Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 21.Б08-мм

Интеграция эволюционного алгоритма поиска численных ассоциативных правил в профайлер Desbordante

ВОЛГУШЕВ Иван Романович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ассистент кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2024

Оглавление

Введение

Поиск ассоциативных правил (association rule discovery) — это метод машинного обучения [3] (вариант обучения без учителя), применяемый для анализа транзакционных наборов данных, который используется как в бизнес-аналитике, так и в науке. Интуитивное определение понятия ассоциативного правила делает поиск ассоциативных правил мощным инструментом для создания заключений о корреляциях атрибутов в больших объёмах данных.

Ассоциативные правила определены на транзакционных наборах данных — наборах, представляющих собой какое-то количество транзакций из фиксированного домена предметов. Каждая транзакция может либо включать предмет из домена, либо нет. О транзакционной базе данных можно думать, как о базе данных чеков из супермаркета, где каждая колонка соответствует типу товара из каталога, а каждая строчка соответствует чеку. Для каждой ячейки указано значение «истина», если в соответствующем чеке этот товар присутствовал, и «ложь», если не присутствовал. Если присутствие товара A в чеке с вероятностью P влечёт присутствие товара B, то говорят, что товары A и B связаны ассоциативным правилом $A \Rightarrow B$ с уровнем доверия (confidence) P.

Анализ ассоциативных правил может привести к открытию неожиданных фактов о данных. Desbordante [2] — наукоёмкий профайлер данных с открытым исходным кодом, с помощью которого возможен поиск множества так называемых примитивов. Примитив — некоторое правило, показывающее закономерности в данных. Ассоциативные правила являются одним из видов примитивов, быстрый поиск которых возможен при помощи Desbordante.

Существует множество алгоритмов для поиска ассоциативных правил, среди них Apriori [5] и Eclat [1]. Apriori уже интегрирован в Desbordante. Однако, определение ассоциативных правил допускает лишь значения «истина» и «ложь» во входном наборе данных. Для алгоритмов поиска ассоциативных правил чеки на покупку десяти молотков и одного являются одним и тем же объектом.

Численное ассоциативное правило (numeric association rule) — обобщение понятия ассоциативного правила. В случае численных ассоциативных правил, в каждой ячейке базы данных может стоять не только значение «истина» или «ложь», но и любое численное значение или строка. С помощью поиска такого примитива возможно, например, сделать выводы не только о том, какие товары покупаются вместе, но и в каком количественном диапазоне. Например, что клиенты, которые тратят на кофе более 500 рублей в месяц, с высокой вероятностью приобретают также чай на сумму более 300 рублей. Такие заключения могут оказаться ещё полезнее. В целях расширения функционала Desbordante было принято решение добавить в ряд имеющихся алгоритмов алгоритм поиска числовых ассоциативных правил.

В статье [3] авторы описали свою платформу для предобработки данных и поиска на них ассоциативных правил с последующей визуализацией, называемой uARMSolver, реализация которой доступна на GitHub[1]. Авторы также описали популяционный стохастический алгоритм, основанный на широко используемом в машинном обучении алгоритме оптимизации — дифференциальной эволюции (differential evolution). С его использованием возможно производить неисчерпывающий поиск численных ассоциативных правил. Неисчерпывающий поиск подразумевает, что не все существующие на наборе данных численные ассоциативные правила будут найдены, а лишь какая-то их часть. Исчерпывающие же алгоритмы находят все правила, удовлетворяющим некоторым заданным ограничениям. Было принято решение интегрировать этот алгоритм в Desbordante, улучшив его, используя стандарты разработки, ориентированные на долгосрочную поддержку кодовой базы.

[1] https://github.com/firefly-cpp/uARMSolver (дата доступа: 6 ноября 2024 г.)

