МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт Математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Применение алгоритмов Data mining для обработки больших массивов данных»

Студент Н.Г. Забегаев
(И.О. Фамилия) (личная подпись)

Руководитель д.т.н., доцент, С.В. Мкртычев
(ученая степень, звание, И.О. Фамилия)

Консультант М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема выпускной квалификационной работы: «Применение алгоритмов Data mining для обработки больших массивов данных»

Работа была выполнена студентом Тольяттинского Государственного Университета, института математики, физики и информационных технологий, группы ПМИп-1702а, Забегаевым Никитой Глебовичем.

Выпускная квалификационная работа посвящена исследованию существующих алгоритмов Data Mining и их оптимизации для работы с большими массивами данных.

Объект исследования – алгоритмы Data Mining.

Предмет исследования – применение алгоритмов Data Mining для работы с большими объемами данных.

Цель работы – представить концепцию использования алгоритмов Data Mining с целью обработки большого количества данных.

Задачи работы:

– описать математический аппарат алгоритмов Data Mining;
– проанализировать модель программировании MapReduce;
– проанализировать алгоритмы для работы с большими объемами данных;
– выполнить программную реализацию алгоритмов.

Актуальность работы заключается в анализе данных с целью найти ценную информации, где метод решения будет использовать алгоритм работы с большими объемами данных.

Результатом работы является концепция алгоритма с использованием модели MapReduce.

Бакалаврская работа содержит пояснительную записку объемом 46 страниц, включая 32 рисунка, 2 таблицы, 13 формул, список литературы из 20 наименований, включая 7 зарубежных.

Abstract

The topic of this thesis is the Application of Data Mining Algorithms for Processing Large Data Sets.

The study is devoted to solving the problem of optimizing data mining algorithms for working with large amounts of data.

The relevance of the study lies in the analysis of data in order to find valuable information, where the solution method will use an algorithm for working with large amounts of data.

The object of the research is the problem of optimizing Data Mining algorithms for working with large data arrays.

The subject of the research is Data Mining solution algorithms and the MapReduce programming model.

The purpose of the work is to present the concept of using Data Mining algorithms to process a large amount of data.

To achieve the goal, you must complete the following tasks:

– Study theoretical information about Data Mining algorithms;
– Explore theoretical information about the MapReduce programming model;
– Develop algorithms;
– Perform software implementation of the developed algorithm.

Chapter 1 discusses general theoretical information about Data Mining techniques and working with large amounts of data.

Chapter 2 provides an analysis of existing solution methods and the formation of a computational algorithm.

Chapter 3 develops the software and program interface, and compares it with existing solution methods. In the conclusion, the results and conclusions about the work done are presented.

The result of the thesis was the concept of an algorithm using the MapReduce model.

Оглавление

Введение

В наше время процесс сбора данных стал неотъемлемой частью практически всех областей человеческой деятельности. Бизнес, торговля, обучение, медицина – компании в этих и многих других сферах целенаправленно или попутно регистрируют огромное количество самой разнообразной информации – данные о финансах, клиентах, покупках, заказах, перевозках и т.д.

При наличии большого количества данных зачастую возникает проблема их обработки, а также появляется вопрос: возможно ли извлечь из собранных данных новую, нетривиальную и полезную информацию, которую можно было бы использовать в деятельности компании?

С задачей обработки позволяют справиться различные технологии анализа данных, например, OLAP, предназначенной для быстрой обработки сложных запросов к базе данных и служащей для подготовки бизнес-отчётов, например, по продажам и маркетингу.

А на последний вопрос дает ответ технология Data Mining (DM), представляющая собой “набор различных методов и алгоритмов для обнаружения в сырых данных ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности”.

На сегодняшний день есть много готовых решений для анализа данных средствами Data Mining, но почти все они распространяются на коммерческой основе за немалые деньги. При этом существует достаточно много небольших организаций, не готовых платить круглые суммы за мощные аналитические пакеты, но желающих использовать Data Mining в своей деятельности. К тому же, зачастую не требуется полный комплекс средств для анализа, а только один-два алгоритма.

Еще одной трудностью в использовании DM является необходимость наличия аналитика, умеющего работать со средствами анализа, знающего специфику настройки алгоритмов, способного должным образом подготовить

данные. Очевидно, что не все организации способны держать в штате такого сотрудника.

Таким образом, является актуальной задача разработки новых и реализации уже существующих алгоритмов анализа данных для нужд небольших организаций.

Объект исследования – алгоритмы Data Mining.

Предмет исследования – применение алгоритмов Data Mining для работы с большими объемами данных.

Цель работы – представить концепцию использования алгоритмов Data Mining для обработки больших массивов данных.

Задачи работы:

– описать математический аппарат алгоритмов Data Mining;
– исследовать модель программировании MapReduce;
– проанализировать алгоритмы для работы с большими объемами данных;
– выполнить программную реализацию алгоритмов.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка используемых источников.

В главе 1 рассматривается общая теоретическая информация о методах Data Mining и работе с большими объемами данных.

В главе 2 приводится анализ существующих методов решения и формирование вычислительного алгоритма.

В главе 3 разрабатывается программное обеспечение и интерфейс программы, а также проводится сравнительный анализ с существующими методами решения. В заключении представлены результаты и выводы о проделанной работе.

Бакалаврская работа содержит пояснительную записку объемом 46 страниц, включая 32 рисунка, 2 таблицы, 13 формул, список литературы из 20 наименований.

Глава 1 Анализ методов интеллектуальной обработки данных

1.1 Обзор решаемых задач интеллектуального анализа данных

Перед тем, как приступить к работе именно с большими данными, следует разобраться, для чего применять Data mining.

«Data Mining – исследование и обнаружение "машиной" (алгоритмами, средствами искусственного интеллекта) в сырых данных скрытых знаний, которые не были известны, нетривиальны, практически полезны, доступны для интерпретации человеком» [4]

Единого мнения относительно того, какие задачи следует относить к Data Mining нет. Ниже приводятся лишь основные из задач решаемые с помощью Data Mining:

«Классификация – наиболее простая и распространенная задача Data Mining. В результате решения задачи классификации обнаруживаются признаки, которые характеризуют группы объектов исследуемого набора данных - классы; по этим признакам новый объект можно отнести к тому или иному классу» [3].

Методы решения. Для решения задачи классификации могут использоваться методы: ближайшего соседа (Nearest Neighbor); k-ближайшего соседа (k-Nearest Neighbor); байесовские сети (Bayesian Networks); индукция деревьев решений; нейронные сети (neural networks)

«Кластеризация является логическим продолжением идеи классификации.

Это задача более сложная, особенность кластеризации заключается в том, что классы объектов изначально не предопределены. Результатом кластеризации является разбиение объектов на группы» [15].

Пример метода решения задачи кластеризации: обучение "без учителя" особого вида нейронных сетей - самоорганизующихся карт Кохонена.

«Ассоциация – в ходе решения задачи поиска ассоциативных правил отыскиваются закономерности между связанными событиями в наборе данных.

Отличие ассоциации от двух предыдущих задач Data Mining: поиск закономерностей осуществляется не на основе свойств анализируемого объекта, а между несколькими событиями, которые происходят одновременно» [11].

Наиболее известный алгоритм решения задачи поиска ассоциативных правил – алгоритм Apriori.

«Последовательность позволяет найти временные закономерности между транзакциями.

Задача последовательности подобна ассоциации, но ее целью является установление закономерностей не между одновременно наступающими событиями, а между событиями, связанными во времени (т.е. происходящими с некоторым определенным интервалом во времени). Другими словами, последовательность определяется высокой вероятностью цепочки связанных во времени событий.

Фактически, ассоциация является частным случаем последовательности с временным шагом, равным нулю.

Эту задачу Data Mining также называют задачей нахождения последовательных шаблонов (sequential pattern). Правило последовательности: после события X через определенное время произойдет событие Y.» [11].

«Прогнозирование. В результате решения задачи прогнозирования на основе особенностей исторических данных оцениваются пропущенные или же будущие значения целевых численных показателей» [14].

Для решения таких задач широко применяются методы математической статистики, нейронные сети и др.

1.2 Выбор методов для реализации

1.2.1 Алгоритм Apriori

Apriori — масштабируемый алгоритм поиска ассоциативных правил. Благодаря использованию свойства анти-монотонности, он способен обрабатывать большие объемы данных за приемлемое время. В основе алгоритма Apriori лежит понятие частого набора (frequent itemset), который также, можно назвать частым предметным набором, часто встречающимся множеством (соответственно, он связан с понятием частоты). Под частотой понимается простое количество транзакций, в которых содержится данный предметный набор. Тогда частыми наборами будут те из них, которые встречаются чаще, чем в заданном числе транзакции.

Алгоритм Apriori довольно прост и понятен, а также легок в реализации в отличии от его предшественников AIS (Agwarawal, Imielinski and Swami) и SETM. Также алгоритм имеет множество вариаций и модификаций, что подчеркивает его актуальность в настоящее время.

Свойство анти-монотонности гласит следующее, что если имеется некий редкий предметный набор, то добавление к этому предметному набору еще одного предмета не делает его более частым. Алгоритм Apriori состоит из двух этапов:

– поиск частых наборов;
– генерация на основе найденных частых наборов ассоциативных правил.

Анализ ассоциативных правил - это метод, позволяющий выявить, как элементы связаны друг с другом. Чтобы сократить пространство поиска ассоциативных правил, алгоритм Apriori использует свойство анти-монотонности. Свойство утверждает, что если предметный набор Z не является частым, то добавление некоторого нового предмета A к набору Z не

делает его более частым. Другими словами, если Z не является частым набором, то и набор $A \in Z$ также не будет являться таковым. Данное полезное свойство позволяет значительно уменьшить пространство поиска ассоциативных правил.

