Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 21.Б08-мм

Реализация интерфейсов и создание примера использования для инструмента поиска условных функциональных зависимостей в профайлере 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 **3**

1. Постановка задачи **5**

2. Обзор **6**

2.1. Основные определения 6

2.2. Примеры 7

3. Реализация **9**

3.1. Описание реализации интерфейса 9

3.2. Особенности реализации примера 11

3.3. Реализация валидатора CFD 12

4. Апробация **14**

Заключение **16**

Список литературы **17**

Введение

Количество собираемых данных в мире велико, как никогда. Хранится всё, от глобальных метеоданных до времени, потраченного пользователями на странице популярного сайта. Однако, одного взгляда недостаточно, чтобы извлечь из гигабайтов табличных данных полезные заключения, скрытые внутри.

Примитив — описание правил, применяемых к определённой части данных с использованием математических методов. Они позволяют сделать точные и ясные заключения о данных. Примером такого примитива являются функциональные зависимости (functional dependency, FD), которые могут быть описаны как бинарные отношения на множествах столбцов таблицы.

Пусть X, Y — это множества атрибутов таблицы. Говорят, что функциональная зависимость $X \to Y$ выполняется (удерживается), если для любых двух кортежей из таблицы равенство атрибутов X означает равенство атрибутов Y.

Продолжением понятия функциональной зависимости является условная функциональная зависимость (conditional functional dependency, CFD). Это функциональная зависимость, существующая только на подмножестве строк в таблице, удовлетворяющих условию некоторого шаблона. Их поиск имеет множество применений в области баз данных и теории зависимостей и может дать новую неожиданную перспективу на данные. В частности, найденные CFD можно использовать [1] для очистки данных.

Desbordante [2] — наукоёмкий профилировщик данных с открытым исходным кодом и упором на производительность, написанный на C++ и имеющий Python интерфейс и интерфейс коммандной строки (command line interface, CLI). Desbordante позволяет обнаруживать множество примитивов, в числе них — условные функциональные зависимости.

В Desbordante функционал поиска условных функциональных зависимостей представлен единственным алгоритмом FD First [3], но в

будущем количество алгоритмов поиска может увеличиться. Одним из основных приоритетов Desbordante является предоставление удобного интерфейса для применения имеющихся примитивов пользователем, в частности Python интерфейса и web интерфейса[1]. Также Desbordante предоставляет примеры использования Python интерфейса в виде наглядно оформленных файлов-примеров, которые служат документацией для конечного пользователя. Целью данной работы стало добавление Python интерфейса и интерфейса коммандной строки для алгоритмов поиска условных функциональных зависимостей в Desbordante, а также в написании наглядного примера их использования в качестве документации.

[1] https://desbordante.unidata-platform.ru/ (дата доступа: 1 мая 2024 г.)

1. Постановка задачи

Целью работы стало добавление Python интерфейса и интерфейса коммандной строки для имеющихся алгоритмов поиска CFD в систему Desbordante и написание наглядного примера использования. Для этого были поставлены следующие задачи:

1. Реализовать Python-привязки для алгоритмов поиска CFD;
2. Реализовать интерфейс коммандной строки для алгоритмов поиска CFD;
3. Добавить пример использования Python интерфейса;
   (a) Реализовать наивный валидатор CFD;
   (b) Реализовать визуализацию результата;
4. Провести тестирование корректности работы привязок, файла-примера и интерфейса коммандной строки.

2. Обзор

2.1. Основные определения

Определение 2.1. Множество всех кортежей таблицы обозначим D.

Определение 2.2. Условная функциональная зависимость φ это пара вида $(X \to Y, t_p)$, где $X \to Y$ это функциональная зависимость, а t_p это шаблонный кортеж (pattern tuple), в нём ключами являются атрибуты из $X \cup Y$. Атрибуты X обозначаются LHS (left hand side), а атрибуты Y обозначаются RHS (right hand side).

Определение 2.3. Будем говорить, что pattern tuple t_p поддерживает (supports) кортеж t, если для всякого атрибута A, $t_p[A] = t[A]$ или $t_p[A] =$ "_".

