Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Смирнов Александр Андреевич

Реализация функционала для поиска ассоциативных правил в рамках платформы Desbordante

Отчет по учебной практике

Научный руководитель:
ассистент кафедры ИАС Чернышев Г. А.

Санкт-Петербург
2022

Оглавление

Введение

Профилирование данных — это совокупность методов, направленная на получение разнообразных метаданных (данных о данных) из наборов данных [1]. Профилирование широко применяется как аналитиками, так и учеными при исследовании наборов данных. Существует множество различных типов метаданных и методов их извлечения при анализе датасета. Один из простейших методов профилирования — подсчёт количества значений NULL в какой-то колонке таблицы. Есть и более сложные методы — например, поиск уникальных столбцов, зависимостей включения, функциональных зависимостей, ассоциативных правил и множество других.

Для каждого из таких методов разработаны различные алгоритмы, которые отличаются производительностью при различных входных данных — влияет, например, количество столбцов, колонок, структура исследуемых данных.

Существует множество готовых инструментов, которые позволяют осуществлять профилирование. Как правило, такие инструменты поддерживают много различных методов. Однако они довольно узконаправлены, потому что орентированы на решение каких-то бизнес задач, и в редких случаях предоставляют выбор того или иного алгоритма для конкретного метода. Таким образом, становится непросто протестировать различные алгоритмы на обрабатываемых данных. Кроме того, зачастую эти инструменты требуют установки на машину пользователя, что также влечет дополнительные сложности и затраты времени. Существуют также и различные open-source библиотеки, которые содержат реализацию алгоритмов. Однако и здесь могут возникнуть затруднения — библиотеку необходимо установить, изучить её API, что не всегда удобно, особенно если нужно проанализировать сразу несколько разных методов или алгоритмов из различных библиотек.

Таким образом, было принято решение создать Desbordante — инструмент для профилирования данных с открытым исходным кодом. Его задача состоит в том, чтобы собрать различные методы профи-

лирования и алгоритмы в одном инструменте. Он представляет собой сочетание консольного приложения и frontend-клиента, которые позволяют запускать различные алгоритмы профилирования данных без установки каких-либо программ с использованием web-интерфейса (система доступна[1] online).

Одним из популярных методов профилирования является поиск ассоциативных правил. Этот метод применяется к транзакционным наборам данных — наборам, похожим на базу данных чеков из супермаркета. Ищутся правила вида “если в чеке есть продукт A, то с какой-то вероятностью в нём будет и продукт B”. Полученные таким способом метаданные применяются [2] в различных областях: анализе покупок, логов веб-сервисов, биоинформатике. Также такие правила могут использоваться при классификации и кластеризации. Поэтому было принято решение добавить функциональность по поиску ассоциатиных правил в Desbordante. Как и для других методов профилирования, разработано [2] множество различных алгоритмов поиска таких правил, и одним из самых первых был представлен алгоритм Apriori. Однако, несмотря на свой возраст, он и сейчас сохраняет свою актуальность — поэтому и был реализован автором данной работы.

[1] `https://desbordante.unidata-platform.ru/`

Постановка задачи

Целью работы является реализация функционала для поиска ассоциативных правил в рамках Desbordante.

Для достижения цели были поставлены следующие задачи:

- произвести краткий обзор предметной области и алгоритма поиска ассоциативных правил Apriori;
- разработать и реализовать архитектуру для поиска ассоциативных правил в рамках существующей кодовой базы Desbordante;
- реализовать алгоритм поиска ассоциативных правил Apriori;
- подготовить юнит-тесты и провести первичное экспериментальное исследование производительности.

1. Обзор

1.1. Основные понятия

Введём основные понятия, которые будут широко использоваться в последующем изложении. Все дальнейшие определения приводятся по [2, 3].