Для сравнения правил, получаемых при генерации ассоциативных правил, используются следующие числовые характеристики:

Поддержка (support) - количество вхождений набора предметов в транзакции, деленное на общее количество транзакций.

$$Support(X) = |\{t \in T; X \subseteq t\}||T|, \quad (1)$$

где:

X– itemset, содержащий в себе i – items;

T – количество транзакций.

Доверие (confidence) говорит о том, насколько вероятно, что предмет Y будет куплен при покупке предмета X, выраженный как $\{X \to Y\}$. Это измеряется долей транзакций с товаром X, в котором также присутствует товар Y.

$$Confidence(X \to Y) = Support(X \cup Y)Support(X) \quad (2)$$

где:

$X \cup Y$ – транзакции с наборами X и Y

Подъем (lift) - отношение наблюдаемой поддержки к математическому ожиданию события, если бы X и Y были бы независимы. Говорит о том, насколько вероятно, что предмет Y будет куплен при покупке предмета X, при этом контролируется, насколько популярен предмет Y.

$$Lift(X \rightarrow Y) = Support(X \cup Y)Support(X) * Support(Y) \quad (3)$$

Убеждение (conviction) - отношение ожидаемой частоты, что X встречается без Y, и произведения частот их отдельных появлений, если бы они были независимы. Говоря иначе, частота неправильного предсказания.

$$Conviction(X \rightarrow Y) = 1 - Support(Y)1 - Support(X \rightarrow Y) \quad (4)$$

Использование (leverage) - разница между ожидаемой частотой появления X и Y вместе и частотой, если бы X и Y были независимы.

$$Leverage(X \rightarrow Y) = Support(X \rightarrow Y) - Support(X) * Support(Y) \quad (5)$$

1.2.2 Алгоритм k-means

Алгоритм k-means создает k – групп их набора объектов таким образом, чтобы члены группы были наиболее однородными. Это популярная техника кластерного анализа для исследования набора данных.

«Элементом множества может быть что угодно, например, данные или вектора характеристик. Сами же группы принято называть кластерами» [8]

Основным достоинством алгоритма является простота. Простота обычно означает высокую скорость выполнения и эффективность по сравнению с другими алгоритмами, в особенности при работе с крупными наборами данных.

Алгоритм k-means состоит из следующих этапов:

Шаг 1. Выбираются k начальных центроидов из исходной выборки данных.

Шаг 2. Находится расстояние от каждой записи исходной выборки данных до всех центроидов по выбранной метрике, выбирают наименьшее

расстояние и причисляют данную запись к тому кластеру, расстояние до которого было наименьшим.

Метрика – формула вычисления расстояния в многомерном пространстве между двумя объектами. Наиболее распространённые метрики:

1. Евклидова метрика $||x||2 = \sqrt{x12 + x22 + \cdots + xn2}$ (6)
2. Расстояние Манхэттена $||x||l = |x1| + |x2| + \cdots + |xn|$
3. Расстояние Чебышева $||x||\infty = max(|x1|, |x2|, \dots, |xn|)$

где x_i – разница между параметрами двух записей

Например, возьмём двумерную плоскость и две точки A, B. У них есть координаты X и Y. Тогда Евклидова метрика будет иметь вид:

$$\sqrt{(xa - xb)2 + (ya - yb)2} \quad (7)$$

Шаг 3. Вычисляются новые центроиды. Каждый параметр нового центроида рассчитывается как среднее арифметическое среди всех записей данного кластера. Если параметр является категориальным, то есть не имеет числовые значения, то выбирается то значение, которое встречалось наиболее часто.

Шаг 4. Шаги 2 и 3 повторяются до тех пор, пока не будет выполнен один из следующих критериев:

Пока границы кластеров и расположения центроидов не перестанут изменяться от итерации к итерации, т.е. на каждой итерации в каждом кластере будет оставаться один и тот же набор записей.

Достигнут критерий сходимости:

$$Ei - Ei + 1 < \varepsilon \quad (8)$$

Где E – сумма квадратов ошибок, i – номер итерации, ε – заданная

точность

$$E = \sum\sum(p - mi)2p \in Ciki = 1 \quad (9)$$

Где p - произвольная запись, принадлежащая кластеру C_i, m_i – центроид данного кластера.

Иными словами, алгоритм остановится тогда, когда ошибка E достигнет достаточно малого значения.

1.2.3 Алгоритм KNN

Алгоритм KNN (K – Nearest Neighbors) – метод ближайших соседей один из самых простых, но в тоже время универсальных алгоритмов, использующийся как для решения задач классификации, так и восстановления регрессии

«Метод k ближайших соседей ($k\,nearest - neighbor,\ k - NN$) относится к наиболее простым и в то же время универсальным методам, используемым как для решения задач классификации, так и восстановления регрессии» [10]

Алгоритм KNN называют ленивым классификатором, так как в отличии от других «активных» алгоритмов он не строит классификационную модель в процессе обучения.

«В случае классификации новый объект классифицируется путем отнесения его к классу, являющемуся преобладающим среди k ближайших (в пространстве признаков) объектов из обучающей выборки. Если $k = 1$, тоновый объект относится к тому же классу, что и ближайший объект из обучающей выборки. » [5]

Аналогичный способ используется и в задаче восстановления регрессии, с той лишь разницей, что в качестве ответа для объекта выступает среднее ответов k ближайших к нему объектов из обучающей выборки.

«Данный алгоритм лучше справляется с несферическими или несвязными классами, чем алгоритм Роккио, поскольку определяет границы между классами локально» [1]

Опишем метод более формально. Пусть $Nk(x)$ множество k ближайших к x объектов из обучающей выборки. Тогда для задачи классификации положим

$$f(x) = argmin \left| \{i : y^i = y, x^{(i)} \in Nk(x)\} \right| \quad (10)$$

«В некоторых случаях данные ответы учитываются с весами, обратно пропорциональными расстоянию до объекта. Это особенно полезно для решения задачи классификации с несбалансированными данными, т. е. когда число объектов, относящихся к разным классам, сильно различно.» [19]

Для определения ближайших соседей обычно используется евклидово расстояние

$$\text{Евклидова метрика } \|x\|2 = \sqrt{x12 + x22 + \cdots + xn2} \quad (11)$$

Однако оно применимо только для признаков, описываемых количественными переменными.

«Если все переменные качественные, то можно использовать расстояние Хэмминга» [17]

$$p_j(x, x') = \sum_{j=1}^{d} I\,(x_j \neq x'_j) \quad (12)$$

В общем случае используют функцию

$$p_j(x, x') = \sum_{j=1}^{d} a_j p_j\,(x_j, x'_j) \tag{13}$$

Где a^j- неотрицательные параметры,

$p_j(x, x') = (x_j - x'_j)^2$для качественных переменных,

$p_j(x, x') = I(x_j \neq x'_j)^2$ для качественных переменных.

«Заметим, что функция расстояния не обязательно должна быть метрикой и неравенство треугольника может быть не выполнено.» [20].

Для повышения точности модели также могут использоваться специальные алгоритмы обучения метрики расстояния (например, Large Margin Nearest Neighbour).

«Одним из основных параметров, влияющих на обобщающую способность алгоритма, является число "ближайших соседей" k. В целом, выбор определенного значения обусловлен характером данных задачи. Большие значения k могут привести как к более точному описанию границы, разделяющей классы, так и переобучению. Обычно для выбора k применяют различные эвристики, в частности, метод перекрестного контроля» [10].

Выводы к главе 1

Проведен анализ и сравнение некоторых алгоритмов Data mining для выбора наиболее оптимальных алгоритмов для работы с большими данными.

Как показал анализ, алгоритм Apriori довольно прост и понятен, а также легок в реализации в отличии от его.

Также данный алгоритм имеет множество вариаций и модификаций, что подчеркивает его актуальность в настоящее время.

Глава 2 Разработка алгоритмов с помощью MapReduce

2.1 Модель обработки больших данных MapReduce

«Из-за огромного количества информации очень малая ее часть будет когда-либо увидена человеческим глазом. Наша единственная надежда понять и найти что-то полезное в этом океане информации — широкое применение методов Data Mining» [9].

MapReduce – это паттерн программирования, предназначенный обработки больших данных с помощью алгоритма параллельных вычислений.

«Одним из самых эффективных методов обработки больших объемов данных в распределенных средах является парадигма MapReduce, предложенная компанией Google в начале 2000-х для сканирования и обработки большого количества страниц из сети Интернет. Впервые такая парадигма была реализована в составе распределенной файловой системы GFS (Google File System) и в высоко производительной нереляционной базе данных Big Table»

Данная модель отличается простотой и удобством использования, скрывает от пользователя детали организации вычислений на кластерной системе. Преимущество MapReduce заключается в том, что она позволяет распределенно выполнять операции предварительной обработки и свертки. Операции предварительной обработки работают независимо друг от друга и могут производиться параллельно.

«Параллелизм также дает некоторые возможности восстановления после частичных сбоев серверов: если в рабочем узле, производящем операцию предварительной обработки или свертки, возникает сбой, то его работа может быть передана другому рабочему узлу (при условии, что входные данные для проводимой операции доступны). Пользователю достаточно описать процедуру обработки данных в виде нескольких функций, после чего система автоматически распределяет вычисления по кластеру,

обрабатывает отказы машин, балансирует нагрузку и координирует взаимодействия между машинами» [6]

«В рамках концепции MapReduce предполагается, что данные организованы в виде некоторого набора упорядоченных записей, а их обработка происходит в три стадии: Map, Combine, Shuffle и Reduce (рисунок 1).

