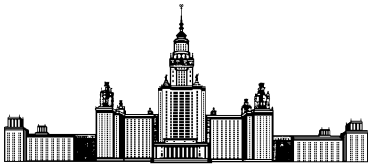
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики
Кафедра Математических Методов Прогнозирования

КУРСОВАЯ РАБОТА СТУДЕНТА 317 ГРУППЫ

«Совершенствование алгоритмов поиска частых множеств и ассоциативных правил»

Выполнил:
студент 3 курса 317 группы
Саенко Иван Александрович

Научный руководитель:
д.ф-м.н., профессор
Рязанов Владимир Васильевич

Москва, 2019

Аннотация

В работе решается задача нахождения ассоциативных правил и, как самая важная ее часть, задача поиска частых множеств. Описаны стандартные алгоритмы нахождения частых множеств и предложен новый алгоритм FP-Growth-new, основанный на самом быстром из них - FP-Growth. Новый алгоритм проверен на реальных данных, после чего проведен анализ, в каких случаях этот алгоритм позволяет улучшить скорость решения задачи.

1 Введение

Нахождение ассоциативных правил - метод машинного обучения для поиска связей между переменными в больших наборах данных. Оно используется во многих прикладных задачах. Первый и самый очевидный пример - анализ рыночных корзин (который оказался настолько близким к данной задаче, что терминология из этой области сейчас постоянно используется и непосредственно в статьях по нахождению ассоциативных правил). Кроме того, также эти алгоритмы работают и при определении тематики документа, в биоинформатике, в Web mining и в других областях.

Неразрывно с задачей поиска ассоциативных правил следует задача о нахождении частых множеств, как ее подчасть. Более того, более 90 процентов времени при нахождении ассоциативных правил всегда тратится именно на нахождение множеств из элементов, часто встречаемых друг с другом. Именно поэтому подавляющее число работ в этой области пишется именно про алгоритмы поиска частых множеств.

1.1 Определения и обозначения

Здесь и далее используются стандартные обозначения. За *support* (поддержку) для множества товаров X обозначают $support(X) = \frac{w(X)}{T}$, где T - число всех транзакций, а $w(X)$ - число транзакций, где все товары X встречаются вместе. Таким образом, *support* множества товаров - показатель того, как часто встречается в имеющихся данных это множество.

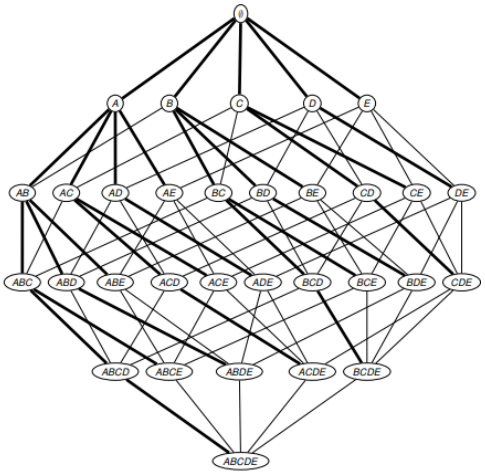
За $confidence$ (уверенность) для правила $X_1 \to X_2$ обозначают $conf(X_1 \to X_2) = \frac{support(X_1 \cup X_2)}{support(X_1)}$. Таким образом, $confidence$ ассоциативного правила - показатель того, как часто это правило срабатывает в имеющихся данных.

Кроме того, будем считать k-множеством множество из k элементов.

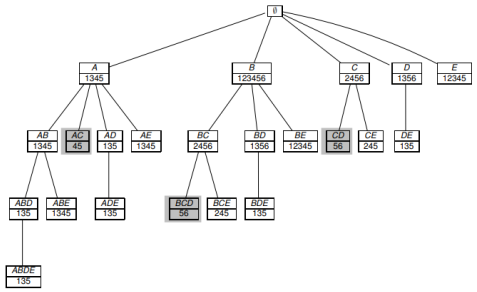
1.2 Задача поиска частых множеств и ассоциативных правил

В работе решается стандартная задача поиска частых множеств и вытекающая из нее задача поиска ассоциативных правил. И если по найденным множествам найти все правила, у которых $confidence$ больше заранее заданного значения, не очень трудно (для этого необходимо знать только *support* каждого из множеств правила), то первая задача более нетривиальная. В научной литературе рассматриваются три основных алгоритма, которые используются при поиске частых множеств для заданного заранее значения поддержки $supp_0$.