Definition 1 *Пусть* $I = \{i_1, i_2, ..., i_n\}$ *— множество некоторых сущностей, называемых предметами. Транзакционым набором называют множество* $D = \{T_1, T_2, ..., T_m\}$*, где* T_j *— множество предметов, такое, что* $T_j \subseteq I$*.*

Definition 2 *Множество* $T_j \in D$ *называется транзакцией, а число* j *— её уникальным идентификатором.*

Изначально проблема поиска ассоциативных правил была представлена в терминах “корзины с продуктами”, поэтому некоторые понятия удобно представлять именно таком ключе. Так, например, транзакцию T можно представить как содержимое корзины или чек из супермаркета — то есть как множество продуктов, купленных покупателем за одно посещение магазина. Стоит отметить, что в данном случае нас не интересует количество или стоимость купленных продуктов — только лишь их наличие в корзине. Транзакционный набор D в данном случае представляет собой базу данных магазина, в которой хранятся чеки. Множество I — это множество всех продуктов, которые продаются в магазине.

Definition 3 *Пусть* $X \subseteq I$ *— какой-то набор предметов. Говорят, что транзакция* T_j *содержит* X*, если* $X \subseteq T_j$

Definition 4 *Пусть* $X \subseteq I$ *— какой-то набор предметов.* ***Поддержкой*** *или* ***support*** *набора* X *называют отношение количества транзакций* $T_j \in D$*, которые содержат* X*, к мощности множества* D

$$sup(X) = \frac{\#\{T_j \in D \mid X \subseteq T_j\}}{|D|}$$

Definition 5 *Пусть $X, Y \subseteq I$ — наборы предметов. Тогда для них справедливо свойство, которое называют свойством монотонности поддержки:*

$$sup(X) \geq sup(Y) \quad \forall X \supseteq Y$$

Definition 6 *Пусть $X \subseteq I$ — набор предметов. X называется* ***частым набором*** *или* ***frequent itemset****, если его поддержка не меньше некоторой заданной константы minsup: $sup(X) \geq minsup$.*

Definition 7 *Пусть $X, Y \subseteq I$ — наборы предметов.* ***Доверием*** *или* ***confidence*** *правила $X \Rightarrow Y$ называют условную вероятность того, что набор Y входит в случайно выбранную транзакцию при условии, что набор X также входит в эту транзакцию*

$$conf(X \Rightarrow Y) = \frac{sup(X \cup Y)}{sup(X)}$$

Definition 8 *Пусть $X, Y \subseteq I$ — наборы предметов. Правило $X \Rightarrow Y$ называют* ***ассоциативным правилом*** *с минимальной поддержкой minsup и минимальным доверием minconf, если оно удовлетворяет двум критериям:*

1. $sup(X \cup Y) \geq minsup$;
2. $conf(X \Rightarrow Y) \geq minconf$.

Таким образом, процесс поиска ассоциативных правил можно разделить [2] на две части:

- сгенерировать множество частых наборов $F = \{X \subseteq I \mid sup(X) \geq minsup\}$;
- с помощью найденых на предыдущем шаге наборов сгенерировать ассоциативные правила $A \Rightarrow B$, такие, что $A \cup B \in F$ и $conf(A \Rightarrow B) \geq minconf$.

Второй шаг довольно простой, и не требует различных ресурсоемких алгоритмов для работы. Однако, первый шаг — генерация частых наборов — весьма требователен к вычислениям, во многом из-за необходимости непосредственной работы с базой данных.

1.2. Алгоритм Apriori

При генерации множества F нужно проверить все подмножества предметов $X \subseteq I$. Множество всех подмножеств удобно представлять в виде алгебраической решётки: на одном её уровне располагаются наборы одинаковой мощности. Однако, количество подмножеств, которые нужно рассмотреть, экспоненциально растёт с увеличением количества предметов $|I|$, и простой перебор будет малоэффективен уже при $|I| = 1000$.

Apriori [2, 3] — один из первых алгоритмов, которые особым образом строят алгебраическую решётку, тем самым избегая полного её перебора. Алгоритм последовательно генерирует решётку по уровням, начиная с уровня наборов мощности один. При этом используется свойство монотонности поддержки: если на уровне n некоторый сгенерированный набор A не является частым, то есть $sup(A) < minsup$, то **любое** надмножество этого набора не имеет смысла рассматривать — по свойству монотонности ни одно из них тоже не будет частым. При таком подходе отсекаются значительные части решётки, которые заведомо не имеет смысла рассматривать. По завершению работы алгоритм возвращает искомое множество F — множество частых наборов, завершив первый шаг поиска. Затем некоторый общий алгоритм генерирует ассоциативные правила на основе этого множества, что соответствует второму шагу поиска.

