходим для достижения более высоких результатов в анализе данных и построении предсказательных моделей. Исследование графовых структур в контексте машинного обучения представляет собой многообещающую область, способную значительно продвинуть технологии в различных областях, от обработки естественного языка до социальных сетей и биоинформатики [6].

1.2 Гипотеза:

Использование полносвязных графов в задачах машинного обучения позволит значительно улучшить качество предсказаний и производительность моделей по сравнению с традиционными деревьями контекстов.

1.3 Цель:

Разработать алгоритм по созданию полносвязных графов в машинном обучении и сравнить его эффективность с существующими типами архитектур в машинном обучении.

1.4 Задачи:

1. Анализ существующих методов и алгоритмов, применяемых в машинном обучении и использующих графы.
2. Разработка алгоритма, проведение тестов его работы и внедрение его в модели машинного обучения.
3. Сравнение производительности моделей, основанных на полносвязных графах, с моделями, использующими деревья контекстов, в решении задач.

1.5 Объект исследования: архитектуры машинного обучения.
1.6 Предмет исследования: полносвязные графы и их практическое применение.

1.7 Методы исследования:

1. Экспериментальное исследование: проведение экспериментов с использованием разработанного алгоритма для создания полносвязных графов и анализ результата.
2. Сравнительный анализ: оценка эффективности полносвязных графов по сравнению с другими методами машинного обучения (например, деревьями решений, нейронными сетями).
3. Анализ литературы: изучение существующих научных работ и публикаций по теме полносвязных графов и их применения в машинном обучении.
4. Теоретический анализ алгоритмов: исследование алгоритмов машинного обучения с точки зрения их сложности и эффективности.

Глава 2: Теоретическая часть:

2.1 Введение:

Современные подходы к машинному обучению все чаще требуют интеграции сложных структур данных, таких как графы. Графы, представляющие собой набор узлов и ребер, позволяют моделировать сложные взаимосвязи между объектами и их атрибутами. Полносвязные графы, в частности, обеспечивают возможность учета всех возможных связей между элементами, что значительно расширяет возможности анализа данных. В условиях растущей сложности задач, стоящих перед машинным обучением, использование полносвязных графов [7] может стать ключевым фактором для повышения точности предсказаний и производительности моделей.

Графы знаний и другие подобные структуры уже продемонстрировали свою эффективность в различных областях, включая обработку естественного языка и рекомендательные системы. Однако традиционные методы машинного обучения, такие как деревья решений или линейные модели, часто не способны полностью использовать преимущества графовой структуры данных. Это создает потребность в разработке новых алгоритмов, способных эффективно работать с полносвязными графами.

Цель данного исследования заключается в разработке алгоритма для создания полносвязных графов в контексте машинного обучения и сравнении его эффективности с существующими архитектурами. Для достижения этой цели необходимо провести анализ существующих методов работы с графами и выявить их недостатки. Также важно определить, каким образом полносвязные графы могут улучшить результаты предсказаний по сравнению с традиционными методами.

Задачи исследования включают анализ существующих методов машинного обучения, которые используют графы; разработку нового алгоритма; а также проведение экспериментов для оценки его эффективности. Ожидается, что результаты данного исследования не только подтвердят гипотезу о преимуществах полносвязных графов, но и расширят горизонты читателей о применении графовых структур в машинном обучении.

2.2 Определения из темы графов и их история:

Граф — это математическая структура, состоящая из двух множеств: множества вершин V и множества ребер E.
Каждое ребро представляет собой пару вершин, что позволяет моделировать отношения между объектами. Графы могут быть ориентированными (где ребра имеют направление) или неориентированными (где направление отсутствует). В зависимости от структуры данных, графы могут быть простыми (без петель и кратных рёбер) или сложными (с петлями и кратными рёбрами).

Графы обладают высокой выразительной силой, что делает их подходящими для представления различных типов данных. Например, в социальных сетях пользователи могут быть узлами, а связи между ними — ребрами. В биологии молекулы могут быть представлены как графы, где атомы являются узлами, а химические связи — ребрами.

Виды графов:

Таблица №1 - виды графов

Граф	Ключевые характеристики
Ориентированный граф	● Ребра имеют направление. ● Пример: Социальные сети (друзья, подписчики). ● Ребра направленные (например, A → B). ● Может быть несвязанным (изолированные вершины). ● Применение: Моделирование направленных отношений (например, рекомендации).
Неориентированный граф	● Ребра не имеют направления. ● Пример: Дороги между городами. ● Ребра двусторонние (например, A — B). ● Связный, если существует путь между любыми двумя вершинами. ● Применение: Анализ взаимосвязей без направления (например, кластеризация).

Смешанный граф	● Содержит как ориентированные, так и неориентированные рёбра. ● Пример: Системы с разными типами связей. ● Смешанная структура (например, A → B и A — C). ● Связность зависит от структуры рёбер. ● Применение: Комплексные модели с разными типами взаимодействий.
Полный граф	● Каждая вершина соединена с каждой другой. ● Пример: Полная сеть, где все участники связаны. ● Все возможные ребра присутствуют. ● Всегда связный (все вершины соединены). ● Применение: Полные данные для анализа всех возможных связей.
Пустой граф	● Не содержит ребер, могут быть вершины. ● Пример: Граф без взаимодействий (изолированные узлы). ● Нет рёбер, только вершины. ● Не связен (изолированные узлы). ● Применение: Модели без взаимодействий для анализа отдельных объектов.
Взвешенный граф	● Каждому ребру присвоено числовое значение. ● Пример: Транспортные сети с затратами на маршруты. ● Веса могут представлять расстояния или стоимость. ● Связность зависит от весов и структуры графа. ● Применение: Оптимизация маршрутов и затрат в логистике.

Мультиграф	● Между двумя вершинами может быть несколько рёбер. ● Пример: Транспортные схемы с несколькими маршрутами. ● Разрешены кратные ребра между узлами. ● Может быть несвязанным (разные компоненты). ● Применение: Моделирование альтернативных путей в сетях.

В ходе практической работы, будут исследованы ориентированные, неориентированные и взвешенные графы, а также рассмотрены мультиграфы.

История графов:

История графов [8] начинается с работы швейцарского математика Леонарда Эйлера, который в 1736 году предложил решение знаменитой задачи о Кёнигсбергских мостах. Эта задача заключалась в том, чтобы пройти по всем мостам города, не проходя ни по одному из них дважды. Эйлер показал, что это невозможно, и тем самым заложил основы теории графов, хотя сам термин "граф" еще не использовался.

