Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б15-мм

Верификация вероятностных функциональных зависимостей в Desbordante

Немакин Никита Антонович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ассистент кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2024

Оглавление

Введение

Профилирование данных [3] — процесс получения метаданных для конкретного набора данных. Оно широко применяется для контроля качества данных в машинном обучении, при разработке информационных систем и баз данных, в бизнес-аналитике. С помощью профилирования можно обнаружить дубликаты, выявить аномальные данные, проверить соответствие данных ожидаемым стандартам.

Одной из ключевых характеристик данных является функциональная зависимость. Пусть R — отношение, X, Y — наборы его атрибутов. Говорят, что отношение удовлетворяет функциональной зависимости $X \to Y \iff \forall t_1, t_2 \in R \;\; t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y]$. Помимо точных выделяют также вероятностные (Probabilistic Functional Dependencies [2], PFD) и приближенные (Approximate Functional Dependencies [4], AFD) функциональные зависимости, которые могут выполняться на наборе данных с некоторой погрешностью.

Desbordante[1] — это высокопроизводительный инструмент профилирования данных с открытым исходным кодом, написанный на C++. Он содержит алгоритмы, способные выявлять и валидировать различные закономерности в табличных данных, в том числе вероятностные функциональные зависимости. На данный момент в проекте имеется поддержка поиска PFD с помощью алгоритма PFDTane [5], однако возможность валидации заданной пользователем зависимости отсутствует. Данная работа направлена на добавление в проект подобной функциональности.

[1] https://github.com/Desbordante/desbordante-core

1. Постановка задачи

Целью работы является расширение функциональности Desbordante, а именно реализация алгоритма валидации вероятностных функциональных зависимостей. Для её достижения были поставлены следующие задачи:

1. провести обзор предметной области;
2. реализовать функциональность подсчёта ошибки для заданной функциональной зависимости;
3. реализовать привязки для Python к алгоритму валидации и добавить возможность его использования в консольной утилите;
4. протестировать реализованный алгоритм на корректность.

2. Обзор предметной области

2.1. Основные определения

Все далее приведенные определения взяты из работы [1].

Определение 1. Пусть R — отношение, $X \subset R$ — набор атрибутов из R. Обозначим за $D_X = \{t[X] \mid t \in R\}$ множество уникальных значений атрибутов из X.

Определение 2. Пусть X_1 — заданные значения атрибутов X. Рассмотрим $V_{X_1} = \{t \in R \mid t[X] = X_1\}$ — множество кортежей с данными значениями X_1 атрибутов из X. Тогда обозначим за

$$(V_Y, V_{X_1}) = \{t \in R \mid t[X] = X_1 \wedge t[Y] = \underset{Y_k \in R}{\operatorname{argmax}}\{|V_{X_1} \cap V_{Y_k}|\}\}$$

кортежи с наиболее популярным значением Y для всех кортежей с одинаковыми значениями атрибутов из X.

Определение 3. Вероятностной функциональной зависимостью (далее PFD) будем называть функциональную зависимость $F : X \xrightarrow{p} Y$, где p — вероятность удержания для значений X_1 атрибута X, которая определяется как $P(X \to Y, V_{X_1}) = \frac{|V_Y, V_{X_1}|}{|V_{X_1}|}$.

Вероятность PFD между атрибутами X и Y определяется так называемыми метриками PerValue и PerTuple:

$$P_{PerValue}(X \to Y, R) = \frac{\sum_{V_X \in D_X} P(X \to Y, V_X)}{|D_X|},$$

$$P_{PerTuple}(X \to Y, R) = \sum_{V_X \in D_X} \frac{|V_X|}{|R|} P(X \to Y, V_X).$$

Эти метрики определяют вероятность удержания PFD на всем отношении R, учитывая различные значения атрибутов X. Стоит отметить, что первая является средним арифметическим вероятностей для всемозможных значений атрибутов X, в то время как вторая учитывает количество кортежей в каждом из наборов различных значений атрибутов X.

Определение 4. Значением ошибки PFD будем называть выражение $\epsilon = 1 - P(X \to Y)$, где P — одна из метрик PerValue или PerTuple.

2.2. Примеры PFD

Таблица 1

X	Y
1	2
1	3
1	2
2	3
2	3
2	3
3	4
3	5
3	4
4	5
5	6

Таблица 2

X	Y
0	1
0	2
0	3
0	4
0	5
...	...
1	1
2	2
3	3
4	4
5	10

Пример 1. Посчитаем вероятность удержания PFD X $\to$ Y в таблице 1 для обоих метрик. Здесь $P_{PerValue}(X \to Y, R) = \frac{\frac{2}{3}+1+\frac{2}{3}+1+1}{5} = \frac{13}{15}$, $P_{PerTuple}(X \to Y, R) = \frac{2+3+2+1+1}{11} = \frac{9}{11}$. Значения ошибок ϵ приблизительно равны 0.13 и 0.18 соответственно.

