ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему разработка системы предсказания успешного завершения учебной дисциплины
(промежуточный, этап 1)

Выполнил:
студент группы БПМИ184 _____ Чурилов Игорь Александрович
Подпись И.О. Фамилия
10.02.2020
Дата

Принял:
руководитель проекта Андрей Андреевич Парилов
Имя, Отчество, Фамилия
младший научный сотрудник
Должность
МНУЛ ИССА ФКН НИУ ВШЭ
Место работы

Дата 10.02.2020 10 _____
Оценка (по 10-тибалльной шкале) Подпись

Москва 2020

Содержание

1 Задачи первого этапа

Изучение и сравнение алгоритмов построения ассоциативных правил. Далее рассмотренны алгоритмы PrefixSpan и FIN.

2 Построение ассоциативных правил

2.1 Постановка задачи

- Входные данные – массив последовательностей событий, параметр $min\_support$.
- Выходные данные – набор паттернов, часто встречающизся в массиве последовательностей.

2.2 Основные понятния

- $I = \{i_1, i_2 \dots i_n\}$ – множество всех возможных событий.
- Последовательность – массив событий $[t_1, t_2, \dots t_m]$, где $t \subseteq I$, $\forall t$
- Пусть $\alpha = [a_1, \dots, a_n]$, $\beta = [b_1, \dots, b_m]$, α называют подпоследовательностью β если существуют целые числа $1 \leqslant j_1 \leqslant j_2 \leqslant \dots . \leqslant j_n \leqslant m$, такие, что $a_1 \subseteq b_{j_1}, \dots a_n \subseteq b_{j_n}$
- $count(pattern)$ – количество появлений $pattern$ как подпоследовательность на множестве всех последовательностей.
- $support(pattern) = \frac{count(pattern)}{|DATABASE|}$
- $min\_support \in [0,1]$ - ограничение снизу на $support()$ для паттернов, которые мы ищем.
- Назовем $sequence$ частым, если $support(sequence) \geqslant min\_support$

3 Алгоритм PrefixSpan

3.1 Необходимые определения

- Если α подпоследовательность β, то β – это $super\ sequence$ для α.
- Пусть $\alpha =< a_1, \dots, a_n >$, $\beta =< b_1, \dots, b_m >$ $m \leqslant n$, β называют префиксом α, если $b_i = a_i$ для всех $i \leqslant m-1$ и $b_m \subseteq a_m$

- Пусть β подпоследовательность α, тогда α' (подпоследовательность α) называют проекцией β, если β это префикс α', и не существует $super\ sequence$ для α' с таким же условием.

3.2 Характеристика алгоритма

- В результате работы получается полное множество частых паттернов.
- В отличие от Apriori-like алгоритмов нет нужды генерировать и проверять последовательности, не содержащиеся в проецированной базе данных. Все более длинные паттерны получаются из уже существующих коротких.
- Проецированная база данных постоянно уменьшается: уменьшается количество последовательностей и длины префиксов.
- Более производительный чем GSP и FreeSpan.
- Можно улучшить с помощью более эффективного проецирования.

3.3 Ход Алгоритма

1. Изначально алгоритм запускается для пустого паттерна и всей базы данных.
2. Алгоритм получает на вход текущую проекцию (массив событий) и текущий паттерн.
3. Просмотр последовательности на предмет частых событий, то есть тех $event$, для которых верно: $support(current\_pattern + event) \geq min\_support$
4. Добавление к $current\_pattern$ найденных $event$. Получение новых паттернов.
5. Построение проекции массива последовательностей для каждого из полученных паттернов.
6. Для каждого из новых паттернов рекурсивно запускается алгоритм на соотвествующей ему проекции массива последовательностей.

3.4 Оптимизации

Оптимизации алгоритма основываются на более эффективном построении проекций, так называемом "псевдо-проецированнии":

Можно заметить что спроецированный массив содержится в исходном. Тогда вместо того чтобы каждый раз "физически"строить новый массив последовательностей, можно строить массив пар $\langle sequence\_number, offset \rangle$, где:

$sequence\_number$ – номер последовательности из базы данных.

$offset$ – сдвиг последовательности, по сути индекс начала события в данной проекции.