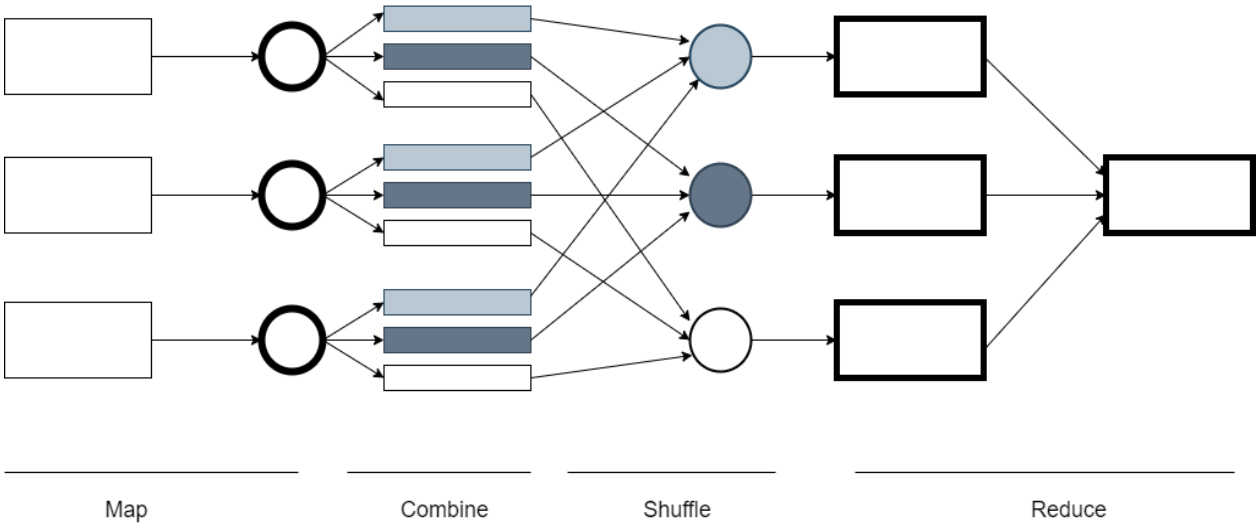

Рисунок 1 – Стадии работы MapReduce

«Стадия Map. На этой стадии выполняется предварительная обработка и фильтрация данных при помощи функции Map, которую определяет пользователь. Принцип работы подобен операции Map в функциональных языках программирования – пользовательская функция применяется к каждой входной записи и возвращает множество пар ключ – значение. Все запуски функции работают независимо и могут работать параллельно, в том числе на разных машинах кластера. Функция Map, как правило, применяется на той же машине, на которой хранятся данные. Это позволяет снизить передачу данных по сети (принцип локальности данных).

Стадия Shuffle. На этой стадии вывод функции Map разбивается на специальные секции (корзины). Каждая корзина соответствует одному ключу вывода стадии Map. Кроме того, она принимает на вход совокупность записей,

соответствующих данному ключу, и общее количество Reduce-задач, а возвращаемым значением является номер задачи, в которой обрабатывалась каждая запись» [6].

Каждая секция формируется на основе функции хеширования, которая вызывается для каждого ключа и зависит от определенных критериев, например, от номера задачи. Для ускорения процесса обработки информации очень часто на данной стадии применяют алгоритмы параллельной сортировки. В первую очередь они требуются в тех случаях, когда разные атомарные обработчики возвращают наборы с одинаковыми ключами, при этом правила сортировки на этой фазе могут быть заданы программно и использовать какие-либо особенности внутренней структуры ключей разделения

«Стадия Reduce. Каждая корзина со значениями, сформированная на стадии Shuffle, попадает на вход функции Reduce. Эта функция вычисляет финальный результат для каждой отдельной секции. Все запуски Reduce, как и функция Map, работают независимо и могут работать параллельно, в том числе на разных машинах кластера. Для некоторых видов обработки свертка не требуется, и каркас возвращает в этом случае набор отсортированных пар, полученных базовыми обработчиками» [6].

«Стадия Combine применяется в тех случаях, когда в результатах функции Map содержится значительное число повторяющихся значений промежуточного ключа, а определенная пользователем задача Reduce является коммутативной и ассоциативной. В таких случаях необходимо осуществить частичную агрегацию данных до их передачи по сети. Функция Combine выполняется на той же машине, что и задача Map. Результаты функции Combine помещаются в промежуточные файлы, которые впоследствии пересылаются в задачи Shuffle или Reduce» [6].

2.2 Разработка алгоритмов с помощью MapReduce

2.2.1 Разработка алгоритма априори

Модель алгоритма представлена на рисунке 2.

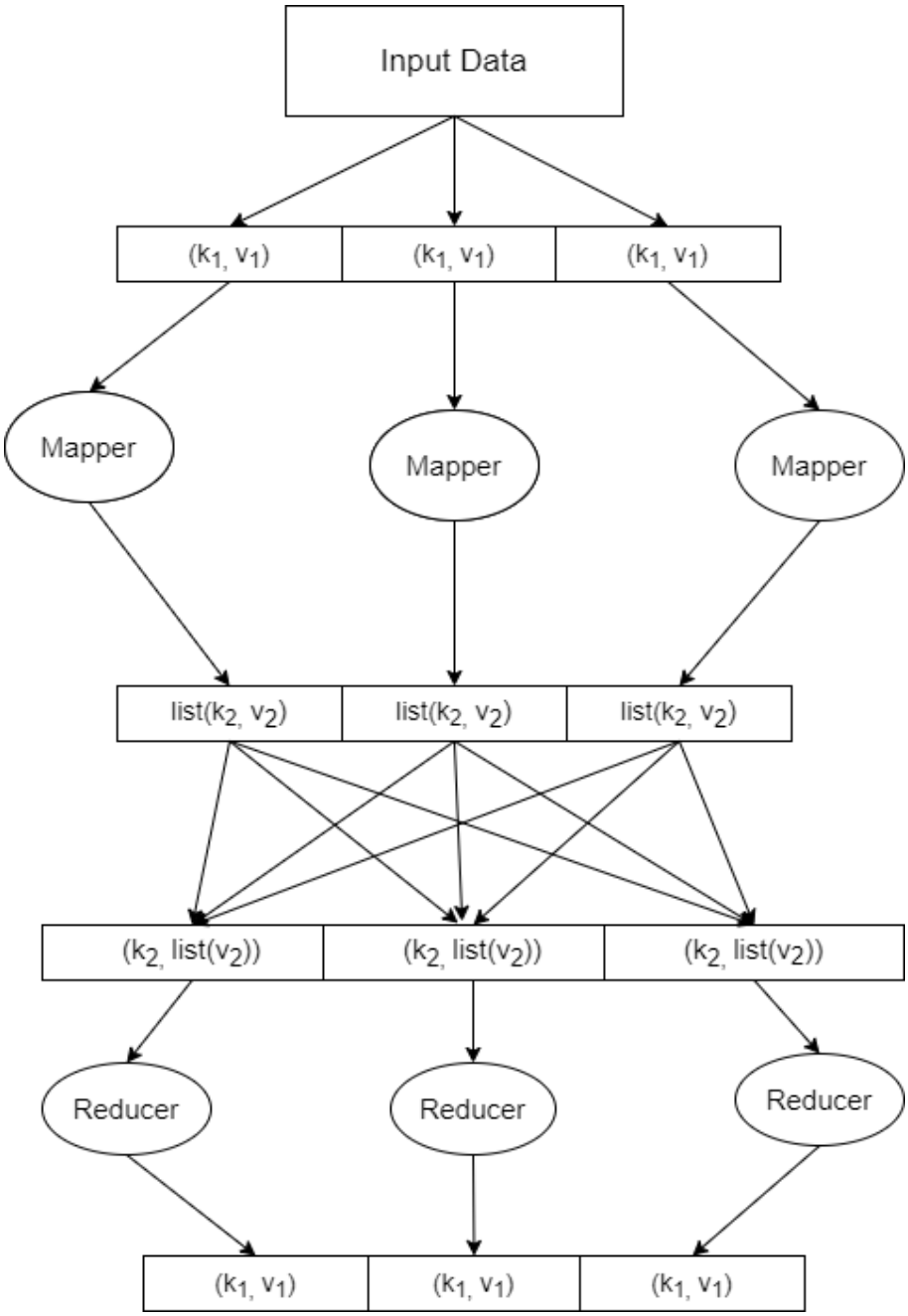

Рисунок 2 – Модель алгоритма

Для реализации алгоритма на модели MapReduce основными задачами являются создание двух независимых функций map и reduce для алгоритма, а также преобразование наборов данных в виде пар (ключ, значение). В программировании MapReduce все преобразователи и редукторы на разных машинах выполняются параллельно, но окончательный результат получается только после завершения reduce. Если алгоритм рекурсивный, то мы должны выполнить несколько этапов сокращения карты, чтобы получить окончательный результат.

«Априорный алгоритм является итеративным процессом, и его двумя основными компонентами являются генерация наборов элементов-кандидатов и генерация частых наборов элементов» [7].

При каждом сканировании базы данных mapper генерирует локальных кандидатов, а reducer суммирует локальный счетчик и выдает частые наборы элементов. Параллельная версия Apriori с распределением счетчиков лучше всего подходит для MapReduce, тогда как для реализации алгоритма распределения данных мы должны контролировать распределение данных.