Наиболее ресурсоемкая стадия работы алгоритма — вычисление поддержки для рассматриваемых подмножеств $X \subseteq I$. Для каждого такого X, которое не было отброшено, поддержка считается по определению: необходимо перебрать все транзакции $T_j \in D$ и посчитать количество тех, в которых содержится X. При таком подходе каждый раз происходит проход по всей базе данных, что может негативно сказаться на производительности. Для оптимизации этого процесса используется хеш-дерево — особая структура данных, представленная в оригинальной статье [3]. Она позволяет проходить базу данных не для каждого рассматриваемого подмножества X, а только лишь для каждого уров-

ня решётки. Таким образом, производительность увеличивается за счёт меньшего количества проходов и, следовательно, меньшего количества обращений к памяти.

Через некоторое время после публикации Apriori был представлен Enumeration-Tree framework — более общий фреймворк для эффективного обхода решётки. Согласно этому подходу, алгебраическая решетка подмножеств строится последовательно в виде дерева. Корнем дерева является узел, представляющий пустое множество. Дерево может расти различными способами — в глубину, в ширину, или некоторым смешанным образом, причем разные варианты по-разному влияют на производительность. На базе этого фреймворка построено [2] множество более современных алгоритмов, например, DepthProject, FP-Tree. Apriori тоже удобно интерпретировать в рамках данного фреймворка — дерево строится строго уровень за уровнем, то есть “в ширину”.

1.3. Существующие решения

Рассмотрим существующие инструменты для анализа данных.

1.3.1. Metanome

Платформа Metanome[2] — инструмент для получения метаданных (профилирования) наборов данных с открытым исходным кодом, написанный на язке Java. Он включает в себя множество алгоритмов профилирования, таких как алгоритмы поиска уникальных наборов столбцов, зависимостей включения, функциональных зависимостей. Задача Metanome — обеспечить удобную для пользователя работу с этими алгоритмами, объединив их в один инструмент [5]. Это позволяет быстро получить информацию из нужного набора данных — достаточно загрузить его в платформу, и становится возможным запускать на нём различные алгоритмы с помощью единого элемента управления. Чтобы сделать этот процесс более удобным, Metanome оснащён frontend-

[2] https://hpi.de/naumann/projects/data-profiling-and-analytics/metanome-data-profiling.html

клиентом, который позволяет совершать эти операции в графическом интерфейсе. Однако при всех своих достоинствах Metanome не поддерживает работу с ассоциативными правилами, что в рамках рассматриваемой задачи является недостатком. Также система недоступна online, и для того, чтобы ей воспользоваться, развертывание необходимо произвести вручную на своём оборудовании.

1.3.2. Desbordante

Платформа Desbordante[3] — новый инструмент для профилирования данных с открытым исходным кодом, реализованный на языке C++. Его назначение отчасти совпадает с назначением Metanome: оба фреймворка объединяют существующие методы профилирования и алгоритмы в один инструмент. Также инструмент оснащён frontend-клиентом, который обеспечивает удобную работу с исследуемым набором данных, в том числе и визуализацию полученных результатов. Система доступна online, и для работы с ней необязательно запускать её на компьютере пользователя. На данный момент Desbordante поддерживает лишь небольшое количество различных методов профилирования, однако находится в стадии активной разработки.

Существенное отличие от Meatnome состоит в том, последний может не подойти для промышленного использования, поскольку производительности Java недостаточно [6] для работы с большими наборами данных. По этой причине Desbordante был реализован с целью достижения максимально возможной производительности алгоритмов, чего планировалось достичь за счёт использования языка C++.

1.3.3. SAP Predictive Analytics

SAP Predictive Analytics [10] — профессиональный инструмент для анализа данных, ориентированный на предсказательную аналитику. Поддерживает различные методы работы с данными, такие как классификация, регрессия, кластеризация, ассоциативные правила. В частно-

[3] `https://github.com/Mstrutov/Desbordante`

сти, позволяет удобно работать с ассоциативными правилами: выводить список найденных правил, сортировать их по различным параметрам, графически представлять результат. По результатам анализа некоторого датасета можно создать предсказывающую модель, которая на основе найденных правил сможет прогнозировать их в аналогичном датасете. Однако инструмент не предоставляет возможности выбора конкретного алгоритма поиска ассоциативных правил. Стоит отметить, что это коммерческий инструмент, его исходный код закрыт, а таже требуется установка на компьютер пользователя.