1. Постановка задачи

Целью работы стала интеграция алгоритма поиска численных ассоциативных правил с использованием дифференциальной эволюции, предоставленного авторами статьи [3] в систему Desbordante. Для этого были поставлены следующие задачи:

- Провести анализ предметной области;
- Провести анализ реализации алгоритма в uARMSolver и выявить недостатки;
- Интегрировать алгоритм в профайлер Desbordante;
- Произвести тестирование и сравнение полученной реализации и реализации из статьи [3].

2. Обзор

2.1. Основные определения

Введём основные понятия, которые будут использоваться в дальнейшем изложении. Определения приведены по [4].

Определение 1. *Действительными и целочисленными атрибутами называют атрибуты, домен которых это конечное подмножество целых или действительных чисел соответственно.*

Определение 2. *Категориальными атрибутами называют атрибуты, домен которых это конечное множество строк.*

Определение 3. *Численное ассоциативное правило — это импликация вида $X \Rightarrow Y$, где антецедент и консеквент это множества атрибутов вида $A = [v_1, v_2]$, если A — численный атрибут, и вида $A = \{v_1, v_2, \dots v_n\}$, если A — категориальный атрибут.*

Приведём пример численного ассоциативного правила из [4].

$$Age \in [21, 35] \wedge Gender : [Male] \Rightarrow Salary \in [2000, 3000] \qquad (1)$$

Это правило гласит, что сотрудники мужского пола возрастом от 21 до 35 лет получают зарплату от 2000$ до 3000$. Для оценки численных правил введём определения для некоторых характеристик.

Определение 4. ***Поддержкой*** *или* ***support*** *правила называют отношение количества кортежей в наборе данных, удовлетворяющих ограничениям на антецедент и консеквент к общему количеству кортежей.*

Определение 5. ***Доверием*** *или* ***confidence*** *правила называют отношение количества кортежей в наборе данных, удовлетворяющих ограничениям на антецедент и консеквент к количеству кортежей, удовлетворяющих лишь условию антецедента.*

Определение 6. ***Включением*** *или* ***inclusion*** *правила называют отношение числа атрибутов, включённых в правило, к общему числу атрибутов в наборе данных.*

Согласно этим определениям рассчитаем поддержку, доверие и включение численного ассоциативного правила (1) на представленном ниже наборе данных.

Таблица 1: Датасет “Employees”

Age	Gender	Salary
30	M	3500
34	F	4000
26	M	2000
20	F	1500
24	M	2000
31	M	3000

Тогда для (1) $support = 0.5$, $confidence = 0.75$ и $inclusion = 1.0$. По вычисленным характеристикам можно сделать следующие выводы: половина сотрудников — мужчины возрастом от 21 до 35 лет с зарплатой от 2000\$ до 3000\$. Если сотрудник мужчина от 21 до 35 лет, то с вероятностью 75% его зарплата находится в промежутке от 2000\$ до 3000\$. И наконец, ассоциативное правило (1) связывает все атрибуты таблицы.

Определение 7. ***Приспособленностью*** *или* ***fitness*** *численного ассоциативного правила $X \Rightarrow Y$ будем называть представленное ниже отношение.*

$$f(X \Rightarrow Y) = \frac{\alpha * support(X \Rightarrow Y) + \beta * fitness(X \Rightarrow Y) + \gamma * inclusion(X \Rightarrow Y)}{3} \quad (2)$$

Где α, β и γ — параметры, выставляемые пользователем, сумма которых равна трём. Приспособленность численного ассоциативного правила и будет целевой функцией для эволюционного алгоритма.

2.2. Описание алгоритма

Дифференциальная эволюция — широко известный метод оптимизации. В конкретном случае поиска численных ассоциативных правил суть алгоритма заключается в определении трёх основных функций: $decode$, $crossover$ и $fitness$.

1. Функция $decode : R^n \to X \Rightarrow Y$ сопоставляет вектору действительных чисел (генов) в отрезке $[0, 1]$ некоторое численное ассоциативное правило. Пространство R^n будем называть пространством решений. При этом n меняется в зависимости от количества и типа атрибутов в конкретном наборе данных, а сама функция зависит от размера доменов атрибутов. Функцию необходимо выбрать таким образом, чтобы близкие по евклидовой метрике векторы в пространстве решений отображались в похожие в каком-то смысле численные ассоциативные правила.