Первым шагом алгоритма является генерация частых наборов из 1 элемента $L1$, что показано на рисунке 3 в качестве примера. База данных транзакций разбивается на блоки и распределяется среди всех мапперов. Каждая транзакция преобразуется в пары (ключ, значение), где ключ - это TID, а значение - это список элементов, то есть транзакция. Mapper считывает одну транзакцию в пары ime и $output$ (ключ , значение), где ключ - это каждый элемент транзакции, а значение’ - 1. Combiner объединяет пары с одним и тем же ключом и составляет локальную сумму значений для каждого ключа. Выходные пары всех объединителей перемешиваются и обмениваются, чтобы составить список значений, связанных с тем же ключом, как пары (ключ, список (значение)) (рисунок 3).

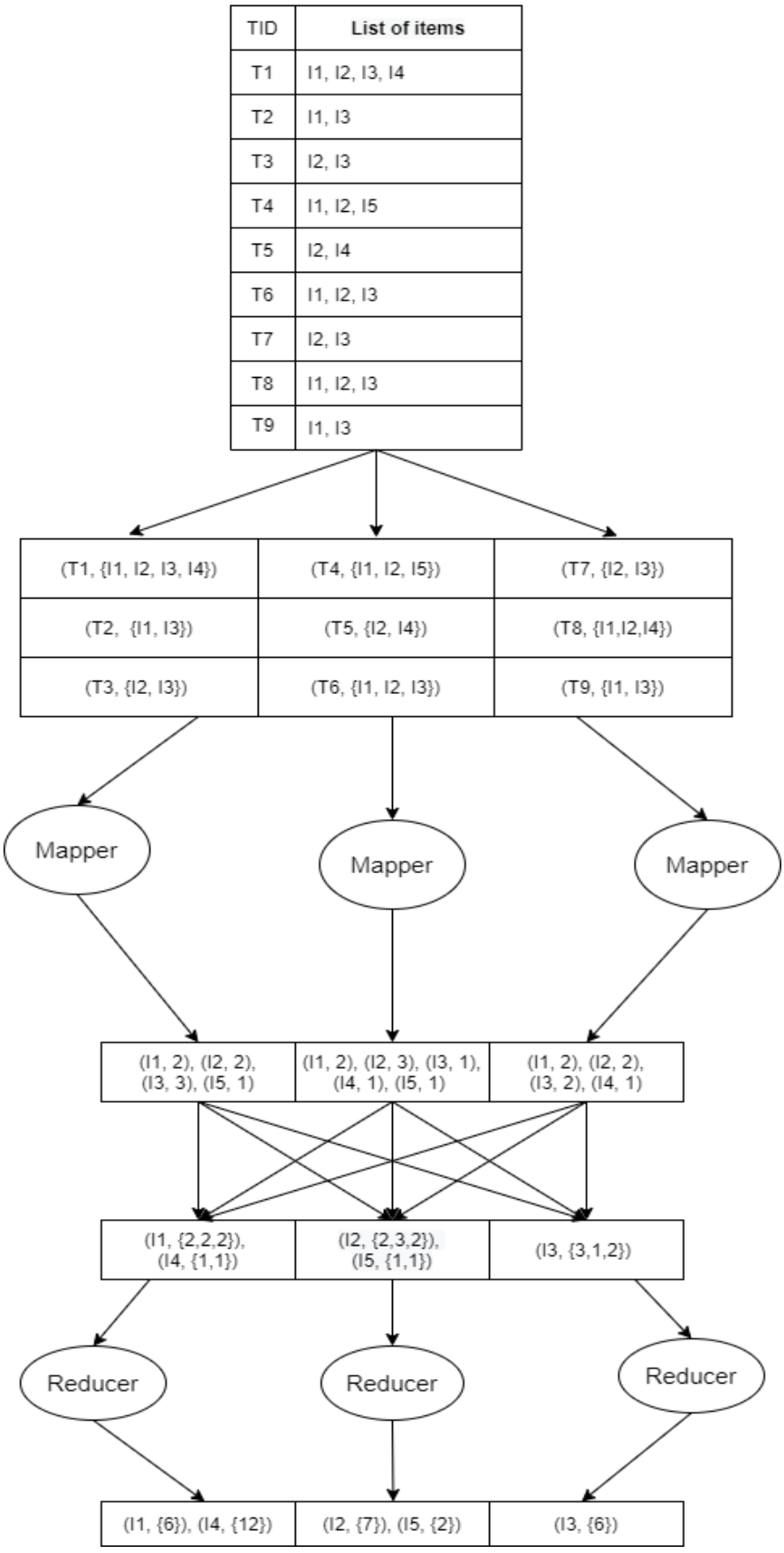

Рисунок 3 – Диаграмма алгоритма априори на основе модели MapReduce

Reducer берут эти пары и суммируют значения соответствующих ключей. Reducer выводят пары (ключ, значение), где ключ - это элемент, а значение – это количество поддерживаемых элементов меньше или равно минимальной поддержки этого элемента. Конечные частые наборы из 1 элемента $L1$ получаются путем объединения выходных данных всех reducer.

Чтобы сгенерировать частные пары из $k - itemsets\ Lk$, каждый mapper читает частные наборы элементов $Lk - 1$ из предыдущей итерации и генерирует наборы элементов кандидатов Ck из $Lk - 1$ как в традиционном алгоритме. Набор элементов-кандидатов в Ck выбирается в качестве ключа и ему присваивается значение 1, если он присутствует в транзакции, назначенной мапперу. Сейчас у нас есть пары (ключ, значение), где ключ - это $k - itemset$, а значение - 1. Все остальные процедуры являются то же, что и при генерации $L1$.

2.2.2 Разработка алгоритма k-means

Классический алгоритм k-средних работает как итерационный процесс, в котором на каждой итерации он вычисляет расстояние между точками данных и центроидами, которые случайным образом инициализируются в начале алгоритма.

«Кластеризация – это автоматическое разбиение элементов некоторого множества на группы в зависимости от их схожести» [2].

Кластеризация схожа по своим задачам на классификацию, однако у них имеются отличия.

«Кластеризация разбивает множество объектов на группы, которые определяются только ее результат. Классификация относит каждый объект к одной из заранее определенных групп» [6].

Как показано на рисунке 4, программа MapReduce принимает набор точек и начальные центроиды кластера в качестве входных данных.

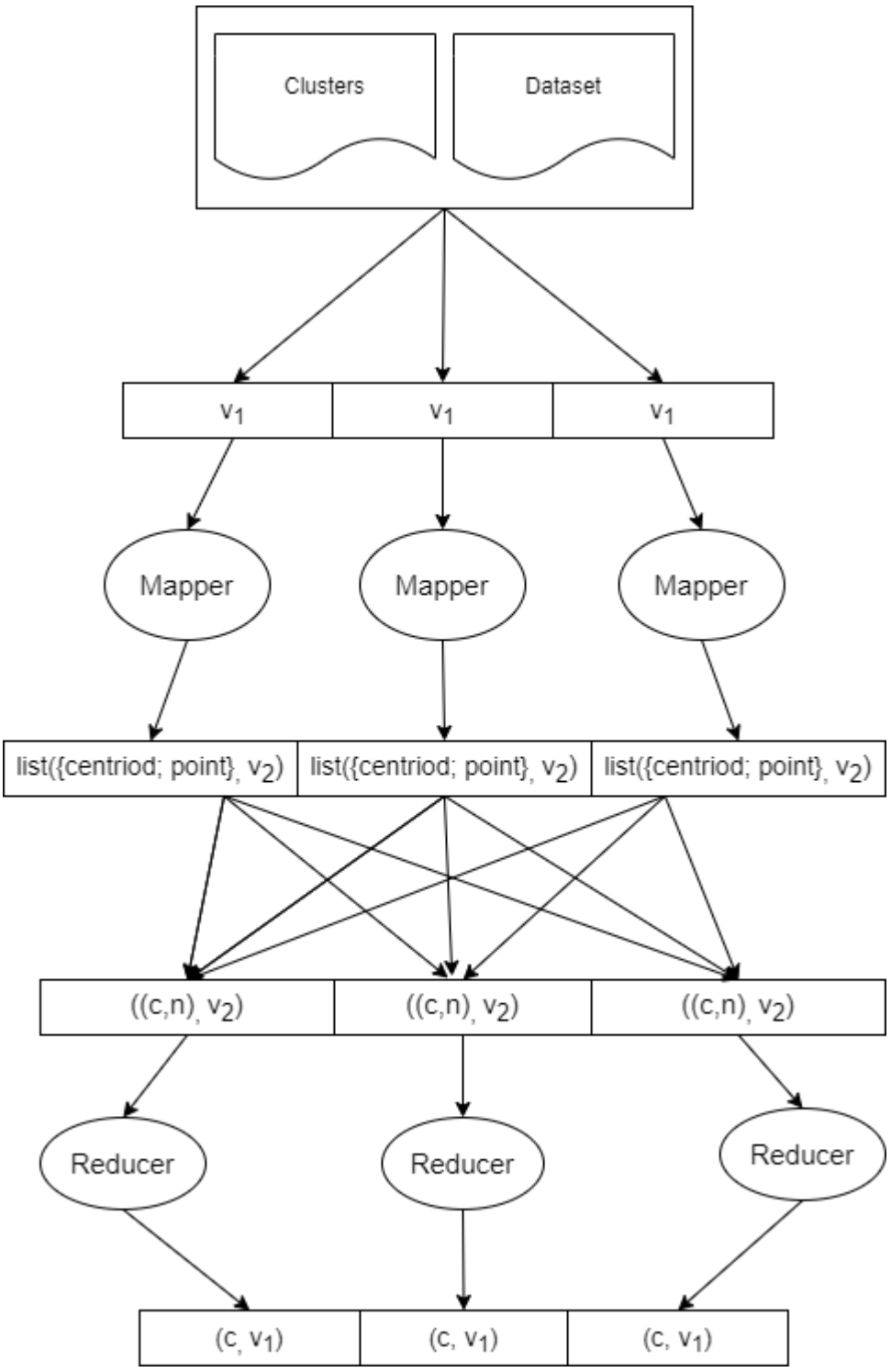

Рисунок 4 – Алгоритм выполнения одной итерации Knn

Один этап MapReduce примерно соответствует одной итерации классического алгоритма. Как и в классическом алгоритме, на первом этапе центроиды выбираются случайным образом из набора точек данных. Функция map принимает в качестве входных данных точку данных и список

центроидов, вычисляет расстояние между точкой и каждым центроидом и выдает точку и ближайший центроид. Функция уменьшения собирает все точки, относящиеся к кластеру, вычисляет новый центроид и излучает его. В конце каждого этапа он находит новое приближение центроидов, которые используются для следующей итерации. Рабочий процесс продолжается до тех пор, пока расстояние от каждого центроида предыдущей ступени и соответствующих центроидов текущей ступени не упадет ниже заданного порогового значения.