Пример 2. Рассмотрим более интересный пример [5] зависимости X $\to$ Y на наборе данных R из таблицы 2, который демонстрирует особенности приведенных выше метрик. Метрика PerValue не зависит от частоты X. Действительно, пусть $|V_0| \to \infty$. В этом случае $P_{PerValue}(X \to Y, R)$ стремится к $\frac{6}{7}$, а соответствующая ошибка стремится к $\frac{1}{7}$. В то же время $P_{PerTuple}(X \to Y, R)$ стремится к 0, а значение ошибки приближается к 1. Таким образом, метрика PerValue позволяет учитывать «ошибочные» значения для X в небольшом количестве. Например, такая «локальная» ошибка может возникнуть, если один датчик из общего исправного набора датчиков начинает передавать ошибочные данные.

3. Реализация

3.1. Валидация PFD

Валидация заключается в проверке, удерживается ли принимаемая на вход функциональная зависимость на наборе данных, и если нет (не выполняется точная ФЗ), то с какой вероятностью для данной метрики. При этом собирается некоторая полезная информация, например, для всех наборов кортежей с одинаковыми атрибутами в левой части можно сообщать количество записей с различными значениями атрибутов в правой части зависимости (нарушающие записи).

Процедура валидации основана на леммах из статьи [4] 2.1 и 2.2, в которых утверждается, что функциональная зависимость $X \to A$ удерживается тогда и только тогда, когда множество кортежей с одинаковой левой частью полностью совпадает с множеством кортежей с одинаковыми левой и правой частью. Поэтому для подсчёта ошибки для заданной метрики достаточно подсчитать эти множества для всех различных значений левой и правой частей ФЗ.

В валидаторе PFD для подсчёта таких множеств применяется структура данных, изначально введенная в TANE, как PositionListIndex (PLI). Она состоит из набор классов эквивалентности (кластеров), построенных относительно равенства значений атрибутов. В процессе валидации также считаются «ошибочные» кластера, в которых есть ненулевое число записей с различной правой частью.

Алгоритм валидации можно разбить на следующие несколько этапов:

- выделение кластеров с одинаковыми значениями атрибутов X, Y, и их последующее пересечение, при этом кластера, состоящие из одиночных записей не учитываются в целях экономии памяти;
- сортировка кластеров из пересечения по их взаимному расположению в кластерах для атрибутов X (это необходимо для удобного подсчёта вероятностей);
- для каждого кластера с атрибутами X подсчитывается макси-

мальный размер кластера пересечения (определение 2), здесь же происходит подсчёт ошибочных кластеров и строк;

- подсчёт итоговой ошибки для заданной изначально метрики с учётом одиночных записей;

Иерархия классов представлена на рис. 1. Все алгоритмы поиска или валидации в Desbordante наследуются от базового класса `Algorithm`, который содержит в себе интерфейс загрузки исходных данных, активацию необходимых для работы алгоритма опций и непосредственно выполнение.

Класс `PFDVerifier` предоставляет интерфейс для получения полезной информации, такой как значение ошибки (метод `GetError`), ошибочные кластера (метод `GetViolatingClusters`). Для подсчёта же необходимой статистики он агрегирует в себе экземпляр класса `PFDStatsCalculator`, который используется в методе `VerifyPFD` путём вызова у него метода `CalculateStatistics`, принимающего на вход кластера, описанные выше. После каждого выполнения алгоритма найденные значение ошибки и ошибочные кластера сбрасываются в методе `ResetState`, чтобы алгоритм можно было запустить повторно с новыми параметрами.

Сценарий использования алгоритма валидации может быть описан следующим образом: пользователь указывает в качестве входных данных путь до табличных данных в формате CSV, задает набор индексов столбцов для левой и правой частей функциональной зависимости, которую собирается исследовать, а также указывает метрику для подсчёта вероятности удержания зависимости и порог ошибки (число от 0 до 1 включительно), превышение которого будет означать, что функциональная зависимость не удерживается. В случае, если значение ошибки превышает заданный порог, пользователю доступна информация об "ошибочных" кластерах, содержащих нарушающие записи, которые повлияли на подсчет итоговой ошибки.

Для удобства использования алгоритма были разработаны привязки для языка Python, а также была реализована возможность запуска

алгоритма в консольном приложении наряду с другими алгоритмами Desbordante.