2. Функция $crossover : R^n \times R^n \times \ldots \times R^n \to R^n$ комбинирует несколько векторов из пространства решений, создавая новое решение, включающее в себя свойства исходных решений.

3. Функция $fitness : X \Rightarrow Y \to [0, 1]$ это функция приспособленности. Она должна отражать информативность и полезность конкретного численного ассоциативного правила. Она уже была упомянута в определении 7.

Алгоритм состоит из следующих основных шагов:

1. Случайно сгенерировать изначальную популяцию $P \subset R^n$.

2. Выбрать из популяции P следующего по порядку кандидата A.

3. Выбрать из популяции P случайные образцы $R_1, R_2, \ldots, R_k$.

4. Получить новое решение $B = crossover(A, R_1, R_2, \ldots, R_k)$.

5. Декодировать решение B и рассчитать приспособленность: $f = fitness(decode(B))$.

6. Если значение $fitness$ нового решения B превосходит значение $fitness$ кандидата A, заменить кандидата A на новое решение B в популяции P: $P = P \setminus A \cup B$ и занести B в список обнаруженных численных ассоциативных правил.

7. Если достигнут лимит числа итераций, закончить работу. Иначе вернуться к шагу 2.

Представленное ниже изображение из статьи [3] иллюстрирует работу функции *decode*.

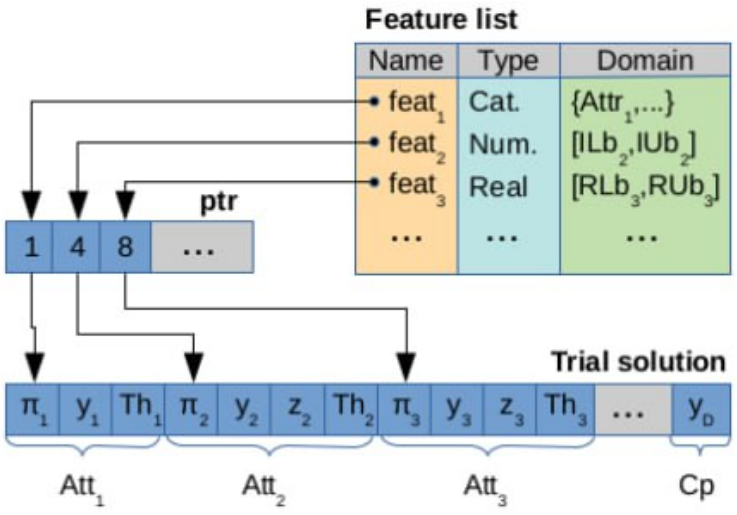

Рис. 1: Принцип действия функции *decode*

Для каждого из атрибутов набора данных $Att_1, Att_2, Att_3 \ldots$ в векторе выделяется три действительных числа, если атрибут категориальный, и четыре, если атрибут целочисленный или действительный. Для атрибута Att_k число π_k называется перестановкой. После декодировки ограничения на атрибуты располагаются в результирующем правиле в порядке возрастания их перестановки. Th_k — это так называемый порог (threshhold). При декодировке ограничение на атрибут войдёт в результат с вероятностью, равной порогу. Наконец, y_k для категориальных атрибутов или y_k и z_k для численных атрибутов отвечают за конкретное ограничение, которое будет наложено на $Attr_k$. Верхнюю и нижнюю границу ограничения, наложенного на численный атрибут или

за единственную строку, которой должен быть равен категориальный атрибут. Благодаря такому алгоритму близкие по евклидовой метрике векторы из пространства решений окажутся похожими численными ассоциативными правилами.