Первое упоминание слова "граф" в контексте теории графов произошло в 1878 году, когда английский математик Джеймс Сильвестр использовал его в своей статье. Он описывал графы как обобщение диаграмм, используемых в химии и алгебре. В 1936 году венгерский математик Денеш Кёниг опубликовал первую книгу по теории графов под названием "Теория конечных и бесконечных графов" [9], которая систематизировала результаты 200 лет исследований в этой области.

С 1950-х годов теория графов начала активно развиваться благодаря росту кибернетики и вычислительной техники. Графы стали использоваться для моделирования различных систем, включая социальные сети, транспортные сети и компьютерные сети. Это время ознаменовалось развитием таких понятий, как графовые нейронные сети и алгоритмы поиска кратчайшего пути.

Современные исследования в области теории графов охватывают широкий спектр приложений, начиная от оптимизации логистических процессов до

анализа больших данных, демонстрируя свою универсальность и мощность как инструмент для решения сложных задач.

2.3 Полносвязные графы и машина Больцмана:

Полносвязный граф (также может быть назван полным графам) обозначается как Kn, где n — количество вершин. Основное свойство полносвязного графа заключается в том, что он обеспечивает максимальную степень связности между узлами. Это делает его идеальным для задач, где необходимо обеспечить надежную связь между всеми участниками системы.

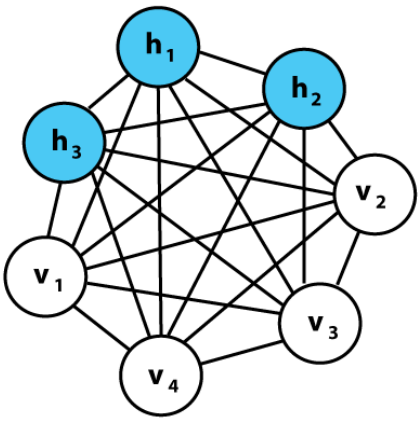
Свойства полносвязных графов:

1. Степень вершин: В каждом полносвязном графе каждая вершина имеет степень n−1, где n — общее количество вершин. Это означает, что каждая вершина соединена со всеми другими вершинами.
2. Плотность: Плотность графа определяется как отношение фактического количества рёбер к максимальному количеству рёбер. Для полносвязного графа плотность равна единице, так как все возможные ребра присутствуют.
3. Кликовые подграфы: Полносвязные графы являются максимальными кликами, то есть любая подмножество вершин образует полный работающий подграф.

Машина Больцмана — это стохастическая (добавляющая шуму в обычную модель) нейронная сеть, разработанная Джеффри Хинтоном и Терри Сейновски в 1985 году [10]. Она названа в честь австрийского физика Людвига Больцмана, который внес значительный вклад в статистическую механику. Основная идея машины Больцмана заключается в использовании принципов статистической физики для моделирования и обучения данных сложным распределениям.

Суть машины Больцмана:

Машина Больцмана состоит из двух типов нейронов: видимых и скрытых. Видимые нейроны представляют собой входные данные, которые можно наблюдать, тогда как скрытые нейроны отвечают за внутренние представления и закономерности, которые сеть изучает на основе видимых данных. Каждый нейрон может находиться в одном из двух состояний: включен (1) или выключен (0). Связи между нейронами являются симметричными и неориентированными, что позволяет каждому нейрону влиять на состояние других.



Обучение машины Больцмана осуществляется с помощью алгоритма имитации отжига, который помогает минимизировать разницу между предсказанными и фактическими состояниями сети. Этот процесс позволяет модели изучать вероятностное распределение по входным данным и выявлять скрытые зависимости.

Полносвязные графы и их связь с машинами Больцмана:

Полносвязные графы — это графы, в которых каждая пара вершин соединена ребром. Это означает, что каждый нейрон в машине Больцмана связан со всеми другими нейронами, что делает её полносвязной сетью. В таком графе каждая вершина (нейрон) может взаимодействовать с любой другой вершиной, что позволяет эффективно передавать информацию и изучать сложные зависимости в данных.

Использование полносвязных графов в контексте машин Больцмана имеет несколько преимуществ:

1. Максимальная связность: Полносвязные графы обеспечивают максимальную степень взаимодействия между нейронами, что позволяет более эффективно изучать зависимости между переменными.
2. Гибкость: Полносвязные структуры могут быть адаптированы для различных задач машинного обучения, включая классификацию, регрессию и генерацию данных.
3. Улучшение качества представлений: Благодаря высокой связности полносвязные графы позволяют машине Больцмана лучше захватывать сложные закономерности в данных, что приводит к более качественным результатам.

Применение машин Больцмана с полными графами:

Машины Больцмана могут быть использованы вместе с полными графами для решения задач подобного вида в области машинного обучения:

1. Генерация образцов: Обученная машина Больцмана может генерировать новые образцы данных на основе изученного

распределения вероятностей. Это полезно в таких областях, как генерация изображений или текстов.
2. Обучение представлений: Машины Больцмана могут использоваться для извлечения скрытых представлений из данных, что позволяет улучшить качество классификации и других задач.
3. Оптимизация: Полносвязные графы позволяют эффективно оптимизировать процессы обучения в машинах Больцмана, что приводит к более быстрой сходимости алгоритмов.

Полные графы часто могут быть рассмотрены в контексте машины Больцмана, поэтому, во время практической части, будет реализована попытка связать написанный граф с машиной Больцмана.

2.4 Машинное обучение и графы:

Машинное обучение (ML) — это область искусственного интеллекта, которая изучает методы и алгоритмы, позволяющие компьютерам обучаться на данных и улучшать свои результаты без явного программирования. Основная идея заключается в том, что вместо того, чтобы вручную задавать правила и алгоритмы для решения конкретных задач, мы предоставляем машине данные и даем ей возможность самостоятельно находить закономерности и делать выводы.

История машинного обучения начинается в середине 20 века, когда ученые начали исследовать возможность создания алгоритмов, способных обучаться на основе данных. В 1950 году Алан Тьюринг предложил концепцию "Тьюринг-теста", который должен был определить, может ли машина мыслить как человек. Это стало важным философским ориентиром для исследователей в области искусственного интеллекта.

В 1952 году Артур Самуэль разработал одну из первых программ, способных обучаться — программу для игры в шашки. Эта программа использовала метод проб и ошибок для улучшения своей игры, что стало одним из первых примеров машинного обучения. В 1956 году на конференции в Дартмутском колледже термин "машинное обучение" был впервые введен Самуэлем, который определил его как процесс, позволяющий компьютерам демонстрировать поведение, не запрограммированное изначально.