3.2. Привязки для Python

Desbordante предлагает пользователю возможность использования в Python алгоритмов, реализованных на C++, в виде библиотеки или консольного приложения. Поэтому для полноценной интеграции нового компонента в проект были реализованы привязки с использованием заголовочной библиотеки pybind11, позволяющей определять свои модули, классы и методы, которые связываются с соответствующей реализацией на C++.

Так, в методе `BindPfdVerification` с помощью шаблонных функций из проекта создается модуль `pfd_verification`, предоставляющий API для взаимодействия с класом `PFDVerifier`.

3.3. Консоль Python

Консольный интерфейс реализован в отдельном репозитории[2] проекта в файле `cli.py`. Утилита имеет несколько опций, необходимых для запуска конкретного алгоритма. Опция `help` выводит информацию об утилите. Опции `task` и `algo` позволяют указать задачу или конкретный алгоритм, который должен быть запущен. Остальные опции специфичны для каждого алгоритма.

Для валидатора PFD опция `table` позволяет указать путь к табличным данным, опции `lhs_indices` и `rhs_indices` — индексы атрибутов для левой и правой частей функциональной зависимости соответственно. Опция `error_mesure` отвечает за метрику, используемую для подсчёта ошибки.

Пример вызова алгоритма валидации через консоль:

```
$> python3 cli.py --task=pfd_verification
--table=examples/datasets/pfd.csv , True
```

[2] https://github.com/Desbordante/desbordante-cli

```
--lhs_indices=0 --rhs_indices=1 --error_measure=per_tuple

Exact functional dependency does not hold, but instead
probabilistic functional dependency holds with error = 0.5
```