Функция *crossover* может быть выбрана разными способами. Имплементация uARMSolver содержит 6 различных способов скрещивания индивидов, но автор настоящей практической работы выбрал часто используемую биномиальную схему кроссовера, имеющую два параметра: масштаба $scale$ и вероятности кроссовера Cr. Необходимо выбрать кандидата A и три случайных образца R_1, R_2, R_3. Каждый ген $A[i]$ кандидата A с вероятностью Cr может приобрести значение:

$$A[i] = R_1[i] + scale * (R_2[i] - R_3[i]), \tag{3}$$

либо остаться неизменным в новом индивиде.

2.3. Анализ реализации uARMSolver

Как было упомянуто ранее, авторы статьи [3] опубликовали свою реализацию[2] эволюционного алгоритма поиска численных ассоциативных правил на языке C++. Возможность заимствования исходного кода была рассмотрена автором курсовой практической работы. Однако, при анализе исходного кода uARMSolver был установлен ряд недостатков, усложняющих понимание кода и нарушающих принципы объектно-ориентированного программирования. Среди них: названия переменных и функций, не отражающие их назначение, отсутствие инкапсуляции, повторная реализация функций, доступных в стандартной библиотеке. Примеры многих из исправленных недостатков содержатся в представленном ниже фрагменте кода uARMSolver:

```
1  . . .
2  /**
3  * Initialize association rule class.
4  *
```

[2] https://github.com/firefly-cpp/uARMSolver (дата доступа: 6 ноября 2024 г.)

```
 5  * @param the sequence number, on which permutation bases, random value
      determining the permutation ordering of features.
 6  * @return no return code.
 7  */
 8 void Rule::init(int i, double x)
 9 {
10   perm.push_back(i);
11   val.push_back(x);
12   norm.push_back(0);
13 }
14  . . .
```

Листинг 1: Rule.cpp

Представленный выше фрагмент кода содержит метод `init()` класса `Rule`, являющегося классом представления численного ассоциативного правила. Хоть метод и называется `init()` и принадлежит классу `Rule`, он не инициализирует объект `Rule`, а добавляет индекс атрибута `i` в один вектор внутри `Rule` и перестановку этого атрибута `x` в другой вектор внутри `Rule`. Имя метода едва ли отражает его назначение. Этот метод в ходе работы программы вызывается один раз для каждого атрибута таблицы, но перед тем, как вектор перестановки станет пригодным для использования, необходимо вызвать у `Rule` метод `sort()`. При этом оба вектора являются публично доступными полями `Rule`. Таким образом, класс `Rule` перекладывает ответственность за согласованность данных внутри себя на своего пользователя, что является нарушением принципа инкапсуляции. Названия параметров также не содержат информации об их назначении, однако функция имеет комментарий над собой, поясняющий их суть. Комментарии необходимо актуализировать при каждом изменении в коде, который они поясняют, что усложняет поддержку и расширение кода. Руководствуясь принципами ООП можно сказать, что многократные вызовы этого метода должны быть переложены на один вызов конструктора, который и должен привести `Rule` в согласованное состояние. Однако у `Rule` есть только конструктор по умолчанию.

Таким образом, было выявлено, что исходный код uARMSolver нуж-

дается в серьёзной реструктуризации, прежде чем он сможет быть интегрирован в Desbordante. Desbordante приоритизирует простоту долговременной поддержки, читаемости и расширяемости исходного кода. Такой подход связан с расчётом на длительный жизненный цикл проекта. Кроме того, Desbordante — это программное обеспечение с открытым исходным кодом. Преимуществом такого подхода является то, что разработка может вестись целиком дистанционно любым заинтересованным лицом. Однако это означает, что люди, совершившие вклад, могут покинуть проект в любой момент, а с ними уйдёт и знание об устройстве их части кода. Таким образом, верные парадигмы предотвращают появление “чёрных ящиков” — разделов исходного кода, об устройстве которых не осведомлён ни один из активных разработчиков, восстановление знаний о которых требует значительных трудозатрат.