Операции map и reduce описываются следующим образом.

Map: для каждой точки с учетом всех K центроидов кластера вычисляется расстояние от точки до всех центроидов и выводит ближайший идентификатор кластера (выходной ключ) и точку (выходное значение).

Reduce: для каждого идентификатора кластера (входной ключ), учитывая все точки (входные значения), назначенные ему, обновляет центроиды кластера путем «усреднения» всех назначенных точек и выводит идентификатор кластера (выходной ключ) и обновленный центроид (выходное значение). Обновленные центроиды используются в качестве ввода кэшированных данных map для следущей итерации MapReduce.

Сначала центроиды и $dataset$ загружаются в распределенный кэш. Это делается путем переопределения функции настройки в классах Mapper и Reducer. После этого файл входных данных разделяется, и каждая точка данных обрабатывается одной из функций карты (в процессе карты). Функция записывает пары ключ-значение <Centroid, Point>, где Centroid является ближайшим к Point. Затем используется Combiner, чтобы уменьшить количество локальных записей. На этом этапе точки данных, которые находятся на одной машине, суммируются, и записывается количество этих точек данных, переменная Point.number. Теперь по причинам оптимизации выходные значения автоматически перемешиваются и сортируются по центроидам. Редуктор выполняет ту же процедуру, что и Объединитель, но также проверяет, сходятся ли центроиды; сравнение разницы между старыми

и новыми центроидами с входным параметром delta. Если центроид сходится, то глобальный счетчик остается неизменным, в противном случае он увеличивается.

2.2.3 Разработка алгоритма KNN

Модель данного алгоритма не будет сильно отличаться от двух предыдущих. Она также будет представлять данные в виде пар (ключ; значение).

На рисунке 5 представлен рабочий процесс вычислений MapReduce и KNN.

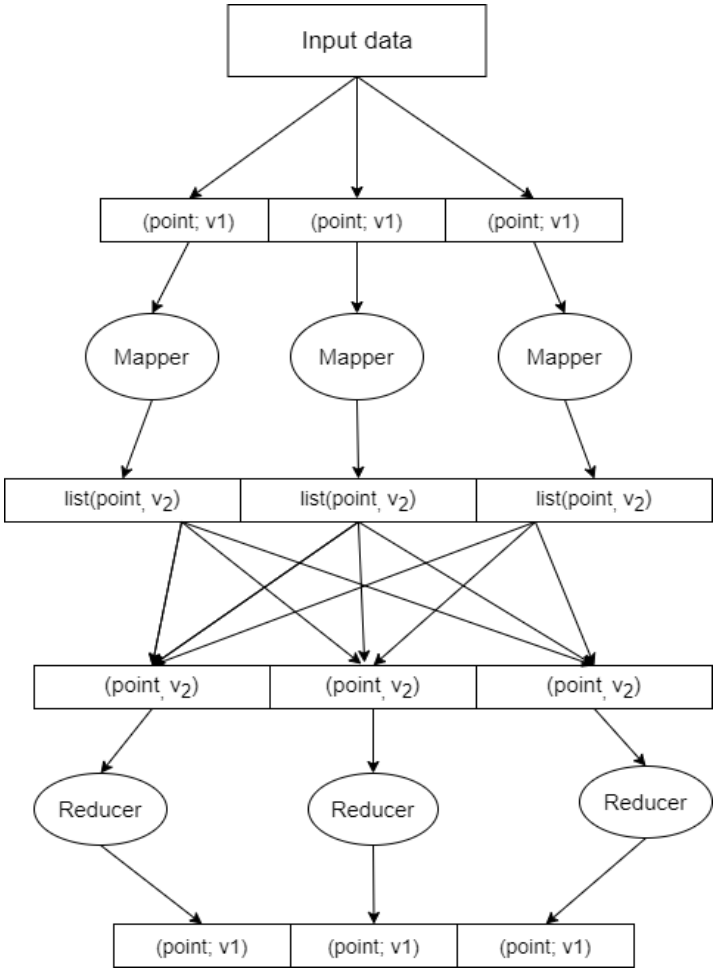

Рисунок 5 – Реализация алгоритма KNN с использованием MapReduce

Рабочий поток архитектуры вычислений MapReduce и KNN обрабатывается в три этапа.

Шаг 1: Первый шаг выполнил обработку данных; кроме того, он выполняет выбор проекции данных (доминирующих точек) от максимального до минимального измерения.

Шаг 2: Этот шаг используется для разделения и разделения обработанных данных. Этот шаг упрощает Процесс вычисления КНН.

Шаг 3: Третий шаг сделал одну или две функции MapReduce, в соответствии с расчетом расстояния и сортировкой.

Алгоритм KNN - очень полезный и несложный алгоритм. Здесь KNN использовал весь набор данных на этапе подготовки. В то время как прогнозирование требует неизвестных входных данных, в это время он ищет подготовленный набор данных для K связанных входных данных и данных с наиболее подходящими входными данными. На заключительном этапе в качестве прогноза рассматривается наиболее подходящий входной сигнал.

Выводы к главе 2

В данной главе описана модель программирования MapReduce, а также были разработаны алгоритмы кластеризации данных с использованием модели для оптимизации работы с большими данными.

Как показал анализ, алгоритм KNN - очень полезный и несложный алгоритм, доступный для реализации на основе MapReduce.

Глава 3 Реализация программы интеллектуального анализа данных

3.1 Описание используемых инструментов

Программная реализация всех алгоритмов была выполнена на языке Python в среде Google Colabaratory. Для реализации алгоритмов были выбраны следующие модули:

- OS – модуль, который предоставляет множество функций для работы с операционной системой, причём их поведение, как правило, не зависит от ОС, поэтому программы остаются переносимыми.
- Operator – модуль, который экспортирует набор эффективных функций, которые соответствуют внутренним операторам Python.
- Numpy – модуль содержащий большую библиотеку математических функций для работы с большими многомерными массивами и матрицами.
- Apyori – модуль простой реализации алгоритма Apriori.
- Matplotlib.pyplot – модуль по визуализации графиков. Представляет собой набор функций, которые заставляют matplotlib работать как MATLAB.
- Matplotlib.ticker – модуль содержащий несколько локаторов, использующие различные алгоритмы расположения меток на графике.
- Sklearn.model_selection.train_test_split – модуль, который разбивает массивы или матрицы на случайные обучающие и тестовые подмножества.
- Pandas.api.types – модуль, содержащий некоторые общедоступные функции, связанные с типами данных в Pandas.

Также был подключен фреймворк Hadoop для корректной работы mapper и reducer. Данный фреймворк разработан на языке Java, но его также можно использовать на других язык.

Для этого подключим фреймворк воспользуемся модулем os который предоставляет интерфейс для взаимодействия с файловой системой (рисунок 6).

```
[ ]  import os
     os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
     os.environ["SPARK_HOME"] = "/content/hadoop/bin/hadoop2.7
```

Рисунок 6 – Подключение фреймворка hadoop

3.2 Реализация алгоритма Apriori

Алгоритм работы классфикации можно разделить на 2 этапа. На 1 этапе идет обучение и классификация. Опишем работу классификатора на примере классификации вин. Для начала нужно подключить все необходимые библиотеки для работы (рисунок 7).

```
import os
import operator
import numpy as np
import pandas as pd
from apyori import apriori
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
from sklearn.model_selection import train_test_split
from pandas.api.types import is_numeric_dtype, is_bool_dtype
from sklearn.metrics import accuracy_score
```

Рисунок 7 – Подключение необходимых библиотек

Далее идет считывание файла с dataset (рисунок 8).

```
def file_contents(file_name):
  f = open(file_name)
  try:
    return f.read()
  finally:
      f.close()

def initData():
  items = file_contents('../itemlist10000.txt')
  data = items.split('\n')[0:10000]
  for i in range(len(data)):
    data[i] = data[i].split(' ')
  return data
```

Рисунок 8 – Считывание файла выборки

Выборка загружается из файла wine.csv, расположенном на облачном хранилище. Данная выборка содержит 5 числовых характеристик вина и 1 метку класса, а именно:

– щелочность золы;

– алкоголь;

– пепел;

– интенсивность цвета;

– флавоноиды;

– класс.

После выполнения блока кода создаётся дата-фрейм, где каждой колонке соответствует своя характеристика.