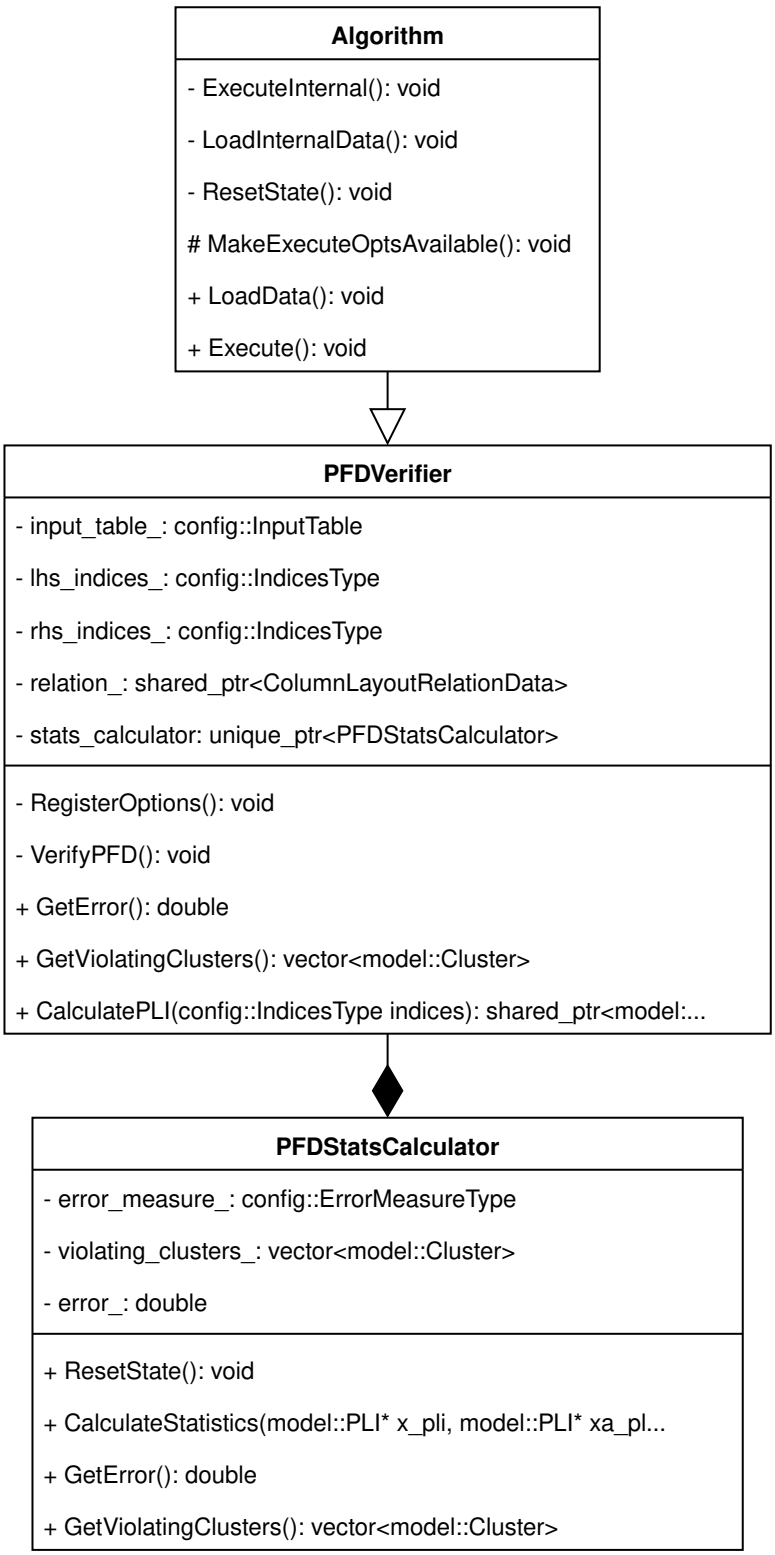

Рис. 1: Иерархия классов

4. Тестирование

Для проверки корректности работы алгоритма валидации были написаны автоматические тесты с использованием фреймворка googletest[3]. Тестирование проводилось на нескольких наборах данных. Значение подсчитанной вручную ошибки для обоих метрик PerValue и PerTuple сравнивалось с найденным в результате работы алгоритма. Были также учтены случаи с различными наборами индексов атрибутов в левой и правой частях функциональных зависимостей.

С целью проверки работоспособности интерфейса валидации в Python был написан пример использования библиотеки, приведенный в Листинге 1. В методе `load_data` загружаются табличные данные, дополнительно указывается используемый разделитель и наличие заголовка. Алгоритм с помощью метода `execute` повторно запускается на одних и тех же данных, но с разными метриками подсчёта ошибки. В функции `print_results` подсчитанная ошибка сравнивается с эталонной и делается вывод, удерживается ли PFD или нет. В последнем случае выводятся «ошибочные» кластера, в которых есть строки с различными значениями в правой части.

[3] https://github.com/google/googletest

Листинг 1: Пример использования в Python

```
import desbordante

ERROR = 0.2
PER_TUPLE = 'per_tuple'
PER_VALUE = 'per_value'
TABLE = 'examples/datasets/glitchy_sensor_2.csv'

def print_results(pfd_verifier):
    error = pfd_verifier.get_error()
    if error <= ERROR:
        print('PFD holds')
    else:
        print(f'PFD with error {ERROR} does not hold')
        print(f'instead it holds with error {error}')
        print(f'Clusters violating PFD:')
        for cluster in pfd_verifier.get_violating_clusters():
            print(cluster)
    print()

algo = desbordante.pfd_verification.algorithms.PFDVerifier()
algo.load_data(table=(TABLE, ',', True))

algo.execute(lhs_indices=[1], rhs_indices=[2],
             error=ERROR, error_measure=PER_VALUE)
print_results(algo)

algo.execute(lhs_indices=[1], rhs_indices=[2],
             error=ERROR, error_measure=PER_TUPLE)
print_results(algo)
```

Заключение

В ходе данной работы в проект была добавлена поддержка верификации вероятностных функциональных зависимостей для метрик PerValue и PerTuple вместе с Python-привязками и консольным интерфейсом. Результаты работы:

1. проведён обзор предметной области;
2. реализован алгоритм подсчёта ошибки удержания вероятностной функциональной зависимости для обоих метрик[4];
3. реализованы привязки для Python к алгоритму и поддержано его использование в консольной утилите[5];
4. реализованный алгоритм протестирован на корректность.

Код работы доступен на GitHub.

[4] https://github.com/Desbordante/desbordante-core/pull/463
[5] https://github.com/Desbordante/desbordante-cli/pull/4

Список литературы

[1] Extending Desbordante with Probabilistic Functional Dependency Discovery Support / Ilia Barutkin, Maxim Fofanov, Sergey Belokonny et al. // 2024 35th Conference of Open Innovations Association (FRUCT). — 2024. — P. 158–169.

[2] Functional Dependency Generation and Applications in Pay-As-You-Go Data Integration Systems. / Daisy Wang, Xin Dong, Anish Das Sarma et al. — 2009. — 01. — URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-119.pdf.

[3] Ilyas Ihab F., Chu Xu. Data Cleaning. — New York, NY, USA : Association for Computing Machinery, 2019. — ISBN: 9781450371520.

[4] Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies / Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, Hannu Toivonen. — Vol. 42. — 1999. — P. 100–111.

[5] Баруткин И. Д. Реализация алгоритма проверки вероятностных функциональных зависимостей в профилировщике данных Desbordante. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/PFD%20-%20Ilia%20Barutkin%20-%20BA%20thesis.pdf.