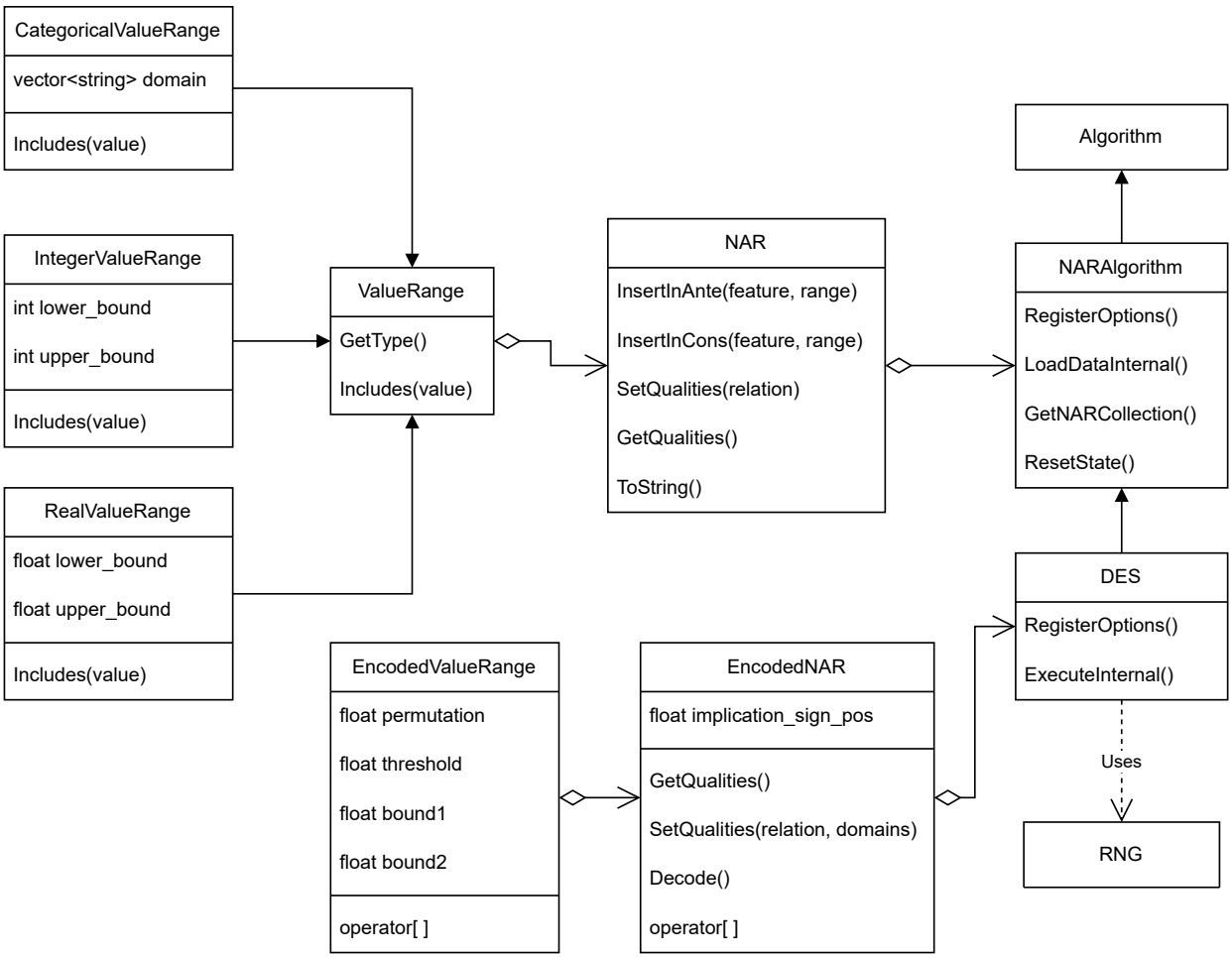

Рис. 2: UML диаграмма DES

3. Реализация

Новый алгоритм поиска численных ассоциативных правил было решено назвать Differential Evolution Solver (DES). На представленной выше UML диаграмме можно увидеть классовую структуру, окружающую DES. В Desbordante данные о данных, извлекаемые из таблиц и графов называются примитивами, а их поиск называется майнингом (mining). Все алгоритмы майнинга примитивов и алгоритмы других типов наследуются от абстрактного класса `Algorithm`. Функции и поля, относящиеся к поиску численных ассоциативных правил, а не конкретно к алгоритму дифференциальной эволюции, были вынесены в отдельный абстрактный класс `NARAlgorithm`. Написание этого промежуточного класса облегчит добавление будущих алгоритмов поиска численных ассоциативных правил, которые смогут унаследовать его функционал.

`Algorithm` объявляет множество интерфейсов и полезных функций. К ним относится `RegisterOption()` — функция для передачи адреса поля для дальнейшего выставления системой опций Desbordante будь то через командную строку, Python интерфейс, или изнутри Desbordante.

```
. . .
unsigned long long DES::ExecuteInternal() {
    FeatureDomains feature_domains = FindFeatureDomains(typed_relation_.get());
    std::vector<EncodedNAR> population = GetRandomPopulationInDomains(feature_domains);

    for (int i = 0; i < num_evaluations_; i++) {
        size_t candidate_i = i % population_size_;
        EncodedNAR mutant = MutateIndividual(population, candidate_i);
        NAR mutant_decoded = mutant.SetQualities(feature_domains, typed_relation_.get());
        double candidate_fitness = population[candidate_i].GetQualities().fitness;

        if (mutant.GetQualities().fitness > candidate_fitness) {
            population[candidate_i] = mutant;
            nar_collection_.emplace_back(mutant_decoded);
        }
    }

    auto CompareByFitness = [](const NAR& a, const NAR& b) -> bool {
        return a.GetQualities().fitness > b.GetQualities().fitness;
    };
    std::sort(nar_collection_.begin(), nar_collection_.end(), CompareByFitness);
    return 0;
}
. . .