1.3.4. SAS Enterprise Miner

SAS Enterprise Miner [11] — Ещё один профессиональный инструмент для анализа данных. Как и SAP Predictive Analytics, поддерживает удобную работу с ассоциативными правилами — вывод списка правил, фильтрацию, сортировку по различным параметрам. Однако предоставляет более разнообразные способы графического представления правил. Этот инструмент тоже не предоставляет возможности выбора конкретного алгоритма, только лишь удобный интерфейс для поиска правил и работы с ними. Как и предыдуший инструмент, является коммерческим продуктом, исходный код закрыт, требует установки.

1.3.5. Mlxtend

mlxtend [9] — open-source библиотека для анализа данных и машинного обучения, реализованная на Python. Поддерживает различные алгоритмы поиска ассоциативных правил, такие как Apriori, FP-Growth. Отсутствует графический интерфейс, все манипуляции необходимо производить при помощи вызова методов библиотеки.

1.3.6. Arules

arules [4] — open-source пакет для языка R. Предоставляет инфраструктуру для работы и поиска ассоциативных правил, в том числе эффективные реализации алгоритмов Apriori и Eclat на языке C. Как

и предыдущая библиотека, не имеет собственного графического интерфейса и рассчитана на работу через вызовы методов.

1.3.7. Вывод

Наглядное сравнение различных решений представлено в таблице 1 по следующим критериям:

- open-source — открыт ли исходный код решения;
- поддержка А. П. — поддерживается ли работа с ассоциативными правилами;
- производительность — использует ли инструмент высокопроизводительные технологии;
- доступность — какие действия требуются, чтобы начать пользоваться инструментом;
- интерфейс — графический (GUI) или программный (API).

Таблица 1: Сравнение различных инструментов для анализа данных

	open-source	поддержка А. П.	производительный	доступность	интерфейс
Metanome	да	нет	нет	развертывание	GUI
SAP PA	нет	да	да	установка	GUI
SAS EM	нет	да	да	установка	GUI
mlxtend	да	да	нет	установка	API
arules	да	да	да	установка	API
Desbordante	да	да	да	online	GUI

Поддержка ассоциативных правил в Desbordante была реализована в рамках данной работы, поэтому в таблице отмечено её наличие. Таким образом, можно заключить, что Desbordante по сочетанию параметров выделяется на фоне аналогов: это высокопроизводительный open-source инструмент с поддержкой ассоциативных правил, графическим интерфейсом и возможностью использования без установки и развертывания.

2. Описание решения

2.1. Архитектура

UML-диаграмма архитектуры решения представлена на рисунке 1.