Определение 2.4. Поддержка (support) условной функциональной зависимости φ это количество кортежей из D, поддерживаемых t_p из φ. Множество всех поддерживаемых кортежей обозначается как $supp(t_p, D)$.

Определение 2.5. Будем говорить, что CFD φ вида $(X \to Y, t_p)$ удерживается (удовлетворяется) на D с уверенностью равной 1, если для $supp(t_p, D)$ удовлетворяется функциональная зависимость $X \to Y$.

Определение 2.6. Если для CFD φ вида $(X \to Y, t_p)$ не удерживается FD $X \to Y$ на $supp(t_p, D)$, она всё ещё может удерживаться на некотором подмножестве кортежей из $supp(t_p, D)$. Минимальное количество кортежей, которое необходимо убрать из $supp(t_p, D)$, чтобы FD удерживалась, обозначим за $I(\varphi, D)$. Тогда уверенность φ — это значение $\frac{|supp(t_p,D)|-I(\varphi,D)}{|supp(t_p,D)|}$.

Таблица 1: Датасет “Play tennis”

tid	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	false
2	sunny	hot	high	true	false
3	overcast	hot	high	false	true
4	rain	mild	high	false	true
5	rain	cool	normal	false	true
6	rain	cool	normal	true	false
7	overcast	cool	normal	true	true
8	sunny	mild	high	false	false
9	sunny	cool	normal	false	true
10	rain	mild	normal	false	true
11	sunny	mild	normal	true	true
12	overcast	mild	high	true	true
13	overcast	hot	normal	false	true
14	rain	mild	high	true	false

2.2. Примеры

Визуализации для последующх примеров представлены в разделе 4. Таблица взята из статьи [3].

Пример 2.1 Рассмотрим таблицу “Play tennis”, состоящую из пяти столбцов атрибутов: Outlook (облачность), Temperature (температура воздуха), Humidity (влажность воздуха), Windy (ветреность), Play (комфортно ли играть в теннис). Она состоит из различных случаев погоды и того, комфортно ли в такую погоду играть в теннис.

На этой таблице удерживается CFD φ_1 вида $\{(Outlook, \_), (Temperature, \_), (Play, true)\} \rightarrow (Windy, \_)$. Строки 2-4, 6, 8-12 поддерживаются φ_1, так как атрибут Play в них равен True. Уверенность этой CFD равна 1, так как FD $\{(Outlook, \_), (Temperature, \_)\} \rightarrow (Windy, \_)$ удерживается на всём множестве поддерживаемых строк.

Смысл φ_1 следующий: Если играть в теннис комфортно, то по облачности и температуре можно наверняка сказать о ветренности.

Пример 2.2. На этой же таблице рассмотрим условную функциональную зависимость φ_2 $\{(Humidity, \_), (Windy, false)\} \to (Play, \_)$. Для неё $supp(\varphi_2, D) = 8$. Строки, поддерживаемые φ_2: 1, 3-5, 8-10, 13, в них аттрибут Windy равен false. FD $(Humidity, \_) \to (Play, \_)$ на них не удерживается, и $I(\varphi_2, D) = 2$. Убрав строки 2 и 3 FD начнёт удерживаться. Тогда уверенность φ_2 равна $\frac{8-2}{8} = 0.75$

Смысл φ_2 таков: Если не ветрено, то комфортность игры зависит от влажности с уверенностью 0.75. В одном из четырёх случаев, когда не ветрено, наше предположение о том, комфортно играть или нет, окажется ложным.

3. Реализация

3.1. Описание реализации интерфейса

Чтобы воспользоваться профайлером Desbordante, его можно установить при помощи менеджера пакетов pip как обычный пакет и далее импортировать в Python коде, после чего использовать как библиотеку. Для поиска CFD отсутсвуют Python-привязки, поэтому через Python этим функционалом воспользоваться нельзя.