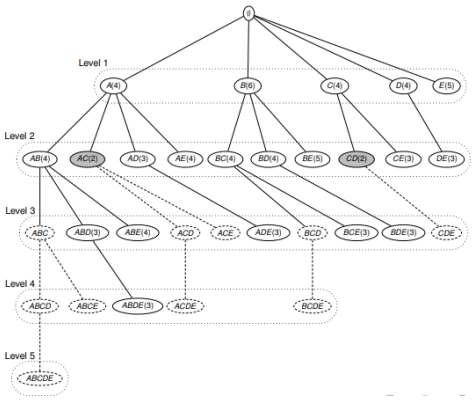
Первый из них - это Apriori[1], в котором строится полное дерево возможных транзакций. В алгоритме используется очевидное свойство монотонности - если множество X полностью содержится в множестве Y, то $supp(X) \geq supp(Y)$. Используя поиск в ширину, алгоритм проходит по дереву, считая показатели *support* для каждого множества. Как только этот показатель становится меньше, чем $supp_0$, то поиск по этой ветке прекращается. Таким образом мы либо доходим до конца дерева, либо



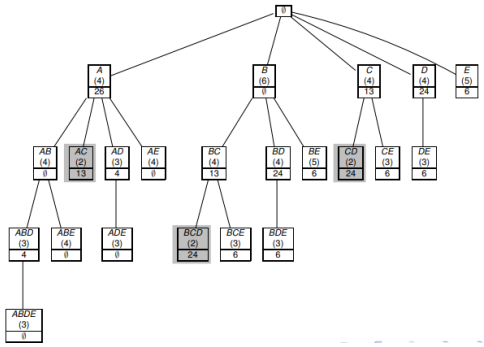
(a) рис.1



(b) рис.2



(a) рис.3



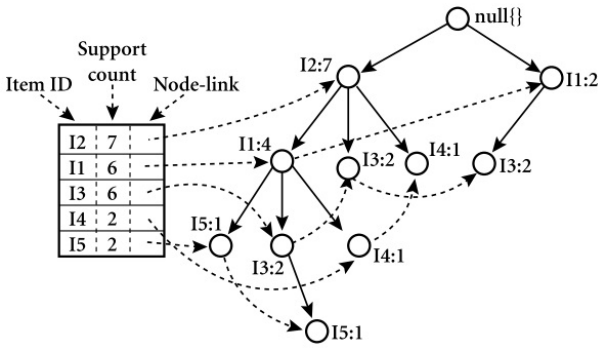
(b) рис.4

прекращаем поиск, так как на каждой ветке мы дошли до того уровня, на котором ее $support < supp_0$. Полностью построенное дерево можно увидеть на рис.1, а пример прохода по дереву - на рис.2.

Таким образом, сначала ищутся все 1-множества, удовлетворяющие ограничению, потом все 2-множества и так далее. Очевидный недостаток этого метода состоит в том, что для нахождения множеств необходимо обойти весь набор данных и считать $support$ для каждого k-множества, удовлетворяющего свойству монотонности.

Второй алгоритм - это ECLAT[2], где мы отказываемся от построения полного дерева и вместо этого строим префиксное дерево. Через поиск в глубину для каждого k-множества рекурсивно строятся все еще ранее не построенные $k+1$-множества до тех пор, пока $support$ каждого из этих $k+1$-множеств не становится меньше $supp_0$ - в этот момент мы возвращаемся вверх. Пример построенного префиксного дерева изображен на рис.3, а пример обхода по дереву - на рис.4.

Третий и самый современный алгоритм - это FP-Growth[3] (FP - от английского frequent patterns). Основное его отличие от предыдущих двух - отсутствие генерации кандидатов, которая необходима в Apriori и ECLAT. Для этого строится префиксное дерево не из возможных множеств, а сразу из всех транзакций (предварительно удалив все товары, которые редки сами по себе, а потому по свойству монотонности



(а) рис.5

не могут присутствовать в других, более крупных множествах). Будем называть такое дерево FP-деревом. Оно строится таким образом, что сверху оказываются самые популярные товары. В каждом узле дерева находится один товар, и он хранит указатели на своего отца, сына, а также свой *support*. Кроме того, для каждого товара генерируется таблица заголовков, в которой находится связный список всех узлов дерева, которые соответствуют этому товару. Пример дерева изображен на рис.5.