Для работы с числовыми и категориальными признаками опишем функции разбиения ряда чисел или критериев принадлежности на отрезки. Для первого случая, когда признак числовой, сначала все значения сортируются в порядке возрастания, по числу групп находится максимальная длина одной группы, и создаются массивы содержащие признаки, чьё общее число

определяется максимальной длиной. Количество групп для разбиения числовых параметров можно также регулировать через интерфейс (по умолчанию используется 5 групп). Функция numericByGroups возвращает двумерный список, содержащий информацию о сформированных группах в формате [<название группы>, (<граница отрезка a>, <граница отрезка b>)] (рисунок 9).

```
group_num = 5 #@param {type:"slider", min:2, max:15, step:1}

def numericByGroups(name, vals, groupNum = 5):

    # Сортировка ряда
    vals = sorted(vals)

    # Разбиение ряда на отрезки
    h = (vals[-1] - vals[0]) / groupNum
    ranges = [(vals[0] + h*i, vals[0] + h*(i+1)) for i in range(groupNum)]

    # Названия отрезков для поиска
    return [[name + '_' +
             '_'.join([('%d' if i.is_integer() else '%.2f') % i for i in r]),
             r] for r in ranges]
```

Рисунок 9 – Разбиение ряда чисел на отрезки

Далее в mapper происходит генерация локальных кандидатов (рисунок 10).

```
[ ]  def mapper(data):
       candidates = []
       if(k < 3):
         candidateItem = getItemFromItemset(data)
         candidates = list(itertools.combinations(candidateItem, k))
       else:
         keys = list(data.keys())
         for i in range(len(keys)):
           items = keys[i].split(' ')
           j = i + 1
           while(j < len(keys)):
             m = 0
             items2 = keys[j].split(' ')
             tag = True
             while(m < k - 2):
               if items[m] != items2[m]:
                 tag = False
                 break
               m += 1
             if tag:
               temp = []
               temp.extend(items)
               temp.append(items2[len(items2)-1])
               candidates.append(temp)
             j += 1
       tuppleToString(candidates)
       return candidates
```

Рисунок 10 – Формирование списка транзакций

После нахождения списка транзакций можно найти список правил. При этом, от пустых элементов и следствий следует избавляться. Формируемые правила хранятся в виде дата-фрейма. Каждое сформированное правило проходит модификацию и либо сохраняется, либо удаляется из списка правил. Для формирования списка правил используется алгоритм аффинитивного анализа Apriori, описанный в открытой библиотеки для Python Apyori (рисунок 11).

```
def genRules(transactions, elements, className):

    # Генерация списка правил с помощью apriori
    association_rules = list(apriori(transactions, min_support=0.0045,
                                min_confidence=0.2, min_lift=3, min_length=2))

    # Для хранения списка правил
    df_rules = pd.DataFrame(columns=['RuleL', 'RuleR', 'Support',
                                     'Confidence', 'SxC', 'Lift', 'Leverage'])
```

Рисунок 11 – Начало функции

После комбинирования Reducer и суммируют значения соответствующих ключей. Reducer выводят пары (ключ, значение), где ключ - это элемент, а значение транзакция (рисунок 12).

```
def Reducer(count):
  itemsets = list(count.keys())
  frequentItemsets = []
  for itemset in itemsets:
    if count[itemset] < threshold:
      del count[itemset]
    else:
      frequentItemsets.append(itemset)
  return frequentItemsets
```

Рисунок 12 – Алгоритм работы Reducer

Сформированная транзакция представляет собой список строк с

названиями групп, содержащихся в данной транзакции. Каждая транзакция сохраняется в массив транзакций.

На выходе функции формируется массив транзакций и словарь (dict) элементов транзакций для последующего перевода записи в транзакцию при классификации. Массив транзакций хранит строки с названиями предметов, словарь элементов хранит информацию о каждом элементе в формате (рисунок 13):

```
param_name: {
  type: 'categorical' or 'numeric',
  groups: [
      ['param_name_a1_b1', (a1, b1)],
      ['param_name_a2_b2', (a2, b2)],
      ...
  ]
}
```

Рисунок 13 – Формат хранения данных об элементе

Здесь param_name соответствует столбцу обучающей выборки. В поле type хранится тип параметра, в поле groups - сформированные группы для каждого параметра (рисунок 14).

```
def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):
        _itemSet = set()
    localSet = defaultdict(int)

    for item in itemSet:
        for transaction in transactionList:
            if item.issubset(transaction):
                freqSet[item] += 1
                localSet[item] += 1

    for item, count in localSet.items():
        support = float(count) / len(transactionList)

        if support >= minSupport:
            _itemSet.add(item)

    return _itemSet
```

Рисунок 14 – Вычисление минимальной поддержки

Далее вычисляется поддержка элементов в наборе и возвращает

подмножество набора элементов, каждый из элементов которого удовлетворяет минимальной поддержке.

3.3 Реализация алгоритма K-means

Первым делом нужно смонтировать Google Drive, потом указать путь к директории с исходным файлом, находящегося в Google Drive и записать в переменную ROOT_DIR (рисунок 15).

```
[ ]  import pandas as pd
     import numpy as np
     import sys

     from google.colab import drive
     drive.mount('/content/drive/', force_remount=True)
     ROOT_DIR = '/content/drive/My Drive/Colab Notebooks'
     sys.path.append(ROOT_DIR)
```

```
Mounted at /content/drive/
```

Рисунок 15 – Импорт необходимых библиотек, а также прописываем путь к исходным файлам

Считываем данные в переменную file, разделяем на две части при помощи операции среза: x - датафрейм с независимыми переменными, y – датафрейм с меткой класса, в файле должен быть последним столбцом (рисунок 16).

```
dataFile = os.path.join(ROOT_DIR, dataPath)
file = pd.read_csv(dataFile)
x = file[file.columns[0:-1]]
y = file[file.columns[-1]]
```

Рисунок 16 – Считывание данных из файла и разделение на две части

Нормируем все атрибуты к диапазону от 0 до 1 при помощи MinMaxScaler из библиотеки sklearn (рисунок 17).

```
[ ]  from sklearn import preprocessing as pre
     scaler = pre.MinMaxScaler(feature_range=(0, 1))
     x = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
```

Рисунок 17– Нормирование атрибутов

Разделяем исходную выборку при помощи train_test_split из пакета sklearn.model_selection. Параметр test_size показывает, сколько процентов исходной выборки уйдёт на тестовую

Теперь производим кластеризацию на k кластеров при помощи готового алгоритма из библиотеки sklearn.cluster (рисунок18).

```
[ ]  from sklearn.cluster import KMeans
     km = KMeans(n_clusters=k)
     km.fit(x_train)
```

Рисунок 18 – Кластеризация

При вызове команды km.cluster_centers_ можно увидеть, какие значения атрибутов содержатся в центрах кластеров (рисунок 19).

```
[ ]  km.cluster_centers_

     array([[0.73611111, 0.47172619, 0.8220339 , 0.86904762],
            [0.25854701, 0.69391026, 0.0821382 , 0.07692308],
            [0.35119048, 0.23363095, 0.5090799 , 0.4702381 ],
            [0.11640212, 0.45634921, 0.07263923, 0.03769841],
            [0.54558405, 0.38141026, 0.65623642, 0.63995726]])
```

Рисунок 19 – - Просмотр атрибутов центров кластеров

Теперь сформируем датафрейм, содержащий информацию о том, какие классы входят в какой кластер (рисунок 20).

```
[ ]  data_array = np.array([km.labels_, y_train.values])
     data = pd.DataFrame(data_array.T, columns=['Cluster', 'Class'])
     data.groupby(['Cluster', 'Class']).aggregate({'Class':'count'})
```

Рисунок 20 – Формировка датафрейма с информацией о кластерах

Далее Mapper вычисляет расстояние между точкой данных и каждым центроидом. Затем выдает индекс ближайшего центроида и точки данных.

```
class MAPPER
  method MAP(file_offset, point)
      min_distance = POSITIVE_INFINITY
      closest_centroid = -1
      for all centroid in list_of_centroids
          distance = distance(centroid, point)
          if (distance < min_distance)
              closest_centroid = index_of(centroid)
              min_distance = distance
      EMIT(closest_centroid, point)
```

Рисунок 21 – Реализация Mapper

Далее описаны функции для вычисления метрик (рисунок 22).

```
[ ]  import math

     def EuclideanMetric(node1, node2):
       return math.sqrt(sum([(node1[i] - node2[i]) ** 2 for i in range(node1.shape[0])]))

     def ManhattanMetric(node1, node2):
       return sum([abs(node1[i] - node2[i]) for i in range(node1.shape[0])])

     def ChebyshevMetric(node1, node2):
       return max([abs(node1[i] - node2[i]) for i in range(node1.shape[0])])
```

Рисунок 22 – Функции для вычисления метрик

Затем описана функция reducer для классификации датафрейма с неизвестными объектами, листинг и результат которой показан на рисунке 23.

На вход получает:

– датафрейм с номером кластера, меткой класса и координатами центра
– нормализованный датафрейм, содержащий объекты для классификации
– функция метрики, выбирается на предыдущем этапе

Возвращает массив с метками классов.