```

Листинг 2: DES::ExecuteInternal

Представленный выше код является основной функциональной частью DES, функция `ExecuteInternal` вызывается после выставления всех опций и чтения входных данных и заполняет поле `nar_collection_`.

Структура функции совпадает с последовательностью шагов алгоритма описанной в пункте 2.2.

В реализации uARMSolver существовал класс для представления численного ассоциативного правила — `Rule`. Для представления элементов пространства решений использовались структуры `std::vector<double>`. Новая реализация имеет два класса представления правила: `NAR` и `EncodedNAR`. Для представления численных ассоциативных правил и элементов пространства решений R^n соответственно.

```
1 . . .
2 public:
3     size_t VectorSize();
4     size_t FeatureCount();
5     double& operator[](size_t index);
6     double const& operator[](size_t index) const;
7
8     model::NARQualities const& GetQualities() const;
9     model::NAR SetQualities(FeatureDomains domains, TypedRelation const*
      typed_relation);
10
11    model::NAR Decode(FeatureDomains domains) const;
12    EncodedNAR(FeatureDomains domains, TypedRelation const* typed_relation);
13    EncodedNAR(size_t feature_count);
14 private:
15 . . .
```

Листинг 3: публичный интерфейс EncodedNAR

Рассмотрим класс `EncodedNAR`. В нём есть приватное поле `qualities_`, которое является структурой, объединяющей в себе значения *support*, *confidence* и *inclusion*. Для расчёта этих полей пользователю необходимо вызвать функцию `SetQualities(TypedRelation const* typed_relation)`, передав ей константный указатель на датасет. Для возвращения значения необходимо вызвать функцию `GetQualities()`. При этом GetQualities вернёт значение только в случае, если до этого была вызвана функция `SetQualities(TypedRelation const* typed_relation)` и после этого `EncodedNAR` не изменялся при помощи `operator[]`, иначе `GetQualities()` вызовет исключение времени выполнения. Та-

ким образом, оценочные значения инкапсулируются и исключается возможность получения нецелостных данных пользователем без генерации исключения, что облегчает поиск ошибок.