Итак, для каждого следующего товара существует два случая - если в его таблице заголовков только один узел, то товар встречается только в одной ветви, и достаточно просто пройти по ней и записать все встретившиеся множества. Все становится сложнее, когда в таблице больше одного узла - тогда для этого узла строится так называемая условная база образцов (conditional pattern base), которая сразу после этого используется для построения нового FP-дерева. Conditional pattern base, говоря простым языком, является компактным представлением той части от общего набора данных, в которой есть этот товар. Для примера рассмотрим ветку $a:4, b:3, c:3, d:2, e:2$ и построим условный образец для товара d. e мы не учитываем, так как он находится ниже (значит, мы уже успели посетить этот узел и нашли для него все наборы), а из верхней части ветки берем $a:2, b:2, c:2$, так как товаров d в этой ветке всего 2, а значит, он не может встречаться более 2 раз в тех множествах, которые можно получить из этой ветки. Строя такие образцы по каждой ветке для одного товара, мы получим полную информацию, по которой для него можно построить FP-дерево.

Таким образом получаем рекурсивный алгоритм. Исследования, такие как [4], показывают, что из этой тройки FP-Growth - самый быстрый алгоритм. Также именно его обычно используют как основу для создания еще более быстрых алгоритмов, и именно решение этой задачи является целью работы.

2 Алгоритм FP-Growth-new

В этом разделе описывается новый алгоритм поиска частых множеств - модификация стандартного алгоритма FP-Growth (далее называемый FP-Growth-new). Основное изменение состоит в избавлении от conditional pattern base - как будет показано ниже, новое FP-дерево для рекурсии можно построить и без него.

Действительно, conditional pattern base представляет собой множество образцов. Образцы состоят из символов и чисел-*support*. Если удастся напрямую вставлять образцы в FP-дерево, то рассмотренная база для построения дерева становится не нужна. Таким образом, в теории удастся ускорить алгоритм без использования дополнительной памяти.

Итак, мы рассматриваем построение FP-дерева для какого-то товара m (будем его называть m-FP-дерево). У нас есть ветви P для этого товара, которые получаются путем прохождения вверх от всех узлов этого товара до корня дерева. Начиная с корня для каждой ветви p из P, мы достраиваем m-FP-дерево таким образом: если товар i из ветви p уже есть в ветке m-FP-дерева, соответствующей этой p, то ничего не делаем, иначе создаем новый узел, куда записываем товар i с нулевым значением *support* (кроме одного исключения - если $i == m$(мы достигли конца ветки), то мы записываем в *support* не ноль, а соответствующее значение *support* для m из первоначального дерева, из которого мы и строим наше m-FP-дерево). После этого идем дальше по p. Стоит заметить, что если мы делали новый узел для i, то узел для любого товара j, который находится в p дальше от корня, чем i, тоже нужно будет создать.

После создания m-FP-дерева осталось корректно посчитать для него показатели *support*, что делается просто - снизу вверх для этого дерева суммируем все значения *support* (напомним, что изначально во всех узлах, кроме листьев, нули, а в листьях - значения *support* для m). Корректность этого подсчета вытекает из свойства монотонности, рассматриваемого ранее. Для примера можно рассмотреть дерево, под корнем которого находится товар a, из этого товара по одной ветке выходят b и m, по другой - d и m. Конечно, при построении m-FP-дерева нам интересны только множества, в которых был товар m, а поэтому здесь $support(b) == support(bm) == support(m)$. Аналогичное рассуждение верно и для товара d, а $support(a) == support(b) + support(d)$.

