Санкт-Петербургский государственный университет

Группа 22.Б09-мм

Реализация метрик схожести в “Desbordante”

Николаев Даниил Владимирович

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 **4**

2. Обзор **5**

2.1. Основные понятия 5

2.2. Метрики сходства 5

2.3. Метрики расстояния 6

3. Реализация **8**

3.1. Связь с поиском MD 8

3.2. Интеграция метрик 9

4. Эксперименты **11**

Заключение **12**

Список литературы **13**

Введение

Профилирование данных представляет собой метод анализа информации, который позволяет извлечь метаданные. Цели их извлечения могут быть самыми разными: изученный набор данных можно преобразовать полезным образом, проведя дедупликацию, или же обнаружить аномалии, пропущенные значения и закономерности, скрытые в данных. Также можно выявить значимую информацию из данных.

Одним из ключевых аспектов этого анализа, как уже было упомянуто, является поиск закономерностей в данных. Для табличных данных такими закономерностями выступают функциональные зависимости (Functional Dependencies, FD), которые описывают отношения между атрибутами. Функциональная зависимость вида $X \to Y$ означает, что если две записи в таблице равны по атрибутам X, то они должны быть равны по атрибутам Y.

Однако в реальных наборах данных нередко встречаются ошибки, пропуски или вариации, которые делают традиционные функциональные зависимости недостаточно гибкими. Для работы с такими неточными данными используются ослабленные функциональные зависимости (Relaxed Functional Dependencies, RFD) [2], которые допускают определенные отклонения от строгих правил. Одним из видов таких зависимостей являются сопоставляющие зависимости (Matching Dependencies, MD). В отличие от FD, где требуется точное совпадение значений атрибутов, MD позволяют учитывать схожесть данных — если значения атрибутов достаточно похожи, это может считаться соблюдением зависимости.

Для реализации сопоставляющих зависимостей важно уметь количественно оценивать степень схожести между значениями атрибутов. Именно здесь на первый план выходят метрики сходства. Эти метрики позволяют измерять, насколько близки или различны данные значения. В рамках данной работы предполагается реализация метрик в платформе `Desbordante`, что позволит эффективно находить и применять ослабленные зависимости в анализе данных.

1. Постановка задачи

Целью работы является реализация и интеграция метрик сходства в платформу `Desbordante`.

Для её достижения были поставлены следующие **задачи**:

- Обзор и анализ существующих метрик сходства.
- Разработка алгоритмов для вычисления метрик сходства.
- Тестирование и оценка производительности.
- Оптимизация алгоритмов вычисления метрик с учетом специфики алгоритма поиска MD.

2. Обзор

2.1. Основные понятия

Рассматриваемые метрики разделяются на два типа: схожести и расстояния. Метрика схожести (similarity measure) — это функция, которая измеряет степень похожести между объектами и возвращает вещественное число в диапазоне от 0 до 1, где 0 означает полное несходство, а 1 — полное сходство. Расстояние же может принимать любые значения.

2.2. Метрики сходства

- **Индекс Жаккара**

  Индекс Жаккара [5] определяет схожесть между двумя множествами как отношение размера их пересечения к размеру их объединения:

  $$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

  Также его можно выразить следующей формулой:

  $$J(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

- **Метрика Монжа-Элкана**

  Обобщенная метрика Монжа-Элкана [1] работает с двумя наборами объектов, принцип её работы следующий:

  – Для каждого элемента первого набора находится наиболее похожий элемент во втором наборе с помощью некоторой заданной функции схожести.

  – Полученные максимальные значения схожести усредняются.

  – Тот же процесс повторяется для второго набора по отношению к первому.

  – Итоговое значение схожести является средним геометрическим между двумя полученными средними.