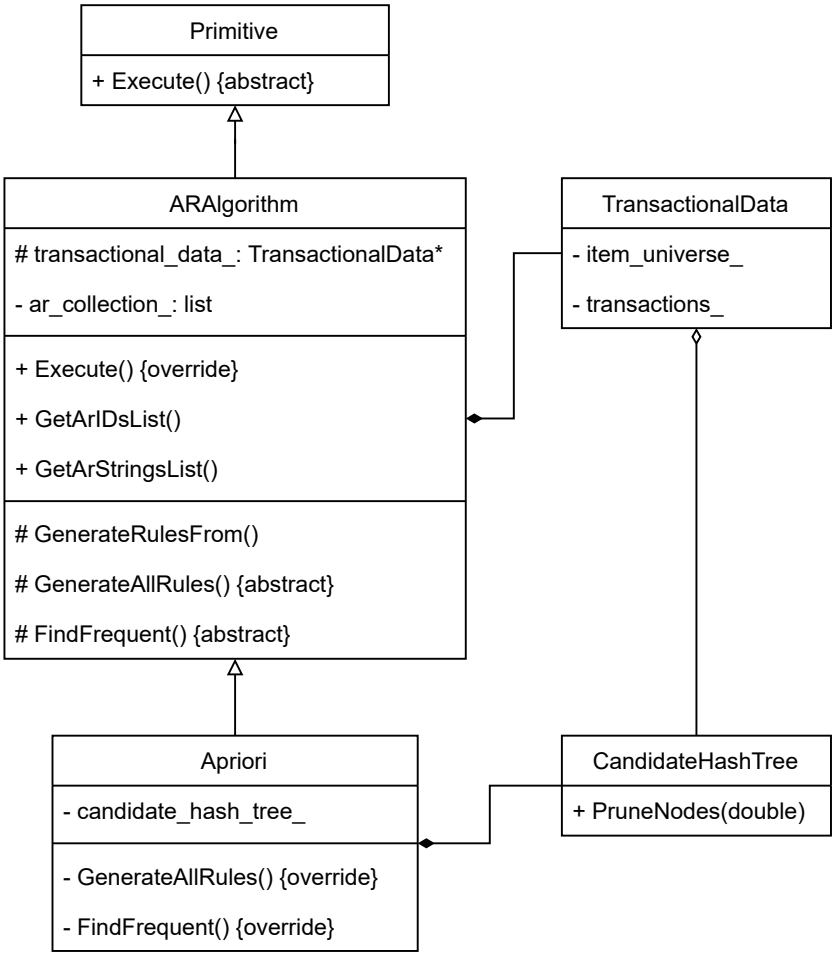

Рис. 1: Диаграмма классов

2.1.1. Общая инфраструктура

В кодовой базе Desbordante базовый класс для различных методов профилирования представлен классом `Primitive`. От него был унаследован класс `ARAlgorithm` — абстрактный класс, который спроектирован как базовый класс для всех алгоритмов поиска ассоциативных правил. Конструктор этого класса принимает две основные константы *minconf* и *minsup*, необходимые для генерации правил. Выполнение поиска производится при вызове переопределенного метода `Execute()`,

который изначально был определен абстрактным в классе `Primitive`. Класс представляет поиск ассоциативных правил в виде двух шагов, как было описано ранее: делегирует процедуру поиска частых наборов с поддержкой не меньше чем *minsup* некоторому классу-потомку, а затем на основе полученных наборов генерирует все ассоциативные правила, у которых доверие не меньше чем *minconf*. Для генерациии множества частых наборов F определен абстрактный метод `FindFrequent()`. Метод `GenerateAllRules()` служит для выполнения второго шага генерации. Он также определён абстрактным, позволяя классу-потомку самостоятельно произвести обход найденного множества F. Однако как логика генерации правил из некоторого частного набора (метод `GenerateRulesFrom`), так и хранилище найденных правил (поле `ar_collection_`) находятся непосредственно внутри самого класса `ARAlgorithm`.

Одним из полей этого класса являтся объект класса `TransactionalData`. Он представляет транзакционную базу данных: поле `item_universe_` соответствует множеству всех предметов I, оно реализовано в виде массива строк-имен предметов. Поле `transactions_` соответствует множеству транзакций D. Список входящих в транзакцию предметов хранится в виде массива чисел, которые отображаются в соответствующие им строки строки на основе индексации в массиве `item_universe_`. Для каждой транзакции также хранится и её уникальный идентификтор. У класса отсутствуют публичные конструкторы, исключение составляет лишь конструкторы копирования. Поэтому экземпляр этого класса можно создать только с помощью статического метода `CreateFrom`, который принимает на вход csv-файл и конфигурацию с форматом входных данных.

Отдельно стоит упомянуть про форматы входных данных. На данный момент поддерживается два типа, называемые `Singular` и `Tabular`. Оба варианта могут быть представлены в csv-файле, однако, имеют несколько различную структуру. Формат `Singular` подразумевает таблицу с двумя колонками. В первой хранится целое число — уникальный идентификатор транзакции, во второй — строка с названем предмета. База данных создается следующим образом: читается уникальный

идентификатор из первого столбца, а затем в транзакцию, соответствующую этому идентификатору, добавляется предмет из второго столбца этой же строки. Второй тип входных данных называется `Tabular`. Он тоже представляет собой таблицу, однако, здесь каждая транзакция представлена одной строкой. В первом столбце строки может быть ункальный идентификатор транзакции, а во всех остальных перечислены предметы, которые входят в эту транзакцию, по одному в каждом столбце. Формат `Singular` больше популярен в промышленных системах, однако наборы данных с форматом `Tabular` тоже распространены.