Кроме того, перегруженный оператор подстрочного индекса "`[ ]`" позволяет обращаться с экземпляром `EncodedNAR` в точности, как с вектором действительных чисел, используя краткий синтаксис. Таким образом, в случаях, когда все гены закодированного правила равнозначны, например, в функции кроссовера, можно воспользоваться оператором подстрочного индекса для последовательного обращения ко всем публичным полям класса. В случае же, когда необходимо обратиться к гену, ответственному за определённый внешний признак правила, например, при декодировке решения, возможно обратится к конкретному гену через его название. Таким образом класс инкапсулирует свои данные и с пользователя снимается необходимость знать конкретный порядок расположения генов и их количество, что позволяет менять внутреннее расположение и количество генов, не инвалидируя пользовательский код, чего нельзя сказать в случае использования `std::vector<double>`.

4. Эксперимент

Созданная автором реализация алгоритма DES и реализация uARMSolver были протестированы на трёх наборах данных с лимитом оценок функции приспособленности 1000 и размером популяции 100. Были получены следующие результаты времени выполнения:

Таблица 2: Полное время работы

	столбцы	строки	время uARMSolver	время DES
Abalone.csv	9	4177	16840 мс	1819 мс
iris.csv	5	150	243 мс	40 мс
breast_cancer.csv	30	569	23400 мс	1470 мс

В течение этих трёх запусков DES завершал свою работу более, чем в шесть раз быстрее.

Далее, рассмотрим результаты, полученные двумя алгоритмами. Численное ассоциативное правило с самым высоким значением приспособленности, обнаруженное DES на наборе данных Abalone:

$$Wholeweight \in [0.002, 1.278] \wedge Shuckedweight \in [0.001, 0.930] \wedge$$
$$Viscera weight \in [0.001, 0.760] \wedge Rings \in [3, 15] \Rightarrow$$
$$Diameter \in [0.069, 0.512]$$

$$support = 0.773043, confidence = 0.986557, fitness = 0.771718$$

Численное ассоциативное правило с самым высоким значением приспособленности, обнаруженное uARMSolver на тех же данных:

$$Diameter \in [0.123, 0.624] \wedge Height \in [0.000, 0.167] \wedge$$
$$Length \in [0.075, 0.075] \wedge Rings \in [4, 29] \wedge$$
$$Shellweight \in [0.121, 0.830] \wedge Shuckedweight \in [0.433, 0.773] \wedge$$
$$Wholeweight \in [0.877, 0.994] \Rightarrow Viscera weight \in [0.069, 0.512]$$

$$support = 0.999761, confidence = 1, fitness = 0.962883$$

Заявленная приспособленность правила, обнаруженного uARMSolver выше приспособленности правила, обнаруженного при помощи DES. Однако полученное правило содержит ограничение $Length \in [0.075, 0.075]$, то есть длина должна быть строго равна 0.075. При этом заявленная поддержка этого правила $0.999761 = 1 - \frac{1}{4177}$. То есть это правило гласит, что лишь одна строка из всех 4177 строк в наборе данных не содержит значение 0.075, что неверно. Для прояснения было проведено тестирование на следующем наборе данных:

Таблица 3: Тестовый набор данных

Name	Age	Salary
Boris	50	10000
Denis	60	20000
Anastasia	28	3000
Gleb	21	25000
Ivan	70	500000
George	25	20000
Alex	26	30000
Kostya	27	1000
Anatoliy	28	1
Gordon	23	53300
Chell	25	52500