  Формально, если A и B — это два набора строк, то схожесть по Монжу-Элкану между ними определяется как:

  $$ME(A, B) = \sqrt{\frac{1}{|A|} \sum_{a \in A} \max_{b \in B} \mathrm{sim}(a, b) \cdot \frac{1}{|B|} \sum_{b \in B} \max_{a \in A} \mathrm{sim}(b, a)},$$

  где $\mathrm{sim}(a, b)$ — функция схожести между двумя элементами a и b.

2.3. Метрики расстояния

- **Наибольшая общая подпоследовательность**

  Как следует из названия, алгоритм [4] позволяет находить самую длинную общую подпоследовательность двух последовательностей (в данном случае — строк). Подпоследовательностью строки называется строка, полученная из исходной удалением некоторых элементов, но не обязательно подряд идущих.

- **Метрика Смита-Уотермана-Готоха (СУГ)**

  Метрика СУГ [1] — это метрика, используемая для локального сравнения двух последовательностей. Алгоритм работает следующим образом:

  – Последовательно сравниваются элементы двух последовательностей с учётом возможных разрывов (вставок/удалений), за которые начисляются штрафы (gap penalty).

  – Для каждой пары элементов вычисляется оценка их схожести, где совпадения дают положительное значение, а несовпадения и разрывы — отрицательное.

  – На основе этих оценок строится матрица, содержащая максимальные локальные выравнивания двух последовательностей.

- **Расстояния между числами и датами**

$$\text{DateDifference}(D_1, D_2) = |D_1 - D_2|,$$

  где $|D_1 - D_2|$ — это количество дней между датами D_1 и D_2.

  Пример:

  Пусть D_1 — 1 октября 2023 года, а D_2 — 15 сентября 2024 года.

$$\text{DateDifference}(D_1, D_2) = 379 \text{ дней}$$

  Аналогично с датами, расстояние между числами:

$$\text{NumberDifference}(N_1, N_2) = |N_1 - N_2|.$$

- **Расстояние Левенштейна**

  Расстояние Левенштейна [3] $d_{\text{lev}}(A, B)$ между двумя строками A и B определяется как минимальное количество операций вставки, удаления или замены символов, необходимых для преобразования строки A в строку B. Для двух строк

длины m и n, расстояние Левенштейна вычисляется рекурсивно по следующей формуле:

$$d_{\text{lev}}(i,j) = \begin{cases} \max(i,j) & \text{если } \min(i,j) = 0, \\ \min \begin{cases} d_{\text{lev}}(i-1,j) + 1, \\ d_{\text{lev}}(i,j-1) + 1, \\ d_{\text{lev}}(i-1,j-1) + \mathbb{1}_{(A_i \neq B_j)} \end{cases} & \text{в остальных случаях,} \end{cases}$$

где $\mathbb{1}_{(A_i \neq B_j)}$ равно 1, если символы A_i и B_j различны, и 0, если они совпадают.

В данной работе не описывается реализация алгоритма Левенштейна, так как она уже была реализована ранее в рамках проекта.

3. Реализация

3.1. Связь с поиском MD

Как уже было сказано, метрики находят применение в поиске MD.

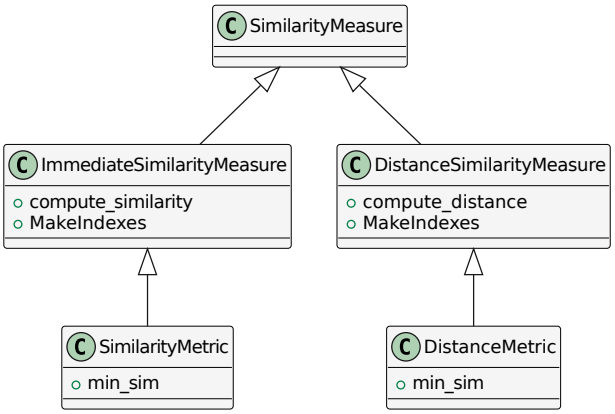

Рис. 1: Структура классов метрик

Как видно из рисунка 1 классы метрик разделяются на два типа: сходства и расстояния. Класс сходств (`ImmediateSimilarityMeasure`) и класс расстояний (`DistanceSimilarityMeasure`) принимают на вход функцию-метрику, а также содержат реализацию метода `MakeIndexes`, который на вход получает два столбца из сопоставления и возвращает структуру, хранящую информацию об их сопоставлении. Подробнее об индексах можно найти в работе, посвященной поиску сопоставляющих зависимостей [8].