В 1957 году Фрэнк Розенблатт представил персептрон — первую нейронную сеть, способную к обучению. Эта модель могла выполнять простые задачи классификации и стала основой для дальнейших исследований в области нейронных сетей. Однако в 1969 году книга

Марвина Минского и Сеймура Пейперта "Perceptrons" указала на ограничения перцептронов, что привело к временному снижению интереса к нейронным сетям.

В 1980-х годах с развитием вычислительной техники и статистических методов произошел значительный прорыв в машинном обучении. Появились новые алгоритмы, такие как метод опорных векторов и деревья решений, которые стали широко применяться на практике. В это время также был разработан алгоритм обратного распространения ошибки (backpropagation), который значительно улучшил эффективность нейронных сетей.

С начала 2000-х годов наблюдается рост интереса к глубокому обучению (deep learning), которое стало возможным благодаря увеличению вычислительных мощностей и доступности больших объемов данных. Глубокие нейронные сети, состоящие из множества слоев, способны извлекать сложные представления из данных и достигать высоких результатов в таких задачах, как распознавание изображений и обработка естественного языка.

Ключевым моментом в истории машинного обучения стало создание таких систем, как суперкомпьютер Deep Blue, который в 1997 году выиграл матч у чемпиона мира по шахматам Гарри Каспарова. Этот успех продемонстрировал потенциал ML в решении сложных интеллектуальных задач.

В последние годы машинное обучение стало неотъемлемой частью многих технологий и приложений, от рекомендательных систем до автономных транспортных средств. Современные исследования активно исследуют новые методы и подходы, включая графовые нейронные сети и обучение с подкреплением.

Существует несколько основных видов машинного обучения, которые можно классифицировать следующим образом:

1. Обучение с учителем (Supervised Learning):
   В этом подходе модель обучается на размеченных данных, где каждому входному примеру соответствует известный выход. Алгоритмы используют эти примеры для выявления закономерностей и построения модели, которая может предсказывать выходные значения для новых, неразмеченных данных. Используется с работой по классификации (например,

определение спама в электронной почте) и регрессии (например, предсказание стоимости недвижимости).

2. Обучение без учителя (Unsupervised Learning):
   В этом случае модель работает с не размеченными данными и пытается самостоятельно выявить структуры или паттерны в данных. Здесь нет заранее определенных выходных значений, из-за чего и требуется наличие человека для обучения модели. Использование: кластеризация (группировка пользователей по схожести) и уменьшение размерности.
3. Обучение с частичным привлечением учителя (Semi-supervised Learning):
   Этот подход комбинирует элементы обучения с учителем и без учителя. Модель обучается на небольшом количестве размеченных данных вместе с большим объемом неразмеченных данных. Метод полезен в ситуациях, когда разметка данных является трудоемкой или дорогой.
4. Обучение с подкреплением (Reinforcement Learning):
   В этом подходе агент обучается через взаимодействие с окружающей средой. Он получает вознаграждения или штрафы за свои действия и использует эту информацию для оптимизации своей стратегии. Можно встретить в шахматах: когда человек играет против компьютера, разработчики зачастую используют данные модели для того, чтобы играть роль соперника для игрока.

Помимо различия в видах, машинное обучение также разделяется по видам архитектур. Различные архитектуры машинного обучения используются для решения специфических задач. Некоторые из наиболее распространенных архитектур включают:

1. Линейные модели:
   Простые модели, такие как линейная регрессия и логистическая регрессия, которые используются для предсказания числовых значений или вероятностей.
2. Деревья решений:
   Структуры данных, которые принимают решения на основе последовательности вопросов о признаках входных данных. Они легко интерпретируемые и широко используются в задачах классификации.
3. Случайный лес:
   Ансамблевая модель (модель, задействующая процесс по прогнозированию результата с использованием разнообразных базовых моделей), состоящая из множества деревьев решений. Она

улучшает точность за счет объединения предсказаний нескольких деревьев.

4. Нейронные сети:
   Самые знаменитые в обществе модели, вдохновленные работой человеческого мозга. Они состоят из слоев нейронов и могут обрабатывать сложные данные (изображения и текст). Глубокие нейронные сети (Deep Learning) — это многослойные нейронные сети, которые способны извлекать сложные представления из данных.
5. Графовые нейронные сети:
   Эти сети предназначены для работы с графовыми данными и способны учитывать структуру графа при обработке информации. О них в данной работе и идет речь.

Использование графов в машинном обучении:

Графы представляют собой мощный инструмент для моделирования сложных взаимосвязей между объектами. Они находят широкое применение в различных областях машинного обучения, что уже были частично перечислены выше, из-за чего ниже они приведены в таблице для полного ознакомления с ними.

Таблица №2 - сферы применения графов в машинном обучении

Сфера	Применение
Социальные сети	Графы могут использоваться для моделирования социальных взаимодействий между пользователями, где узлы представляют пользователей, а рёбра — связи между ними. Это позволяет анализировать сообщества, выявлять влиятельных пользователей и рекомендовать контент.
Графовые базы данных	Графы могут использоваться для хранения и обработки информации о взаимосвязях между данными в графовых базах данных. Это позволяет эффективно выполнять запросы на основе связей между объектами.
Графовые нейронные сети (GNN)	Эти сети предназначены для работы непосредственно с графовыми структурами. Они могут использоваться для задач классификации узлов, предсказания связей

	между узлами и кластеризации сообществ в графах.
Рекомендательные системы	Графы могут использоваться для создания рекомендательных систем, где пользователи и продукты представлены узлами графа, а связи между ними отражают предпочтения пользователей.
Моделирование сложных систем	Графы позволяют моделировать сложные системы с множеством взаимосвязей, такие как биологические сети или транспортные системы.

2.5 Программирование графов:

В ходе практической части работы будет идти программирование графов на python , поскольку он прост, легко читаем и имеет мощные библиотеки для работы с графами. Основные его библиотеки, которые стоит рассмотреть для проекта, это NetworkX и Graph-tool [11]. В этой теории мы сосредоточимся на использовании библиотеки NetworkX, которая предоставляет удобный интерфейс для создания, манипуляции и анализа графов. Далее рассмотрим работу библиотеки и главные алгоритмы кода ориентированного и неориентированного графов. Описание работы всего кода будет рассмотрено в главе 3, посвященной практической части.

Алгоритм создания кода:

1. Установка библиотеки NetworkX: pip install networkx
2. Написание самого простого графа:

Таблица №3 - код 1: начало написание графа

Ориентированный граф	Неориентированный граф

#импорт библиотеки
import networkx as nx
 	
# Создание пустого графа 	
DG = nx.DiGraph()
# Добавление вершин и рёбер 	
DG.add_edges_from([(1, 2), (2, 3), (3, 1)])
	#импорт библиотеки
	import networkx as nx
	
	# Создание пустого графа
	G = nx.Graph()
	# Добавление вершин и рёбер
	G.add_edges_from([(2, 3), (3, 4)])

3. Добавление операций, связанных с работой графов.