2.1.2. Алгоритм Apriori

Реализация алгоритма Apriori представлена в классе `Apriori`, который унаследован от класса `ARAlgorithm`. Его задача заключается в том, чтобы найти все частые наборы, то есть множество F. Реализация опирается на Enumeration-Tree представление алгоритма — построение решетки подмножетств происходит уровень за уровнем при помощи древовидной структуры, которую можно интерпретировать как префиксное дерево. Генерация множества F происходит при помощи переопределенного метода `FindFrequent()`. По окончании работы этого метода найденные наборы хранятся в дереве, обход которого реализован в переопределенном методе `GenerateAllRules()`. Оба этих метода вызываются в методе `Execute()` класса `ARAlgorithm`. При обходе дерева для каждого из частых наборов вызывается метод родительского класса `GenerateRulesFrom()`, который генерирует ассоциативные правила из этого набора и добавляет их в хранилище `ar_collection_`.

Хеш-дерево, используемое для вычисления поддержки и генерации множетва F, представлено классом `CandidateHashTree`. Эта структура создается каждый раз при генерации нового уровня Enumeration-Tree дерева, позволяя сделать только один проход по базе данных на этом уровне. За счёт этого повышается эффективность вычисления поддержки для каждого узла-подмножества. Те узлы, для которых поддержка оказалась меньше чем $minsup$, удаляются из дерева. Для того чтобы иметь возможность работать с базой данных, этот класс содержит ука-

затель на объект класса `TransactionalData` — на тот же самый, который хранится в `ARAlgorithm` и был упомянут ранее.

2.2. Тестирование

Для проведения тестов было подготовлено пять наборов данных. Часть из них была создана вручную для проверки особых случаев, часть была взята с ресурса [8]. Всего было подготовлено пять тестов, по одному набору данных на каджды, причем для всех тестов изначально заданы параметры *minsup* и *minconf*. Один тест состоит из двух фаз, согласуясь с процессом поиска ассоциативных правил. На первой фазе ищутся все частые наборы с параметром *minsup* и проверяется, совпадает ли список найденных (множество F) со ожидаемым списком. Если на этой фазе списки получаются разные, то тест считается проваленным и завершается. На второй фазе с помощью F генерируются ассоциативные правила с параметром *minconf* и аналогично происходит проверка на равенство полученого и ожидаемого списков. В обоих случаях ожидаемые списки были сгенерированы путём запуска на тех же наборах данных алгоритма поиска ассоциативных правил из библиотеки mlextend [9]. Все тестирование автоматизировано при помощи фреймворка GoogleTest [7], а для проверки списков на соответствие были реализованы методы `CheckFrequentListsEquality` и `CheckAssociationRulesListsEquality`.

3. Первичное экспериментальное исследование

Кроме проверки на корректность работы реализованного алгоритма было решено провести первичное экспериментальное исследование производительности и сравниться с одним из аналогов. Для сравнения была выбрана библиотека `mlxtend`, написанная на Python, которая ранее была использована для генерации ожидаемой выдачи алгоритма.

Запуск алгоритмов производился на тестовом стенде со следующими характеристиками:

- операционная система: Manjaro Linux 20.2.1 (Linux kernel 5.13.19-2-MANJARO), gcc version 11.1.0;
- процессор: x86_64 Intel Core i5-8300H 4 cores 8 threads, 4.00 GHz 8 MiB L3 cache;
- оперативная память: 16 GiB DDR4.

Для тестирования был выбран набор данных[4], содержащий 9835 транзакций. Стоит отметить, что при тестировании не учитывалось время обработки входного файла. Задача первичного тестирования — сравнить примерный порядок времени выполнения алгоритмов при различных входных параметрах, поэтому погрешность и доверительные интервалы не были подсчитаны.

Таблица 2 показывает время выполнения алгоритмов при различном значении $minsup$ и зафиксированном значении $minconf = 0.2$. При уменьшении параметра $minsup$ увеличивается количество узлов алгебраической решетки, которые необходимо рассмотреть при генерации множества частых набров F. Этим и объясняется увеличение времени работы. Заметно, что при наибольшем значении параметра $minsup = 0.015$ и, соответственно, малом размере множества F время работы различается не так существенно, как при малом значении $minsup$ и большом размере F.