Первым шагом к реализации Python-привязок стало добавление файлов `bind_cfd.cpp` и `bind_cfd.h`, которые описывают, какие методы поиска CFD будут доступны через Python и как они будут называться, а также добавляют Python интерфейсы двум классам, которые являются представлениями результата поиска: `RawItem` и `RawCFD`. Для привязки классов примитивов в Desbordante предусмотрены шаблонные функции.

```
1  . . .
2 py::class_<RawCFD::RawItem>(cfd_module, "Item")
3     .def_property_readonly("attribute", &RawCFD::RawItem::GetAttribute)
4     .def_property_readonly("value", &RawCFD::RawItem::GetValue);
5
6 py::class_<RawCFD>(cfd_module, "CFD")
7     .def("__str__", &RawCFD::ToString)
8     .def_property_readonly("lhs_items", &RawCFD::GetLhs)
9     .def_property_readonly("rhs_item", &RawCFD::GetRhs);
10
11 BindPrimitive<FDFirstAlgorithm>(cfd_module, &CFDDiscovery::GetCfds,
12                                  "CfdAlgorithm", "get_cfds", {"FDFirst"
       });
13  . . .
```

Листинг 1: Ключевой фрагмент bind_cfd.cpp

Пока что в Desbordante есть только один алгоритм поиска CFD, это FD First [3]. `RawItem` это пара вида номер атрибута-значение, при этом

значение атрибута является опциональным.

```
struct RawItem {
        AttributeIndex attribute;          /* attribute column index */
        std::optional<std::string> value; /* pattern value is optional */
    };
```

Листинг 2: Представление Item в C++

Класс `RawCFD` является представлением конкретной условной функциональной зависимости.

Вторым шагом реализации привязок стала вставка шаблонной функции привязки примитива в общий список:

```
for (auto bind_func : {BindMainClasses, BindDataTypes, BindFd, BindCfd,
        BindAr, BindUcc, BindAc, BindOd, BindFdVerification,
        BindMfdVerification, BindUccVerification, BindStatistics, BindInd,
        BindGfdVerification})
{
    bind_func(module);
}
```

Листинг 3: Место вызова BindCfd() в C++

Также была добавлена недостающая функция `RawCFD.ToString()`.

Помимо привязок Desbordante имеет CLI интерфейс. Чтобы сделать поиск CFD доступным через него, в файле `cli.py` был добавлен случай для поиска CFD, а также добавлено описание алгоритма FD First и поиска CFD, которое выводится при вызове команды help пользователем.

```
    case algo_name if algo_name in TASK_INFO[Task.gfd_verification].algos:
        result = algo.get_gfds()
    case Algorithm.fd_first:
        result = algo.get_cfds()
    case _:
```

```
6         assert False, 'No matching get_result function.'
7 return result
```

Листинг 4: место вызова поиска CFD в cli.py

3.2. Особенности реализации примера

Desbordante предлагает пользователям ряд примеров использования для наглядного пояснения того, как пользоваться этим инструментом и какие типы задач можно решать при помощи профайлинга данных.

В качестве примера для поиска CFD было решено представить визуализацию различных условных функциональных зависимостей на таблице. Для каждой найденной CFD должна быть выведена эта таблица с выделенными красным и зелёным цветом кортежами. Допустим была найдена CFD φ вида $(X \to Y, t_p)$. Если кортеж выделен любым цветом, это значит, что он поддерживается этой CFD. Красным цветом указываются кортежи, которые были убраны для удержания $X \to Y$ из φ. Зелёным цветом выделены оставшиеся кортежи, удовлетворяющие $X \to Y$.

```
1 def visualize_cfd(cfd, table):
2     print('CFD:')
3     print(cfd, ":\n")
4 
5     rows_satisfying_cfd, supported_rows = validate_cfd(cfd.lhs_items, cfd.
    rhs_item, table)
6 
7     header, *rows = table.to_string().splitlines()
8     print(DEFAULT_BG_CODE, header)
9     for index, row in enumerate(rows):
10        if index not in supported_rows:
11            color_code = DEFAULT_BG_CODE
12        elif index in rows_satisfying_cfd:
13            color_code = GREEN_BG_CODE
14        else:
15            color_code = RED_BG_CODE
```

```
16          print(color_code, row, DEFAULT_BG_CODE)
17      . . .
```

Листинг 5: функция визуализации CFD в mining_cfd.py

Функция validate_cfd, необходимая для visualize_cfd, могла бы быть реализована с помощью Desbordante, поскольку в нём уже имеется алгоритм валидации CFD, однако для него ещё не реализован Python интерфейс. Его реализация не входит в эту работу, поэтому было принято решение реализовать наивный валидатор CFD в рамках файла-примера mining_cfd.py.