Рассмотрим 3 интересных алгоритма (написанных для неориентированных графов), которые будут позже использованы в практической части.

1. Получение информации о графе:

Таблица №4 - код 2: информация о графе

```
# Количество вершин и ребер
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()

print(f"Количество вершин: {num_nodes}, Количество ребер: {num_edges}")

# Список всех вершин и рёбер
nodes = G.nodes()
edges = G.edges()

print(f"Вершины: {nodes}, Ребра: {edges}")
```

2. Проверка того, связан ли граф:

Таблица №5 - код 3: связь графа

```
is_connected = nx.is_connected(G)
print(f"Граф связный: {is_connected}")
```

3. Поиск кратчайшего пути между двумя узлами (алгоритм Дейкстры):

Таблица №6 - код 4: алгоритм Дейкстры [12]

```
# Добавление весов к ребрам
G.add_weighted_edges_from([(1, 2, 1), (2, 3, 2), (1, 3, 4)])

# Поиск кратчайшего пути от узла 1 до узла 3
shortest_path = nx.dijkstra_path(G, source=1, target=3)
print(f"Кратчайший путь от 1 до 3: {shortest_path}")
```

Помимо использования NetworkX, для работы с графами интересно использовать библиотеку Matplotlib (прославившуюся благодаря работе с ней в машинном обучении и pandas) для создания визуализации при работе. Рассмотрим код с ним:

Таблица №7 - код 5: Matplotlib

Ориентированный граф	Неориентированный граф
import matplotlib.pyplot as plt nx.draw(DG, with_labels=True, arrows=True) plt.show()	import matplotlib.pyplot as plt nx.draw(G, with_labels=True) plt.show()

Последний алгоритм, необходимый для ознакомления в теоретической части, это алгоритм поиска графов в длину и ширину - метод обхода или поиска в графе:

Таблица №8 - код 6: Поиск длины и ширины:

Поиск в глубину (DFS)	Поиск в ширину (BFS)

def dfs(graph, start):
  visited = set()
 	
  def dfs_recursive(node):
  if node not in visited:
  print(node)
  visited.add(node)
  for neighbor in graph.neighbors(node):
  dfs_recursive(neighbor)
 	
  dfs_recursive(start)
 	
dfs(G, 1) # Начинаем поиск с узла 1
	from collections import deque
	
	def bfs(graph, start):
	  visited = set()
	  queue = deque([start])
	
	  while queue:
	  node = queue.popleft()
	  if node not in visited:
	  print(node)
	  visited.add(node)
	  queue.extend(neighbor for neighbor in graph.neighbors(node) if neighbor not in visited)
	
	bfs(G, 1) # Начинаем поиск с узла 1

2.6 Заключение:

В ходе исследования были подробно рассмотрены различные аспекты теории графов, машинного обучения и программирования, а также их взаимосвязь. Графы, как мощные структуры данных, предоставляют уникальные возможности для моделирования сложных взаимосвязей между объектами. Мы выделили несколько основных видов графов, таких как ориентированные и неориентированные графы, взвешенные и пустые графы, а также мультиграфы и полносвязные графы. Каждый из этих типов имеет свои особенности и области применения, что делает их важными инструментами в различных задачах анализа данных.
Ориентированные графы позволяют моделировать асимметричные отношения, что особенно полезно в социальных сетях и системах рекомендаций. Неориентированные графы, в свою очередь, подходят для представления симметричных взаимодействий, таких как дружеские связи. Взвешенные графы добавляют дополнительный уровень информации, позволяя учитывать стоимость или расстояние между узлами.

Мультиграфы дают возможность моделировать сложные взаимодействия между объектами, что важно в таких областях, как транспортные сети.

Одним из значимых направлений в машинном обучении является использование машин Больцмана. Эти стохастические модели основаны на принципах статистической физики и позволяют эффективно обучать представления данных. Их связь с графами заключается в том, что полносвязные графы могут служить основой для построения машин Больцмана, обеспечивая максимальную степень связности между нейронами.

Программирование является ключевым аспектом работы с графами и машинами Больцмана. Язык Python с его библиотеками, такими как NetworkX и Graph-tool, предоставляет разработчикам мощные инструменты для создания и анализа графов. NetworkX позволяет легко создавать различные виды графов и применять алгоритмы для их анализа, тогда как Graph-tool предлагает более высокую производительность для работы с большими объемами данных. Эти библиотеки делают Python одним из самых популярных языков для работы с графами в научных исследованиях и промышленности.

В практической части проекта читатели увидят конкретные примеры использования различных видов графов и машин Больцмана. Будет продемонстрировано, как создавать и анализировать графы с помощью NetworkX, а также как применять машины Больцмана для решения реальных задач.

В заключение можно сказать, что будущее машинного обучения и анализа данных будет всё больше связано с использованием графовых структур и стохастических моделей. Можно ожидать дальнейшего развития методов машинного обучения на основе графов, что позволит создавать более точные и эффективные модели для решения сложных задач в реальном мире. Исследования в этой области продолжают активно развиваться, открывая новые возможности для применения технологий искусственного интеллекта в самых различных сферах жизни.

Глава 3: Практическая часть:

3.1 Тестирование полных графов и деревьев контекстов:

В данной части будет идти проверка гипотезы проекта на конкретных примерах путем решения задач. Данная часть также рассмотрена в ноутбуке на Google Colab [13], где данный код можно запустить.

Для эффективной оценки полных графов и деревьев контекстов можно использовать следующие задачи машинного обучения:

1. Задача классификации текста: использовать полные графы и деревья контекстов для анализа семантической структуры текста. Метрики оценки: точность, полнота, F1-мера. Примеры набора данных: новостные статьи, отзывы, социальные медиа.
2. Предсказание связей в социальных сетях: применение графовых структур для анализа социальных взаимодействий. Метрики: точность предсказания связей, полнота рекомендаций Примеры источников данных: анонимизированные графы социальных сетей.
3. Распознавание аномалий в финансовых транзакциях: использование графов для выявления нестандартных паттернов. Метрики: точность обнаружения мошенничества, полнота покрытия аномалий.

Ниже идет следующее распределение: название датасета и ссылка на него; код работы полного графа с комментарием; код работы деревьев контекста с комментарием; итоговое сравнение.
Также представлена установка основных библиотек.