[4] `https://www.kaggle.com/datasets/samanemami/online-transactions-of-the-electronic-devices`

В таблице 3 приведено время выполнения алгоритмов при различных значениях *minconf* и зафиксированном $minsup = 0.0015$. При изменении параметра время выполнения остатся практически одинаковым. Это объясняется тем, при генерации ассоциативных правил на основе параметра *minconf* во всех трех случаях алгортму необходимо перебрать **все** частые наборы из множества F, которое здесь одинаково, и, соответственно, время на его генерацию тоже неизменно. От параметра *minconf* зависит только то, какие из сгенерированных правил будут выданы в качестве ответа, а какие отброшены, и в данном случае это влияет на производительность довольно несущественно.

Таблица 2: Время работы в секундах при различных значениях *minsup* (указаны в заголовке таблицы) и фиксированом $minconf = 0.2$

	0.015	0.005	0.0015	0.0012
mlxtend	0.2	1.3	14.7	20.3
Desbordante	0.1	0.4	1.4	1.9

Таблица 3: Время работы в секундах при различных значениях *minconf* (указаны в заголовке таблицы) и фиксированом $minsup = 0.0015$

	0.2	0.6	0.9
mlxtend	14.7	13.7	13.6
Desbordante	1.4	1.4	1.4

Заключение

В ходе работы над реализацией функционала для поиска ассоциативных правил в рамках Desbordante были выполнены следующие задачи:

- произведён краткий обзор предметной области и алгоритма поиска ассоциативных правил Apriori;
- разработана и реализована архитектура для поиска ассоциативных правил в рамках существующей кодовой базы Desbordante;
- реализован алгоритм поиска ассоциативных правил Apriori;
- подготовлены юнит-тесты и проведено первичное экспериментальное исследование производительности.

Исходный код инструмента доступен[5] на Github. Файлы распределены по разным директориям, код самого алгоритма находится в src/algorithms/association-rules, код класса ARAlgorithm — в src/algorithms.

[5] `https://github.com/Mstrutov/Desbordante`, пользователь `alexandrsmirn`

Список литературы

[1] Abedjan Ziawasch, Golab Lukasz, and Naumann Felix. Profiling Relational Data: A Survey // The VLDB Journal. — 2015. — aug. — Vol. 24, no. 4. — P. 557–581. — Access mode: https://doi.org/10.1007/s00778-015-0389-y.

[2] Aggarwal Charu C. Data Mining: The Textbook. — Springer Publishing Company, Incorporated, 2015. — ISBN: 3319141414.

[3] Agrawal Rakesh and Srikant Ramakrishnan. Fast Algorithms for Mining Association Rules // Proc. 20th Int. Conf. Very Large Data Bases VLDB. — 2000. — 08. — Vol. 1215.

[4] Arules package. — Access mode: https://www.rdocumentation.org/packages/arules/versions/1.7-3 (online; accessed: 2022-05-25).

[5] Papenbrock Thorsten, Bergmann Tanja, Finke Moritz, Zwiener Jakob, and Naumann Felix. Data Profiling with Metanome // Proc. VLDB Endow. — 2015. — aug. — Vol. 8, no. 12. — P. 1860–1863. — Access mode: https://doi.org/10.14778/2824032.2824086.

[6] Strutovskiy Maxim, Bobrov Nikita, Smirnov Kirill, and Chernishev George. Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms // 2021 29th Conference of Open Innovations Association (FRUCT). — 2021. — P. 344–354.

[7] GoogleTest framework. — Access mode: https://github.com/google/googletest (online; accessed: 2022-05-25).

[8] Kaggle — платформа по исследованию данных. — Access mode: https://www.kaggle.com/ (online; accessed: 2022-05-25).

[9] Raschka Sebastian. MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack // The Journal of Open Source Software. — 2018. — Apr. — Vol. 3,

no. 24. — Access mode: http://joss.theoj.org/papers/10.21105/joss.00638.

[10] SAP Predictive Analytics. — Access mode: https://help.sap.com/docs/SAP_PREDICTIVE_ANALYTICS?version=3.0 (online; accessed: 2022-05-25).

[11] SAS Enterprise Miner. — Access mode: https://www.sas.com/ru_ru/software/enterprise-miner.html (online; accessed: 2022-05-25).