Всего после работы uARMSolver было выведено 206 численных ассоциативных правил. Фрагмент результата представлен ниже.

```
1  .  .  .
2  0.888889  ['Wage_320370_403206']=>['Age_29_42']  1  1
3  0.888889  ['Age_21_46']=>['Wage_272405_500000']  1  1
4  0.888889  ['Age_21_45']=>['Wage_1_239074']  1  1
5  0.888889  ['Age_21_46']=>['Wage_425109_500000']  1  1
6  0.888889  ['Age_35_65']=>['Wage_87289_173260']  1  1
7  0.69697  ['Name_Anastasia','Wage_50599_450271']=>['Age_24_70']  0.0909091  1
8  0.69697  ['Age_37_58','Name_Ivan']=>['Wage_1_500000']  0.0909091  1
9  0.69697  ['Age_43_60','Name_Kostya']=>['Wage_47379_500000']  0.0909091  1
10  0.69697  ['Age_63_64','Name_Boris']=>['Wage_1_134145']  0.0909091  1
11  0.69697  ['Name_Denis','Wage_241241_500000']=>['Age_21_44']  0.0909091  1
```

Листинг 4: Фрагмент вывода uARMSolver

Полученные правила имеют поддержку, равную одному из двух значений: либо 1, либо $0.0909091 = \frac{1}{11}$, где 11 — число кортежей в наборе данных, при чём все правила, налагающие ограничения только на численные атрибуты, имеют поддержку 1, а все правила, налагающие ограничение на имя, имеют поддержку 0.0909091, поскольку повторений имён в наборе данных нет. Из этого можно сделать вывод, что поддержка для правил рассчитывается без учёта ограничений на численные атрибуты. Дальнейший анализ исходного кода uARMSolver подтвердил эту гипотезу. Функция, определяющая включение значения атрибута в ограничения правила `isIncluded(i, j, rule)` для численных атрибутов проверяет их вхождение в домен атрибутов, а не в ограничение правила `rule`. Автор курсовой работы не имеет гипотез в отношении поведения uARMSolver при тестировании на наборе данных Abalone.

Очевидно, что полученные uARMSolver характеристики численных ассоциативных правил некорректны. Возможно, обнаружению этой ошибки авторами работы [3] помешало отсутствие вышеупомянутых парадигм написания кода.

Пример правила, обнаруженного на этом наборе данных при помощи DES, представлен ниже.

$$Salary \in [1, 461176] \Rightarrow Age \in [21, 59]$$

$$support = 0.818182, confidence = 0.900000, fitness = 0.794949$$

Заключение

В ходе работы в систему Desbordante был интегрирован алгоритм поиска численных ассоциативных правил. Этот алгоритм использует дифференциальную эволюцию и его исходная версия была предоставлена авторами статьи [3]. Автором настоящей работы были выполнены следующие задачи:

- Проведён анализ предметной области;
- Проведён анализ реализации алгоритма в uARMSolver и были выявлены недостатки;
- Алгоритм интегрирован в профайлер Desbordante;
- Произведено тестирование и сравнение полученной реализации и реализации из статьи [3].

Реализация находится на рассмотрении для включения в проект Desbordante. Pull request #490 доступен на GitHub[3].

[3] https://github.com/Desbordante/desbordante-core/pull/490 (дата доступа: 6 ноября 2024 г.).

Список литературы

[1] Borgelt Christian. Frequent item set mining // WIREs Data Mining and Knowledge Discovery. — 2012. — Vol. 2, no. 6. — P. 437–456. — https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1074.

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

[3] Fister Iztok, Fister Iztok. uARMSolver: A framework for Association Rule Mining // ArXiv. — 2020. — Vol. abs/2010.10884. — URL: https://arxiv.org/abs/2010.10884.

[4] Kaushik Minakshi, Sharma Rahul, au2 Iztok Fister Jr., Draheim Dirk. Numerical Association Rule Mining: A Systematic Literature Review. — 2023. — 2307.00662.

[5] Zhou Huanyin, Liu Jinsheng. A Novel Algorithm for Association Rule Mining without Candidate // 2009 International Joint Conference on Artificial Intelligence (JCAI). — Los Alamitos, CA, USA : IEEE Computer Society, 2009. — . — P. 116–119. — URL: https://doi.ieeecomputersociety.org/10.1109/JCAI.2009.16.