3.3. Реализация валидатора CFD

Валидация некоторой CFD φ вида ($X \to Y$, t_p) происходит по следующему алгоритму:

- Составление $supp(t_p, D)$, то есть маски кортежей, поддерживаемых φ;
- Составление словаря “LHS в кортежи”;
- Определение самого распространённого RHS для каждой LHS;
- Пометка кортежей как удовлетворяющих FD, если они содежат самую частую RHS для своего LHS;
- Расчёт поддержки, уверенности и количества различных LHS валидируемой CFD.

После составления маски поддерживаемых кортежей, рассматриваются кортежи из неё. Оставляя только кортежи, содержащие самую частую RHS для их LHS мы не только обеспечиваем валидность $X \to Y$ на оставшемся множестве кортежей, но и делаем это, отбрасывая наименьшее возможное их количество $I(\varphi, D)$, что максимизирует рассчитанную уверенность CFD и гарантирует соответствие определению.

Для считывания таблицы из файла и вспомогательных типов данных для кортежей и масок была использована библиотека Pandas[2].

```
def make_lhs_to_row_nums(lhs, supported_df):
    lhs_to_row_nums = defaultdict(list)
    lhs_column_indices = items_to_indices(lhs)
    for row_num, row_series in supported_df.iterrows():
        key = tuple(row_series.iloc[lhs_column_indices])
        lhs_to_row_nums[key].append(row_num)
    return lhs_to_row_nums
```

Листинг 6: составление словаря LHS в кортежи

В словаре `lhs_to_row_nums` ключом является массивом атрибутов из LHS валидируемой CFD и их значений, а значением является массивом индексов кортежей с такими значениями атрибутов. После составления словаря в нём окажутся все встречающиеся вариации значений атрибутов из LHS.

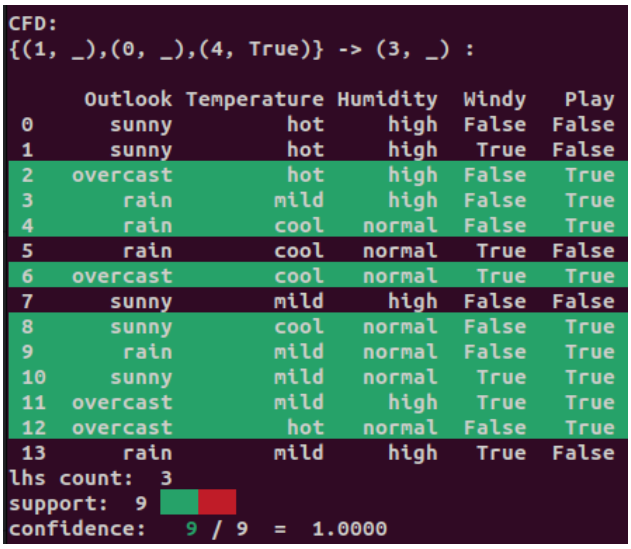
```
    for lhs_tuple, row_nums_with_lhs in lhs_to_row_nums.items():
        # fill rhs_to_row_nums
        rhs_to_row_nums = defaultdict(list)
        for row_num in row_nums_with_lhs:
            key = table.iloc[row_num, rhs.attribute]
            rhs_to_row_nums[key].append(row_num)
        row_nums_with_most_frequent_rhs = max(rhs_to_row_nums.values(), key=
    len)
        rows_satisfying_cfd.extend(row_nums_with_most_frequent_rhs)

    return rows_satisfying_cfd, supported_df.index
```

Листинг 7: Нахождение кортежей с самой частой RHS для своего LHS.

[2] https://pandas.pydata.org/pandas-docs/stable/index.html (дата доступа: 1 мая 2024 г.)