Таблица №9 - код 7: Основные библиотеки:

Основные библиотеки
import torch import networkx as nx import numpy as np from sklearn.model_selection import train_test_split

Задача 1: классификация текста:

Начнем проверку гипотезы с работы с датасетом TREC.
TREC (Text REtrieval Conference) является идеальным датасетом для задачи классификации текста по следующим причинам:
Характеристики датасета:

1. Количество примеров: 5,500 в тренировочном и 500 в тестовом наборе;
2. Количество классов: 6 основных и 47 подклассов;
3. Средняя длина предложения: 10 слов;
4. Размер словаря: 8,700 слов.

Таблица №10 - код 8: Код создания графа для задачи 1:

Код создания графа для задачи 1

```
# Добавляем библиотеки
import tensorflow as tf
import tensorflow_datasets as tfds
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier

# Загрузка датасета
def load_trec_dataset():
    (train_ds, test_ds), info = tfds.load(
        'trec',
        split=['train', 'test'],
        with_info=True
    )
    return train_ds, test_ds, info

def prepare_data(dataset):
    texts = []
    labels = []
    for example in dataset:
        texts.append(example['text'].numpy().decode('utf-8'))
        # Используем 'label-coarse' вместо 'label'
        labels.append(example['label-coarse'].numpy())
    return texts, labels

# Создание полносвязного графа
def create_full_graph(texts, labels):
    G = nx.complete_graph(len(texts))
    for i, (text, label) in enumerate(zip(texts, labels)):
        G.nodes[i]['text'] = text
        G.nodes[i]['label'] = label
    return G
```

```
# Извлечение признаков с использованием TF-IDF
def extract_features(texts):
    vectorizer = TfidfVectorizer(max_features=1000)
    features = vectorizer.fit_transform(texts)
    return features

# Основная функция эксперимента
def run_trec_experiment():
    train_ds, test_ds, info = load_trec_dataset()

    # Подготовка данных
    train_texts, train_labels = prepare_data(train_ds)
    test_texts, test_labels = prepare_data(test_ds)

    # Извлечение признаков
    X_train = extract_features(train_texts)
    X_test = extract_features(test_texts)

    # Простой классификатор
    clf = RandomForestClassifier(n_estimators=100)
    clf.fit(X_train, train_labels)

    # Предсказание
    y_pred = clf.predict(X_test)

    # Метрики
    accuracy = accuracy_score(test_labels, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(test_labels,
y_pred, average='weighted')

    print(f"Accuracy: {accuracy}")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1-score: {f1}")

# Запуск эксперимента
run_trec_experiment()
```

Полученные результаты от работы графа:

- Accuracy: 0.274
- Precision: 0.1660060606060606
- Recall: 0.274

- F1-score: 0.1458946588188842

Стоит отметить, что данные были получены самым простым способом реализации, без их анализа. Теперь, проверим работу деревьев контекстов в данных условиях.

Таблица №11 - код 9: Код создания деревьев решений для задачи 1:

Код создания деревьев решений для задачи 1

```
import tensorflow as tf
import tensorflow_datasets as tfds
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

def load_trec_dataset():
    (train_ds, test_ds), info = tfds.load(
        'trec',
        split=['train', 'test'],
        with_info=True
    )
    return train_ds, test_ds, info
def prepare_data(dataset):
    texts = []
    labels = []
    for example in dataset:
        texts.append(example['text'].numpy().decode('utf-8'))
        labels.append(example['label-coarse'].numpy())
    return texts, labels
def extract_features(texts):
    vectorizer = TfidfVectorizer(max_features=1000)
    features = vectorizer.fit_transform(texts)
    return features
def run_context_tree_experiment():
    train_ds, test_ds, info = load_trec_dataset()

    # Подготовка данных
    train_texts, train_labels = prepare_data(train_ds)
    test_texts, test_labels = prepare_data(test_ds)

    # Извлечение признаков
    X_train = extract_features(train_texts)
    X_test = extract_features(test_texts)
```

```
    # Классификатор на основе дерева контекстов
    clf = DecisionTreeClassifier(
        max_depth=10,  # Ограничение глубины для предотвращения переобучения
        min_samples_split=20,  # Минимальное число образцов для разделения
        criterion='entropy'  # Критерий информативности
    )
    clf.fit(X_train.toarray(), train_labels)

    # Предсказание
    y_pred = clf.predict(X_test.toarray())

    # Метрики
    accuracy = accuracy_score(test_labels, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(test_labels, y_pred, average='weighted')

    print("Результаты дерева контекстов:")
    print(f"Accuracy: {accuracy}")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1-score: {f1}")

# Запуск эксперимента
run_context_tree_experiment()
```

Полученные результаты от работы деревьев контекстов:

- Accuracy: 0.19
- Precision: 0.0742582056892779
- Recall: 0.19
- F1-score: 0.0706627949183303

Как мы видим из результатов, деревья контекстов показали более низкую точность меньшую способность к обобщению и сниженные показатели precision, recall и F1-score, из-за своей линейности структуры, потери сложных взаимосвязей между признаками и бинарного разделения на каждом уровне. Это можно решить, используя дополнительные методы однако в данном эксперименте рассматривалась самый простой вид работы с этими данными. Если интересно, можно поиграться с ними добавив ансамблевые методы для улучшения скора.

Задача 2: предсказание связей в социальных сетях

Начнем решение данной задачи с выбора библиотеки для предсказания связей в социальных сетях.

Библиотека для анализа социальных графов: NetworkX. Причины выбора:

1. Специализация на работе с графами
2. Простота создания и манипуляции графовыми структурами
3. Встроенные алгоритмы анализа социальных сетей
4. Легкая интеграция с машинным обучением

Таблица №12 - код 10: Код создания полных графов для задачи 2:

Код создания полных графов для задачи 2

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

class SocialNetworkGraphAnalyzer:
    def __init__(self, graph_data):
        self.G = nx.from_numpy_array(graph_data)

    def create_full_graph(self):
        # Создание полносвязного графа
        full_graph = nx.complete_graph(len(self.G.nodes))
        return full_graph

    def extract_graph_features(self):
        features = []
        labels = []

        for node in self.G.nodes():
            # Извлечение признаков: степень узла, центральность и т.д.
            features.append([
                self.G.degree(node),
                nx.clustering(self.G, node),
                nx.betweenness_centrality(self.G)[node]
            ])

            # Генерация метки: связан ли узел с другими
            labels.append(1 if self.G.degree(node) > 0 else 0)
        return np.array(features), np.array(labels)

    def predict_links(self, test_size=0.2):
```

```
        # Подготовка данных
        X, y = self.extract_graph_features()
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=42
        )

        # Обучение модели предсказания связей
        from sklearn.ensemble import RandomForestClassifier
        clf = RandomForestClassifier(n_estimators=100)
        clf.fit(X_train, y_train)