Стоит заметить, что использовать эту оптимизацию можно только тогда, когда вся база данных помещается в оперативную память, так как множественное обращение к разным участкам памяти ПЗУ сильно влияет на производительность.

4 Алгоритм FIN

4.1 Характеристика алгоритма

- Работает эффективнее чем PrePost и FR-growth.
- POC-tree в отличие от PPC-tree хранит только pre-order, делая алгоритм эффективнее по памяти.
- На плотных датасетах потребляет намного меньше памяти чем PrePost.

4.2 Необходимые определения

- $pre-order$ – номер вершины для обхода в порядке: (текущая вершина; левый ребенок; правый ребенок)
- $N-info$ – для вершины N из POC-tree это пара $(pre-order, count)$ (соответствующие вершине pre-order, и частота одноэлементного патерна).
- Если отсортировать частые элементы в каждой последовательности из базы данных в порядке убывания их частоты, получится множество $L1$.
- $i_1 \succ i_2$, если $i_1, i_2 \in L1$, и частота i_1 больше (то есть в $L1$ он стоит раньше чем i_2).
- $Nodeset(i)$ – множество всех $N-info$ для события i.
- $Nodeset(i_1, i_2)$ – если $i_1 \succ i_2$ - множество тех (pre_k, c_k) из $Nodeset(i_2)$, для которых верно, что существует вершина N, соответствующая i_1 и являющаяся предком (pre_k, c_k).
- $Nodeset$ K-элементного множества: Пусть $P = p_1 p_2 \dots p_k$ – K-элементное множество ($p_j \in L1$ и $p_1 \succ p_2 \succ \dots \succ p_k$). Положим $P_1 = p_1 p_3 \dots p_k$ и $P_2 = p_2 p_3 \dots p_k$, тогда $Nodeset(P) = Nodeset(P_1) \cap Nodeset(P_2)$.

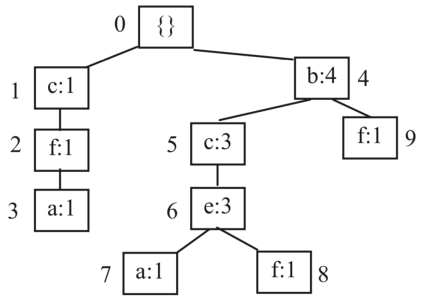
4.3 Ход Алгоритма

1. Построение POC-tree и выявления всех частых одноэлементных множеств событий. (и их Nodesets)
2. Запускается обход POC-tree для нахожденя всех частых двухэлементных множеств и их Nodesets.
3. Построение всех частых K (>2) - элементных множеств.

4.4 POC-tree

POC-tree отличается от PPC-tree только тем что для каждой вершины хранит лишь ее pre-order. Для каждой последовательности из базы данных отсортируем ее частые элементы в порядке убывания частоты. Получим упорядоченные последовательности частых элементов ($L1$), по сути множество слов. По нему построим префиксное дерево и посчитаем pre-order порядок каждого элемента. Пример построения:

ID	Items	Ordered frequent items
1	*a, c, g, f*	*c, f, a*
2	*e, a, c, b*	*b, c, e, a*
3	*e, c, b, i*	*b, c, e*
4	*b, f, h*	*b, f*
5	*b, f, e, c, d*	*b, c, e, f*



4.5 Нахождение паттернов старших размеров

Утверждается, что если $Nodeset(P) = \{(pre_1, c_1), \ldots (pre_m, c_m)\}$, где P – K-элементное множество, то $count(P) = c_1 + c_2 + \ldots + c_m$. На использовании этого свойства и строится алгоритм FIN. Построение POC-tree позволяет расчитать $Nodesets$ одноэлементных множеств. Далее по нему, пользуясь определением, можно рассчитать все $Nodesets$ двухэлементных множеств. Заметим, что зная их можно рекурсивно посчитать $Nodesets$ трехэлементных множеств. Таким образом, увеличивая размер рассматриваемых множеств можно найти все часто встречаемые паттерны.

5 Источники

PrefixSpan:

- PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth (Jian Pei, Jiawei Han, Behzad Mortazavi-asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu. (2002))

FIN:

- Fast mining frequent itemsets using Nodesets(Expert Systems with Applications 41 (2014), Zhi-Hong Deng, Sheng-Long Lv)