4. Апробация



(a) Визуализация примера 2.1

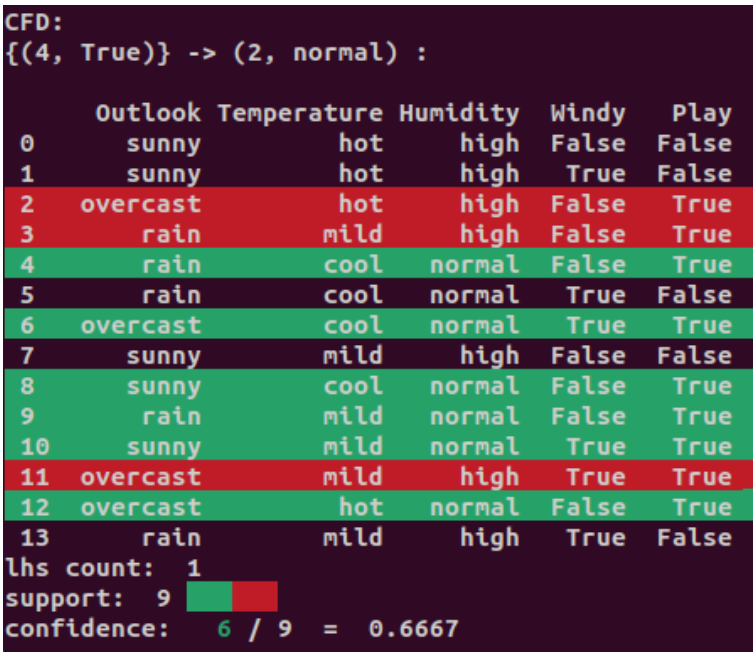
(b) Визуализация примера 2.2

При запуске файла с помощью интерпретатора Python из корневой директории Desbordante в консоль выводится визуализация нескольких первых найденных CFD. Вместо названий атрибутов выводятся их индексы, начиная с нуля.

```
1 MINIMUM_SUPPORT = 8
2 MINIMUM_CONFIDENCE = 0.7
3 MAXIMUM_LHS_COUNT = 3
```

Листинг 8: Параметры майнинга по умолчанию

Визуализация работает корректно, даже в более сложных случаях.



(a) Константная CFD

(b) CFD с confidence < 0.5

Консоль также корректно работает и выводит результаты работы алгоритма поиска условных функциональных зависимостей:

```
$ python3 ./cli/cli.py --task=cfd --table=./examples/datasets/play_tennis.csv , True --
cfd_max_lhs=4 --cfd_minconf=0.8 --cfd_minsup=1
{(2, normal)} -> (4, TRUE)
{(4, FALSE)} -> (2, high)
{(3, _),(2, normal)} -> (4, _)
{(3, FALSE),(2, normal)} -> (4, TRUE)
{(4, FALSE),(2, normal)} -> (3, TRUE)
{(3, _),(4, FALSE)} -> (2, _)
```

Рис. 3: Часть вывода интерфейса командной строки

Заключение

В ходе работы к существующему функционалу поиска CFD в Desbordante были добавлены Python-привязки, CLI интерфейс и был добавлен пример использования.

Результаты:

1. Реализованы Python-привяки для алгоритмов поиска CFD;
2. Добавлен интерфейс командной строки для алгоритмов поиска CFD;
3. Реализованы наглядные примеры использования поиска CFD с визуализацией;
4. Проведён эксперимент по тестированию корректности работы привязок, консоли и примера, ручная проверка общих и краевых случаев подтвердила правильность работы.

Реализация вошла в проект Desbordante. Pull request #349 доступен на GitHub[3].

[3] https://github.com/Desbordante/desbordante-core/pull/349 (дата доступа: 1 мая 2024 г.).

Список литературы

[1] Conditional Functional Dependencies for Data Cleaning / Philip Bohannon, Wenfei Fan, Floris Geerts et al. // 2007 IEEE 23rd International Conference on Data Engineering. — 2007. — P. 746–755.

[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] Discovering Conditional Functional Dependencies / Wenfei Fan, Floris Geerts, Laks V. S. Lakshmanan, Ming Xiong // 2009 IEEE 25th International Conference on Data Engineering. — 2009. — P. 1231–1234.