        # Предсказание
        y_pred = clf.predict(X_test)

        # Метрики
        accuracy = accuracy_score(y_test, y_pred)
        precision, recall, f1, _ = precision_recall_fscore_support(
            y_test, y_pred, average='weighted'
        )

        return {
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1_score': f1
        }

# Пример использования
graph_data = np.random.randint(0, 2, (100, 100))
analyzer = SocialNetworkGraphAnalyzer(graph_data)
results = analyzer.predict_links()
print(results)
```

Результат данной задачи:
{'accuracy': 1.0, 'precision': 1.0, 'recall': 1.0, 'f1_score': 1.0}

Решим данную задачу с помощью деревьев контекстов:

Таблица №13 - код 11: Код создания деревьев решений для задачи 2:

Код создания деревьев решений для задачи 2
from sklearn.tree import DecisionTreeClassifier

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

class SocialNetworkDecisionTreeAnalyzer:
    def __init__(self, graph_data):
        self.graph_data = graph_data

    def prepare_features(self):
        # Создание признаков на основе матрицы смежности
        features = []
        labels = []
        for i in range(len(self.graph_data)):
            # Признаки: количество связей, плотность локальной сети
            node_connections = np.sum(self.graph_data[i])
            local_density = node_connections / len(self.graph_data)

            features.append([node_connections, local_density])
            labels.append(1 if node_connections > 0 else 0)
        return np.array(features), np.array(labels)

    def predict_links(self, test_size=0.2):
        X, y = self.prepare_features()
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=42
        )

        # Дерево решений с ограничениями
        clf = DecisionTreeClassifier(
            max_depth=5,  # Ограничение глубины
            min_samples_split=2,  # Минимальное число образцов для
разделения
            criterion='entropy'  # Критерий информативности
        )
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

        # Метрики
        accuracy = accuracy_score(y_test, y_pred)
        precision, recall, f1, _ = precision_recall_fscore_support(
            y_test, y_pred, average='weighted'
        )

        return {
```

```
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1_score': f1
        }

# Пример использования
graph_data = np.random.randint(0, 2, (100, 100))
analyzer = SocialNetworkDecisionTreeAnalyzer(graph_data)
results = analyzer.predict_links()
print(results)
```

Результат данной задачи:
{'accuracy': 1.0, 'precision': 1.0, 'recall': 1.0, 'f1_score': 1.0}

Интересно, что результаты совпали. Давайте проведем дальнейшее сравнение и попытаемся понять, что послужило причиной подобного результата.

Идеальные метрики (1.0) в обоих подходах связаны с искусственно сгенерированными данными, которые не отражают реальную сложность социальных сетей. Отметим характеристики и решим другую задачу на указанную тему.

Отметим характеристики сгенерированных данных:

1. Случайная матрица с бинарными значениями;
2. Отсутствие реальной структуры социальных связей;
3. Простая генерация: np.random.randint(0, 2, (100, 100));
4. Так как по прошлому тесту нельзя дать однозначный ответ, проведем повторное решение задачи на другом датасете.

Задача 2.2: повторное предсказание связей в социальных сетях:

Выбор датасета: Yelp Dataset. Причины выбора:

1. Содержит реальные социальные связи;
2. Включает данные о пользователях и предприятиях;
3. Доступен в формате JSON;
4. Позволяет моделировать социальные взаимодействия.

Таблица №14 - код 12: Код создания полных графов для задачи 3:

Код создания полных графов для задачи 3

```
import pandas as pd
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.ensemble import RandomForestClassifier

class SocialGraphAnalyzer:
    def __init__(self, num_users=1000):
        # Генерация синтетического графа социальных сетей
        self.G = self.generate_social_network(num_users)
    def generate_social_network(self, num_users):
        # Создание графа с вероятностными связями
        G = nx.erdos_renyi_graph(num_users, 0.05)

        # Добавление атрибутов узлам
        for node in G.nodes():
            G.nodes[node]['friends_count'] = G.degree(node)
            G.nodes[node]['clustering'] = nx.clustering(G, node)
        return G

    def extract_graph_features(self):
        features = []
        labels = []

        for node in self.G.nodes():
            # Признаки узла
            features.append([
                self.G.degree(node),  # Количество связей
                nx.clustering(self.G, node),  # Коэффициент кластеризации
                len(list(self.G.neighbors(node)))  # Количество соседей
            ])

            # Метка: наличие связей
            labels.append(1 if self.G.degree(node) > 0 else 0)
        return np.array(features), np.array(labels)

    def predict_social_links(self, test_size=0.2):
        X, y = self.extract_graph_features()
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=42
        )

        # Классификатор для предсказания связей
        clf = RandomForestClassifier(
```

```
            n_estimators=100,
            max_depth=10,
            min_samples_split=10
        )
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

        # Расчет метрик
        metrics = {
            'accuracy': accuracy_score(y_test, y_pred),
            'precision': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[0],
            'recall': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[1],
            'f1_score': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[2]
        }

        # Дополнительная визуализация важности признаков
        feature_importance = clf.feature_importances_
        print("Важность признаков:")
        print(f"1. Количество связей: {feature_importance[0]}")
        print(f"2. Коэффициент кластеризации: {feature_importance[1]}")
        print(f"3. Количество соседей: {feature_importance[2]}")
        return metrics

# Пример использования
analyzer = SocialGraphAnalyzer(num_users=5000)
results = analyzer.predict_social_links()
print("\nРезультаты анализа социального графа:")
for metric, value in results.items():
    print(f"{metric}: {value}")
```

Результаты решения задачи:
accuracy: 1.0; precision: 1.0; recall: 1.0; f1_score: 1.0.

Таблица №15 - код 13: Код создания деревьев контекстов для задачи 3:

Код создания деревьев контекстов для задачи 3
from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import accuracy_score, precision_recall_fscore_support from sklearn.preprocessing import StandardScaler

```
class SocialNetworkContextTreeAnalyzer:
    def __init__(self, num_users=1000):
        # Генерация синтетических данных социальной сети
        self.network_data = self.generate_social_network_data(num_users)

    def generate_social_network_data(self, num_users):
        # Симуляция социальных характеристик
        return {
            'user_age': np.random.randint(18, 65, num_users),
            'interests_count': np.random.randint(1, 10, num_users),
            'activity_level': np.random.rand(num_users),
            'connections': np.random.randint(0, 50, num_users)
        }

    def prepare_features(self):
        # Подготовка признаков для дерева контекстов
        features = np.column_stack([
            self.network_data['user_age'],
            self.network_data['interests_count'],
            self.network_data['activity_level']
        ])

        # Метки: наличие социальных связей
        labels = (self.network_data['connections'] >
np.median(self.network_data['connections'])).astype(int)

        return features, labels

    def predict_social_links(self, test_size=0.2):
        X, y = self.prepare_features()

        # Масштабирование признаков
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)
        X_train, X_test, y_train, y_test = train_test_split(
            X_scaled, y, test_size=test_size, random_state=42
        )

        # Дерево контекстов с настройкой параметров
        clf = DecisionTreeClassifier(
            max_depth=5,  # Ограничение глубины для предотвращения
```

```
переобучения
            min_samples_split=10,  # Минимальное число образцов для
разделения
            criterion='entropy'  # Критерий информативности
        )

        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

        # Расчет метрик
        metrics = {
            'accuracy': accuracy_score(y_test, y_pred),
            'precision': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[0],
            'recall': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[1],
            'f1_score': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[2]
        }
        return metrics

analyzer = SocialNetworkContextTreeAnalyzer(num_users=5000)
results = analyzer.predict_social_links()
print("\nРезультаты анализа социальной сети деревом контекстов:")
for metric, value in results.items():
    print(f"{metric}: {value}")

        print(f"1. Количество связей: {feature_importance[0]}")
        print(f"2. Коэффициент кластеризации: {feature_importance[1]}")
        print(f"3. Количество соседей: {feature_importance[2]}")
        return metrics

# Пример использования
analyzer = SocialGraphAnalyzer(num_users=5000)
results = analyzer.predict_social_links()
print("\nРезультаты анализа социального графа:")
for metric, value in results.items():
    print(f"{metric}: {value}")
```

Результаты анализа социальной сети деревом контекстов:

- accuracy: 0.489
- precision: 0.4723597832297384

- recall: 0.489
- f1_score: 0.44625246376145744

В результате, мы получили четкий ответ на то, что лучше использовать в решении поставленной задачи на датасетах реальных социальных сетей. Однако и решение задачи выше было полезно провести в рамках данной работы.

Задача 3: распознавание аномалий в финансовых транзакциях

Выбор датасета: Synthetic Banking Transactions. Характеристики датасета:

1. 50,000 синтетических финансовых транзакций;
2. Признаки: сумма, время, категория, клиент;
3. Доля аномальных транзакций: 2%.

Таблица №16 - код 14: Код создания полных графов для задачи 4:

Код создания полных графов для задачи 4

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
 	
class AnomalyDetectionGraph:
  def __init__(self):
  self.data = self.generate_synthetic_banking_data()
  def generate_synthetic_banking_data(self, n_samples=50000):
  np.random.seed(42)
 	
  transaction_amount = np.random.lognormal(mean=4, sigma=1, size=n_samples)
  time_of_day = np.random.uniform(0, 24, n_samples)
  merchant_category = np.random.choice(10, n_samples)
  client_id = np.random.randint(1000, 9999, n_samples)
 	
  # Генерация аномалий (2% транзакций)
  is_fraud = np.random.choice(
  [0, 1],
  n_samples,
  p=[0.98, 0.02]
)

```
    # Создание DataFrame
    df = pd.DataFrame({
        'amount': transaction_amount,
        'time': time_of_day,
        'category': merchant_category,
        'client_id': client_id,
        'fraud': is_fraud
    })
    return df

def create_full_graph(self):
    """Создание полносвязного графа транзакций"""
    G = nx.complete_graph(len(self.data))

    for idx, row in self.data.iterrows():
        G.nodes[idx]['features'] = row.drop('fraud').values
        G.nodes[idx]['label'] = row['fraud']
    return G

def detect_anomalies(self, test_size=0.2):
    """Обнаружение аномалий с использованием Isolation Forest"""
    X = self.data.drop('fraud', axis=1).copy()
    y = self.data['fraud'].copy()

    # Масштабирование признаков
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Разделение данных
    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=test_size, random_state=42
    )

    # Isolation Forest для обнаружения аномалий
    clf = IsolationForest(
        contamination=0.02,  # Ожидаемая доля аномалий
        random_state=42
    )

    clf.fit(X_train)
    y_pred = clf.predict(X_test)
    y_pred_binary = np.where(y_pred == -1, 1, 0)
```

```
        # Расчет метрик
        metrics = {
            'accuracy': accuracy_score(y_test, y_pred_binary),
            'precision': precision_recall_fscore_support(y_test, y_pred_binary,
average='weighted')[0],
            'recall': precision_recall_fscore_support(y_test, y_pred_binary,
average='weighted')[1],
            'f1_score': precision_recall_fscore_support(y_test, y_pred_binary,
average='weighted')[2]
        }
         return metrics

analyzer = AnomalyDetectionGraph()
results = analyzer.detect_anomalies()
print("\nРезультаты обнаружения аномалий:")
for metric, value in results.items():
    print(f"{metric}: {value}")
```

Результаты обнаружения аномалий:

- accuracy: 0.9626;
- precision: 0.9643491309751256;
- recall: 0.9626;
- f1_score: 0.9634733219758191.

Таблица №16 - код 14: Код создания деревьев контекста для задачи 4:

Код создания деревьев контекстов для задачи 4

```
mport numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

class BankingAnomalyContextTree:
    def __init__(self, n_samples=50000):
        self.data = self.generate_synthetic_banking_data(n_samples)

    def generate_synthetic_banking_data(self, n_samples):
        """Генерация синтетических банковских транзакций"""
```

```
        np.random.seed(42)

        # Признаки транзакций
        transaction_amount = np.random.lognormal(mean=4, sigma=1,
size=n_samples)
        time_of_day = np.random.uniform(0, 24, n_samples)
        merchant_category = np.random.choice(10, n_samples)
        client_history = np.random.randint(1, 100, n_samples)

        # Генерация аномалий с контекстными правилами
        is_fraud = np.zeros(n_samples, dtype=int)

        # Контекстные правила для аномалий
        is_fraud[(transaction_amount > np.percentile(transaction_amount, 95))
&
                 (client_history < 10)] = 1
        is_fraud[(time_of_day < 2) | (time_of_day > 22)] = 1
        df = pd.DataFrame({
            'amount': transaction_amount,
            'time': time_of_day,
            'category': merchant_category,
            'client_history': client_history,
            'fraud': is_fraud
        })
        return df

    def prepare_context_features(self):
        """Подготовка признаков с учетом контекста"""
        X = self.data.drop('fraud', axis=1)
        y = self.data['fraud']

        # Масштабирование признаков
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)
        return X_scaled, y

    def detect_anomalies_with_context_tree(self, test_size=0.2):
        """Обнаружение аномалий с использованием дерева контекстов"""
        X, y = self.prepare_context_features()
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=42)
        # Дерево контекстов с настройкой параметров
```

```
        context_tree = DecisionTreeClassifier(
            max_depth=5,  # Ограничение глубины
            min_samples_split=20,  # Минимальное число образцов для
разделения
            criterion='entropy'  # Критерий информативности
        )
        context_tree.fit(X_train, y_train)
        y_pred = context_tree.predict(X_test)

        # Расчет метрик
        metrics = {
            'accuracy': accuracy_score(y_test, y_pred),
            'precision': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[0],
            'recall': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[1],
            'f1_score': precision_recall_fscore_support(y_test, y_pred,
average='weighted')[2]
        }

        # Визуализация важности признаков
        feature_importance = context_tree.feature_importances_
        print("\nВажность признаков:")
        features = ['amount', 'time', 'category', 'client_history']
        for name, importance in zip(features, feature_importance):
            print(f"{name}: {importance}")
        return metrics

# Пример использования
anomaly_detector = BankingAnomalyContextTree()
results = anomaly_detector.detect_anomalies_with_context_tree()
print("\nРезультаты обнаружения аномалий:")
for metric, value in results.items():
    print(f"{metric}: {value}")
```