На вход получает:

– датафрейм с номером кластера, меткой класса и координатами центра
– нормализованный датафрейм, содержащий объекты для классификации
– функция метрики, выбирается на предыдущем этапе

Возвращает массив с метками классов.

```
[ ]  def predict(trained_clusters, x_undef, metric):
       y_result = np.array([])
       for i in range(x_undef.shape[0]):
         obj = x_undef.iloc[i]
         cl = trained_clusters.iloc[0]["Class"]
         min_dist = metric(obj, trained_clusters.iloc[0]["Center"])
         for j in range(1, trained_clusters.shape[0]):
           cur_dist = metric(obj, trained_clusters.iloc[j]["Center"])
           if cur_dist < min_dist:
             min_dist = cur_dist
             cl = trained_clusters.iloc[j]["Class"]
         y_result = np.append(y_result, cl)
       return y_result

     predict(trained, x_test, EuclideanMetric)
```

```
array(['Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa'], dtype='<U32')
```

Рисунок 22 – Результат работы функции result

3.4 Реализация алгоритма KNN

Для реализации так же подключаем библиотеки (рисунок 24).

```
import random
import math
import pylab as pl
import numpy as np
from matplotlib.colors import ListedColormap
```

Рисунок 23 – Подключение библиотек

Далее считываем выборку из файла (рисунок 25).

```
from google.colab import drive
drive.mount('/content/drive/', force_remount=True)
ROOT_DIR = '/content/drive/My Drive/Colab Notebooks'
sys.path.append(ROOT_DIR)
```

```
Mounted at /content/drive/
```

Рисунок 24 – Считывание dataset

Далее разделим нам набор данных на обучающую и тестовую выборки (рисунок 26).

```
def splitTrainTest (data, testPercent):
trainData = []
testData  = []
for row in data:
    if random.random() < testPercent:
        testData.append(row)
    else:
        trainData.append(row)
return trainData, testData
```

Рисунок 25 – Разбиение на тестовую и обучающую выборки

Далее Mapper рассчитывает Евклидово расстояние до всех точек.

```
def Mapper (trainData, testData, k, numberOfClasses):
    #Евклидово расстояние между 2-мерной точкой
    def dist (a, b):
        return math.sqrt((a[0] - b[0])**2 + (a[1] - b[1])**2)
    testLabels = []
    for testPoint in testData:
        #Рассчет расстояния между тестовыми и всеми данными
        testDist = [ [dist(testPoint, trainData[i][0]), trainData[i][1]] for i in range(len(trainData))]
        #Подсчет сколько точек каждого класса среди ближайших K
        stat = [0 for i in range(numberOfClasses)]
        for d in sorted(testDist)[0:k]:
            stat[d[1]] += 1
        #Назначается класс с наибольшим количеством вхождений среди K ближайших соседей
        testLabels.append( sorted(zip(stat, range(numberOfClasses)), reverse=True)[0][1] )
    return testLabels
```

Рисунок 27 – Алгоритм работы mapper

Теперь можно оценить, насколько хорошо работает наш классификатор (рисунок 28).

```
Confusion Matrix:
[[21 0 0]
[ 0 16 0]
[ 0 7 16]]
Classification Report:
                 precision   recall   f1-score   support
    Iris-setosa        1.00     1.00       1.00        21
Iris-versicolor        0.70     1.00       0.82        16
 Iris-virginica        1.00     0.70       0.82        23
      micro avg        0.88     0.88       0.88        60
      macro avg        0.90     0.90       0.88        60
   weighted avg        0.92     0.88       0.88        60

Accuracy: 0.8833333333333333
```

Рисунок 26 – пример работы классификатора

Для этого сгенерируем данные, разобьем их на обучающую и тестовую выборку, произведем классификацию объектов тестовой выборки и сравним реальное значение класса с полученным в результате классификации

3.5 Сравнение результатов

Для сравнения возьмем выборки размером 10000, 50000 и 100000 элементов. Для каждого dataset тестирование проводилось 10 раз, так как для алгоритма k–means очень важно начальное расположение центроидов, так что время выполнения алгоритмов – это среднее время выполнение всех итераций.

Для начала рассмотрим сравнение алгоритм apriori с использованием модели MapReduce и оригинальный алгоритм (рисунок 29).

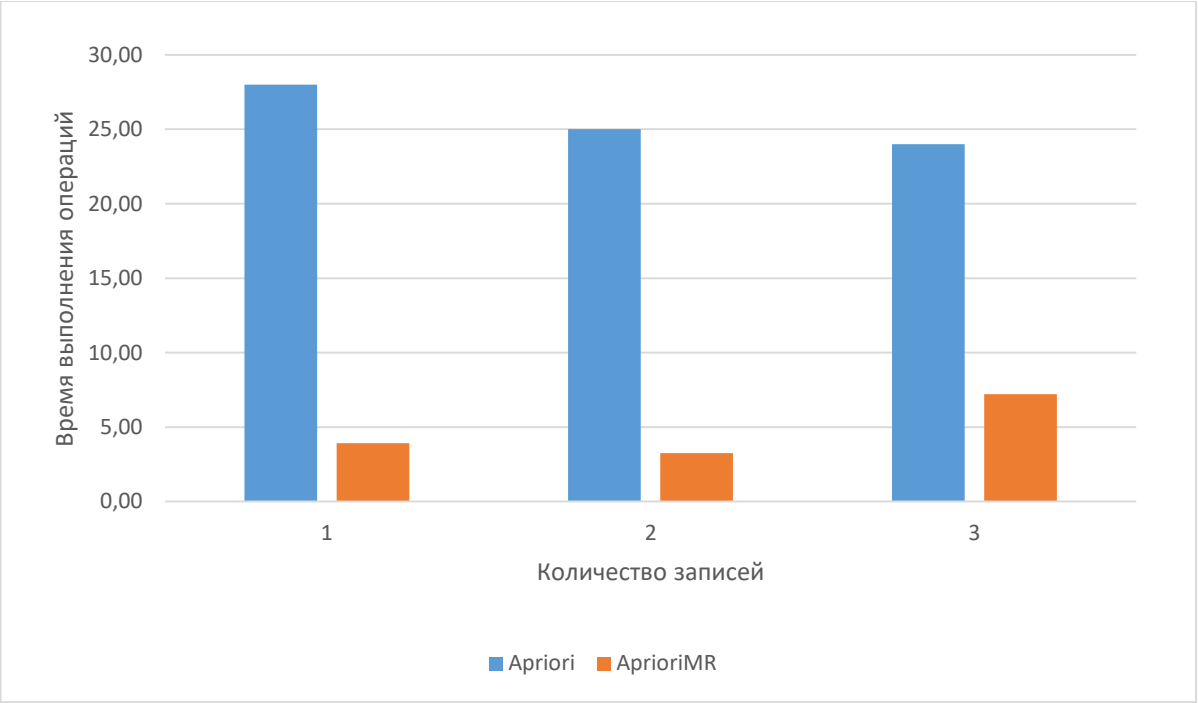

Рисунок 29 – Диаграмма сравнения алгоритма Apriori

Как мы видим из диаграммы алгоритм с использованием MapReduce алгоритм работает в разы быстрее, чем оригинальный. Ниже в таблицы приведены результаты тестирования (таблица 21).

Таблица 1 – Сравнение времени, затраченное на обработку dataset

Количество записей	Apriori	AprioriMR
10000	28,12 с	3,93 с
50000	25,43 с	3,25 с
100000	24,37 с	7,21 с

Теперь рассмотрим алгоритм k–means.

На рисунке 30 представлена диаграмма сравнения алгоритмов k-means MapReduce с оригинальным алгоритмом.

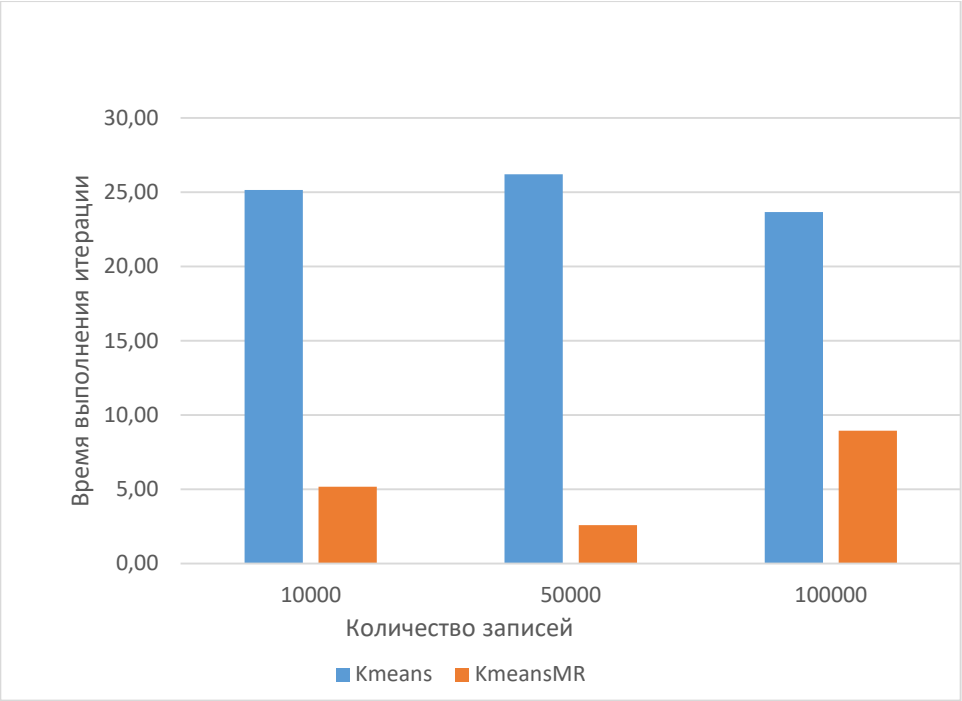

Рисунок 30 – Диаграмма сравнения алгоритмов k-means MapReduce с оригинальным алгоритмом

В данном случае алгоритм показал лучшее время обработки при dataset в 50000 записей, но сильно отстал при обработке 100000 записей (таблица 2).