В контексте данной работы все строковые метрики будем относить к типу метрик схожести, поскольку можно подобрать соответствующие им функции, возвращающие результат из диапазона [0,1] с незначительными дополнительными вычислениями. Пример такого подхода для метрики `Longest Common Subsequence`:

Listing 1: Пример кода на C++

```
LcsSimilarityMeasure(model::md::DecisionBoundary min_sim, ...)
    : ImmediateSimilarityMeasure(...,
    [min_sim](string const& left, string const& right) {
        size_t dist = LongestCommonSubsequence(left, right);
        size_t max_dist = max(left.size(), right.size());
        double sim = (max_dist - dist) / max_dist;
        return sim;
    }, ...) {}
```

Значения пары считаются схожими, если результат применения метрики схожести к ним больше или равен некоторому пределу, который называется границей решения.

Именно за это в данном случае отвечает `min_sim`.

3.2. Интеграция метрик

- **Индекс Жаккара**

  Реализована обобщенная версия данной метрики, параметрами которой является пара множеств (`std::set`), а также реализована перегрузка для строк с помощью токенизации по пробелам.

- **Наибольшая общая подпоследовтельность**

  Алгоритм основан на динамическом программировании и использует двумерный массив `lcs_table` для хранения промежуточных результатов. В итоге в ячейке `lcs_table[m][n]` будет храниться длина наибольшей общей подпоследовательности для строк `word1` и `word2`, где `m` и `n` — длины этих строк.

- **Метрика Монжа-Элькана**

  Реализована обобщенная версия метрики для пары векторов строк. В данной версии алгоритм помимо двух векторов строк принимает функцию, которая определяет сходство между двумя строками. Затем он перебирает все строки из первого вектора и для каждой из них находит наиболее похожую строку во втором векторе с помощью предоставленной функции. После этого алгоритм суммирует значения сходства для всех пар строк и делит на количество строк в первом векторе, чтобы получить среднее значение сходства. Пусть это значение равняется `S`, тогда для того, чтобы метрика стала симметричной, результат необходимо вычислять следующим образом:

  $$\text{result} = \sqrt{S(a, b) \cdot S(b, a)}.$$

  Для строк реализована версия с токенизацией по пробелам и использованием по умолчанию метрики Смита-Уотермана-Готоха (нормализованной).

- **Метрика Смита-Уотермана-Готоха**

  Метрика СУГ предназначена для выполнения локального выравнивания двух строк с учётом штрафов за разрывы. Разрыв (gap) возникает, когда один из символов в одной строке пропускается для достижения лучшего соответствия с символами другой строки. Штраф за разрыв задаётся параметром `gapValue`, по умолчанию равным -0.5.

  Основная идея алгоритма заключается в вычислении наилучшего локального выравнивания символов строк `s` и `t`. Для каждой пары символов из строк вычисляются значения, отражающие:

– Совпадение символов (при использовании функции `SubstitutionCompare`, которая определяет степень совпадения).

– Штраф за пропуск символа из одной строки (разрыв).

– Нулевое значение, если выравнивание нецелесообразно.

Алгоритм использует матрицу для отслеживания наилучших значений выравнивания на каждом шаге и обновляет максимальное значение, найденное на текущем этапе. Итоговым результатом является максимальное значение, отражающее лучшее локальное выравнивание строк с учётом штрафов за разрывы.