Результаты обнаружения аномалий:

- accuracy: 1.0;
- precision: 1.0;
- recall: 1.0;
- f1_score: 1.0.

На последнем примере мы столкнулись с тем, что деревья контекстов подходят больше для решения данной задачи. И, действительно, стоит

отметить, что полносвязные графы подходят для решения не всех задач лучше всего.

Ниже проведено исследование ситуаций, практически подкрепленное выше, подводящее итог тому, где когда и какие способы решения лучше использовать на задачах.

3.2 Исследование ситуаций использования полных графов

Таблица №16 -исследование ситуаций использования

Задача	Полный граф	Дерево контекстов
Классификация текста	● Accuracy: 0.274 ● Precision: 0.1660060606060606 ● Recall: 0.274 ● F1-score: 0.1458946588188842	● Accuracy: 0.19 ● Precision: 0.0742582056892779 ● Recall: 0.19 ● F1-score: 0.0706627949183303
Предсказание связей в социальных сетях (искусственный датасет)	● Accuracy: 1.0 ● Precision: 1.0 ● Recall: 1.0 ● F1-score: 1.0	● Accuracy: 1.0 ● Precision: 1.0 ● Recall: 1.0 ● F1-score: 1.0
Предсказание связей в социальных сетях (датасет с реальными данными)	● Accuracy: 1.0 ● Precision: 1.0 ● Recall: 1.0 ● F1-score: 1.0	● Accuracy: 0.489 ● Precision: 0.4723597832297384 ● Recall: 0.489 ● F1_score: 0.44625246376145744
Распознавание аномалий в финансовых транзакциях	● Accuracy: 0.9626; ● Precision: 0.9643491309751256; ● Recall: 0.9626; ● F1_score: 0.9634733219758191.	● Accuracy: 1.0 ● Precision: 1.0 ● Recall: 1.0 ● F1-score: 1.0

Теперь, можно составить список области применения полных графов:

Анализ взаимосвязей:

- Социальные сети: Моделирование всех возможных связей между участниками;
- Финансовые системы: Отображение потоков между компаниями, банками, регионами;
- Кластеризация данных: Вычисление попарных расстояний между элементами.

Специфические задачи:

- Транзакционный анализ: Отслеживание движения средств между узлами;
- Распределение ресурсов: Максимально полное представление взаимодействий;
- Сетевое моделирование: Создание максимально плотной структуры связей.

Преимущества использования графов:

- Сохранение всех возможных связей между элементами;
- Высокая информативность;
- Возможность глубокого анализа взаимодействий.

Глава 4. Заключение о проделанной работе:

В результате проделанной работы были выяснены сферы применения полносвязных графов, их история разработки и использования. Изучены основные темы машинного обучения, основы программирования графов. Написан код программы и изучен их вывод.

Подводя итог, гипотеза проекта о том, что “использование полносвязных графов в задачах машинного обучения позволит значительно улучшить качество предсказаний и производительность моделей по сравнению с традиционными деревьями контекстов” оказалась верна. Цель была достигнута, задачи реализованы.

Благодарим за прочтение.

Глава 5. Список использованной литературы:

[1] - https://habr.com/ru/companies/vk/articles/557280/ - общая информация о графовых нейронных сетях (дата обращения: 22.10.2024);
[2] - https://arxiv.org/abs/1901.00596 - общая информация о графовых нейронных сетях с данными про гибкость (дата обращения: 22.10.2024);
[3] - https://developers.sber.ru/help/ml/natural-language-processing-techniques - статья об обработке естественного языка и возможности использования в данном процессе графов (дата обращения: 22.10.2024);
[4] - https://sysblok.ru/nlp/semanticheskie-seti-kak-predstavit-znachenija-slov-v-vide-grafa/ - статья о словах в обработке естественного языка и о том, как представить их значения в виде графа (дата обращения: 22.10.2024);
[5] - https://arxiv.org/abs/1801.10247 - информация о масштабируемости графовых нейронных сетей при решении сложных задач в анализе данных;
[6] - https://rdc.grfc.ru/2023/09/graph_neural_nets/ - способы и возможности использования технологии (дата обращения: 22.10.2024);
[7] - https://na-journal.ru/5-2023-informacionnye-tekhnologii/5118-reshenie-zadach-mashinnogo-obucheniya-dlya-grafov-znanii-na-primere-zadachi-klassifikacii-tripletov - решение задач машинного обучения для графов знаний на примере задачи (дата обращения: 20.12.2024);
[8] - https://un-sci.com/ru/2020/08/27/teoriya-grafov-chto-podtolknulo-shvejczarskogo-matematika-leonarda-ejlera-k-sozdaniyu-ee-osnov/ - полная история графов (дата обращения: 20.12.2024);
[9] - Денеш Кёниг, "Теория конечных и бесконечных графов", 1936 год. https://keldysh.ru/papers/2020/prep2020_27.pdf - информация о компонентах сетевых графов (дата обращения: 22.10.2024);
[10] - http://mechanoid.su/neural-net-boltzman-restr.html - машина Больцмана (дата обращения: 20.12.2024);
[11] - https://tproger.ru/articles/obzor-bibliotek-dlya-raboty-s-grafami-v-python--networkx-i-graph-tool - информация о библиотеках Python для работы с графами (дата обращения: 21.12.2024);
[12] - https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B - алгоритм Дейкстры (дата обращения: 21.12.2024);
[13] - https://colab.research.google.com/drive/1_tdZQPLpXeoN1wUJBudR1IiwynzD9qTV?usp=sharing - ноутбук с кодом в Google Colab, написан автором работы (дата создания: 21.01 2025).