Таблица 2 – Время итерации алгоритмов k-means и k–means MapReduce.

Количество записей	Kmeans	KmeansMR
10000	25,14	5,17
50000	26,19	2,58
100000	23,66	8,94

С алгоритмом KNN дела обстоят по-другому.

На рисунке 31 представлена диаграмма сравнение алгоритмов KNN и KNN MapReduce.

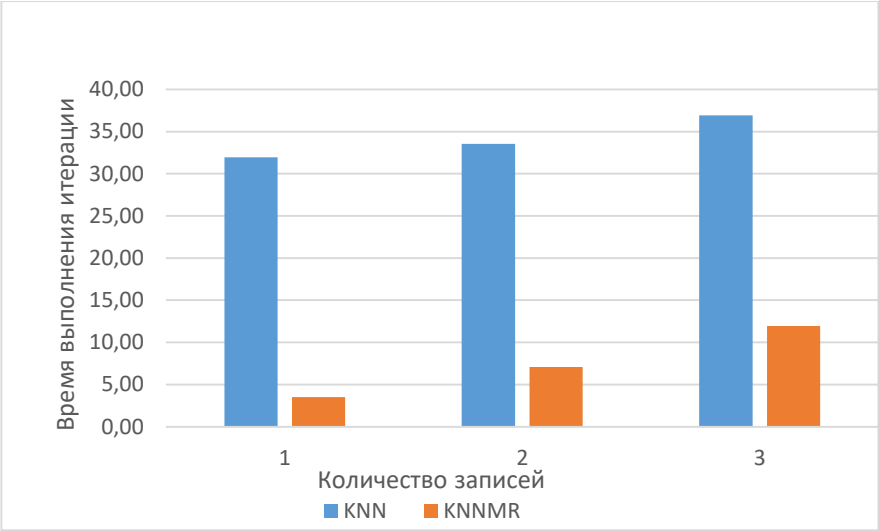

Рисунок 31 – Сравнение алгоритмов KNN и KNN MapReduce

Как видно из диаграммы KNN с использованием MapReduce все еще намного быстрее оригинального алгоритма, но с увеличением выборки скорость вычисления падает. Связано это с тем, что knn не строит никакой модели обучения, а просто перебирает все возможные варианты, из-за чего количество итераций сильно увеличивается.

Теперь сравним алгоритмы с использованием MapReduce между собой (рисунок 32).

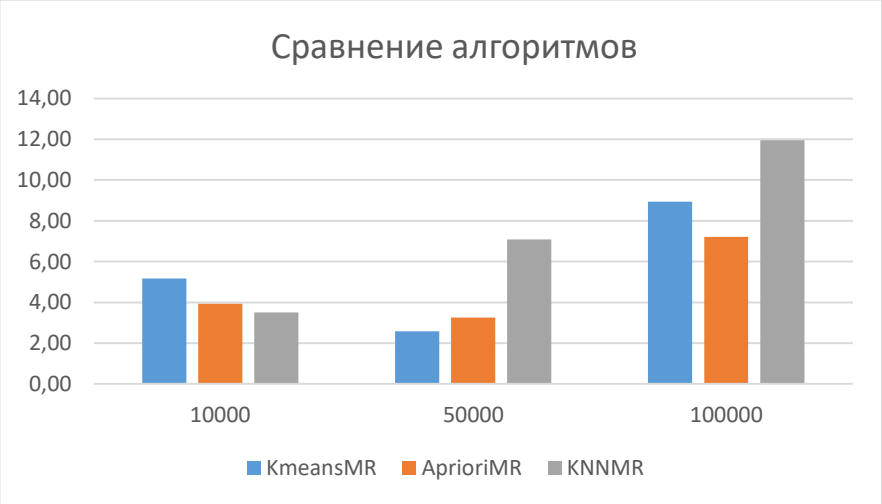

Рисунок 32 – Сравнение алгоритмов с использованием MapReduce

На диаграмме можно выделить, что наилучшим алгоритмом является Apriori. Несмотря на то, что алгоритм не отработал лучше других во всех 3 выборках, он обладает хорошей стабильностью.

K–means хоть и показал достаточно хороший результат в тестировании с 50000 записями, сильно зависит от начальных расположений центроидов, что в итоге и сказалось на результатах.

Выводы к главе 3

В данной главе была представлена реализация 3 методов с использованием модели программирования MapReduce, а также было проведено сравнение эффективности данных алгоритмов с оригинальными версиями, а также между собой.

Как показал анализ, наилучшим алгоритмом является Apriori.

K–means сильно зависит от начальных расположений центроидов, что в итоге и сказалось на результатах.

KNN намного быстрее оригинального алгоритма, но с увеличением выборки скорость вычисления падает.

Заключение

Выпускная квалификационная работа посвящена актуальной проблеме применение алгоритмов Data Mining для работы с большими объемами данных.

Для достижения данной цели в процессе работы над бакалаврской работой решены следующие задачи:

- описан математический аппарат алгоритмов Apriori, k-means и KNN;
- проведен анализ и сравнение указанных алгоритмов Data mining для выбора наиболее оптимальных алгоритмов для работы с большими данными. Как показал анализ, алгоритм Apriori довольно прост и понятен, а также легок в реализации в отличии от его. Также данный алгоритм имеет множество вариаций и модификаций, что подчеркивает его актуальность в настоящее время;
- описана модель программирования MapReduce, а также были разработаны алгоритмы кластеризации данных с использованием модели для оптимизации работы с большими данными.
- выполнена программная реализация алгоритмов и выполнена оценка их эффективности. Как показал анализ, наилучшим алгоритмом является Apriori. K–means сильно зависит от начальных расположений центроидов, что в итоге и сказалось на результатах. KNN намного быстрее оригинального алгоритма, но с увеличением выборки скорость вычисления падает.

Результаты бакалаврской работы представляют научно-практический интерес и могут быть рекомендованы для разработчиков программ анализа больших массивов данных на основе методов Data mining.

Список используемой литературы

1. Айвазян С. А., Бухштабер В.М., Енюков И.С. Прикладная статистика: классификация и снижение размерности // Финансы и статистика. 2007. № 1. С. 50 -57.
2. Барсегян А.А, Куприянов М.С., Степаненко В.В., Холод И.И. Методы и модели анализ данных: OLAP и Data Mining. Санкт-Петербург: БХВ - Петербург, 2004. 360 с.
3. Басалаева Ю. С. Исследование алгоритмов кластеризации с целью анализа данных проверяющей системы // Молодые исследователи - регионам: материалы межд. научн. конф., 20-25 апр. 2015г. Вологда, 2015. С. 50 - 51.
4. Басалаева Ю.С. Выбор инструментов Data Mining для анализа результатов дистанционного образования // Современные материалы, техника и технология. 2015. № 2. С. 22 - 25.
5. Басалаева Ю.С. Проблема очистки данных дистанционного практикума по программированию в процессе кластерного анализа // Современные тенденции развития науки и производства. 2016. № 2. С. 119 - 120.
6. Большакова Е.И., Клышинский Э.С., Ландэ Д.В. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика: учеб. пособие. Москва: МИЭМ, 2011. 272 с.
7. Гладкий М.В. Модель распределенных вычислений MapReduce// Труды БГТУ. 2016. № 6. С. 194–198.
8. Информационно-аналитический ресурс, посвященный машинному обучению, распознаванию образов и интеллектуальному анализу данных [Электронный ресурс]. URL: www.machinelearning.ru (дата обращения 10.05.2021).
9. Котов А. Кластеризация данных // Инновации. 2015. №1. С.34-37.
10. НОУ ИНТУИТ: Введение в машинное обучение [Электронный ресурс]. URL: https://intuit.ru/studies/courses/10621/1105/lecture/17981 (дата обращения 10.05.2021).

11. НОУ ИНТУИТ: Задачи Data Mining. Информация и знания [Электронный ресурс]. URL: https://intuit.ru/studies/courses/6/6/lecture/164 (дата обращения 10.05.2021).

12. Паклин Н. Б. Бизнес-аналитика: от данных к знаниям / Н. Б. Паклин, В. И. Орешков. - Санкт-Петербург: Изд. Питер, 2009. - 624 с.

13. Ржеуцкая С.Ю., Басалаева Ю.С. Опыт применения методов кластеризации для анализа результатов дистанционного обучения // Информатизация инженерного образования: материалы международной науч.-практ. конф., 12-13 апр. 2016 г. Москва: МЭИ, 2016. С. 617 - 620.

14. Berkhin P. Survey of Clustering Data Mining Techniques / P. Berkhin. - USA: Accrue Software, 2002. 55 p.

15. CVAP: Cluster Validity Analysis Platform (cluster analysis and validation tool) [Электронный ресурс]. URL: http://www.mathworks.com/matlabcentral/fileexchange/14620-cvap--cluster-validity-analysis-platform--cluster-analysis-and-validation-tool/ (дата обращения 10.05.2021).

16. Data Mining: Practical Machine Learning Tools and Techniques (The Morgan Kaufmann Series in Data Management Systems), 2010.

17. Fisher D.H. Knowledge acquisition via incremental conceptual clustering. German: University of Tubingen, 2008. 211 p.

18. Hastie T. et. al. The Elements of Statistical Learning. German: Springer, 2011. 70 p.

19. Ian H. Data Mining: Practical Machine Learning Tools and Techniques / H. Ian, E. Frank, A. Hall. - San Francisco: Morgan Kaufmann, 2011. P. 664 - 666.

20. McCallum A. Efficient Clustering of High Dimensional Data Sets with Application to Reference Matching / A. McCallum, K. Nigam // ACM Sigkdd. 2012. P. 169 - 178.