Нормализованная версия метрики Смита-Уотермана-Готоха вычисляется следующим образом:

$$\text{Normalized\_SWG} = \frac{\text{SmithWatermanGotoh}(s, t, \text{gapValue})}{\min(s_{\text{size}}, t_{\text{size}}) \times \max(1.0, \text{gapValue})}.$$

4. Эксперименты

Для проверки правильности интеграции метрик, были реализованы тесты[1], доступные в файле `test_hymd_metrics.cpp`, при различных входных данных (по 10 сценариев на метрику в среднем). В качестве ожидаемых значений были взяты результаты метрик из библиотеки `simmetrics` [7], поскольку они используются в `Metanome` [6].

Также было произведено тестирование производительности метрик на наборе данных `iowa1kk.csv` (1 миллион строк). Результаты тестирования отражены в Таблице 1.

Характеристики системы, на котором проводилось тестирование — `AMD ® Ryzen 3 3200u, 8 GiB RAM, Ubuntu 23.04, gcc 13.2.0`.

Таблица 1: Время выполнения метрик для Desbordante и Simmetrics

	JaccardIndex	LCS	SWG	Monge-Elkan
Desbordante	0.585 ± 0.016 s	0.245 ± 0.009 s	0.282 ± 0.010 s	1.201 ± 0.019 s
Simmetrics	0.756 ± 0.048 s	0.367 ± 0.025 s	0.820 ± 0.057 s	1.764 ± 0.095 s

Каждый набор данных был обработан по 30 раз. В результате экспериментов по полученным данным были построены 95% доверительные интервалы для среднего. Время необходимое для загрузки данных в систему не учитывалось, были отключены все фоновые процессы, была установлена максимальная частота процессора, а также был совершен прогрев из 10 проходов. Программа была исполнена одним ядром.

Как видно из Таблицы 1, среднее время выполнения метрик, реализованных в данной работе, приблизительно в 1.5 раза меньше среднего времени выполнения тех же метрик, используемых в `Simmetrics`.

[1] https://github.com/niklvdanya/Desbordante/blob/optimize-hymd2/src/tests/test_hymd_metrics.cpp

Заключение

В ходе работы были выполнены следующие задачи:

- Был проведен обзор метрик схожести.
- Метрики интегрированы в fork-репозиторий `Desbordante`, позже — в сам проект.
- Было произведено тестирование метрик.
- Была достигнута производительность алгоритма метрик схожести, в 1.5 раза превышающая производительность метрик из Simmetrics, поэтому было решено отложить оптимизацию алгоритмов вычисления метрик с учетом специфики алгоритма поиска MD на следующий семестр.

Код доступен на `Github`[2].

[2] https://github.com/BUYT-1/Desbordante/pull/8

Список литературы

[1] A. E. Monge C. Elkan. The field matching problem: algorithms and applications. // In Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD). — 1996. — P. 267–270.

[2] Caruccio Loredana, Deufemia Vincenzo, and Polese Giuseppe. Relaxed Functional Dependencies—A Survey of Approaches // IEEE Transactions on Knowledge and Data Engineering. — 2016. — Vol. 28, no. 1. — P. 147–165.

[3] Levenshtein V. I. Binary codes capable of correcting deletions, insertions, and reversals. // In Soviet Physics Doklady, volume 10. — 1966. — P. 707–710.

[4] Longest common subsequence, Wiki. — Access mode: https://en.wikipedia.org/wiki/Longest_common_subsequence_problem (online; accessed: 17-09-2024).

[5] M. Levandowsky D. Winter. Distance between sets. — 1971.

[6] Metanome=algorithms source code repository. — Access mode: https://github.com/HPI-Information-Systems/metanome-algorithms.git (online; accessed: 17-09-2024).

[7] Simmetrics source code repository. — Access mode: https://github.com/Simmetrics/simmetrics (online; accessed: 17-09-2024).

[8] Шлёнских Алексей. Обзор сопоставляющих зависимостей и алгоритма их поиска HyMD. — 2023. — Access mode: https://github.com/Mstrutov/Desbordante/blob/main/docs/papers/HyMDreview-ShlyonskikhAlexey-2023spring.pdf (online; accessed: 17-09-2024).