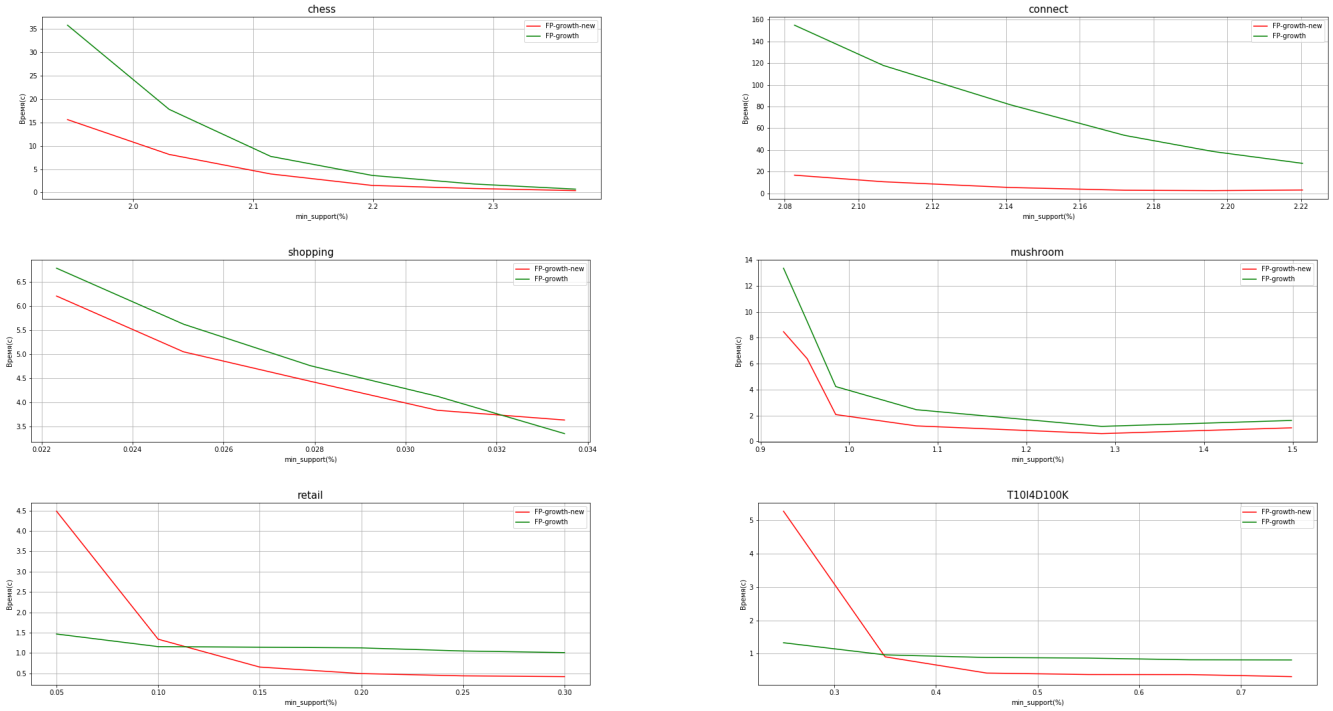
3 Вычислительные эксперименты

3.1 Исходные данные и условия экспериментов

В этом разделе проводится экспериментальное сравнение стандартного алгоритма FP-Growth и описанного выше алгоритма FP-Growth-new. Сравнение производилось на общедоступном сервере Kaggle Kernel. Оба алгоритма были запущены на нескольких наборах данных, которые можно найти на репозитории FIMI [5] - название каждого из наборов указано над соответствующим графиком. Новый алгоритм был реализован на языке Python, а реализация стандартного алгоритма FP-Growth для проведения сравнения была взята из [6].

3.2 Результаты экспериментов

Здесь представлены результаты проведенного сравнения. Алгоритмы сравнивались между собой по скорости их работы. Для каждого набора данных находились все частые множества с показателем *support*, большим либо равным какого-либо определенного значения, а также все ассоциативные правила из этих множеств с показателем $confidence$, большим либо равным 0.01.



Благодаря оптимизации, новый алгоритм работает быстрее на большинстве из предложенных наборов данных. Таким образом, на них за одинаковое время он может находить больше ассоциативных правил. С другой стороны, есть данные, на которых новый алгоритм работает хуже при существенном уменьшении показателя $min\_support$.

3.3 Анализ экспериментов

По результатам экспериментов было получено, что на одних наборах данных лучше себя проявляет новый алгоритм, а на других - классический. Стоит отметить, что общее свойство наборов, на которых FP-Growth-new работает быстрее - это большая плотность. Небольшое число товаров и большое число транзакций - идеальные свойства для этого алгоритма, так как в таком случае он не тратит время на постоянное построение новых conditional pattern base. С другой стороны, на менее плотных данных этот выигрыш почти не имеет значения, а непосредственное построение FP-дерева через conditional pattern base быстрее, чем используемое в FP-Growth-new.

Безусловно, хотелось бы обрабатывать любой набор данных максимально быстро, для чего хотелось бы уметь определять, какой алгоритм применять в той или иной ситуации. Скорее всего, для этого необходимо каким-то формальным образом

определить плотность набора данных и подобрать порог, начиная с которого становится эффективным использование того или иного алгоритма.

4 Заключение

В работе был предложен новый способ оптимизации алгоритма поиска частых множеств FP-Growth-new. Были проведены вычислительные эксперименты, подтверждающие более быструю работу алгоритма в некоторых случаях по сравнению со стандартным FP-Growth. Для проведения экспериментов предложенный новый алгоритм был реализован на языке Python.

Список литературы

[1] Agrawal, Rakesh and Srikant, Ramakrishnan; Fast algorithms for mining association rules in large databases in Bocca, Jorge B.; Jarke, Matthias; and Zaniolo, Carlo; editors, Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago, Chile, September 1994, pages 487-499

[2] Zaki, Mohammed Javeed; Parthasarathy, Srinivasan; Ogihara, Mitsunori; Li, Wei; New Algorithms for Fast Discovery of Association Rules, 1997, pages 283–286

[3] J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation in ACM SIGMOD Record, vol. 29, no. 2. ACM, 2000, pages 1–12

[4] Jeff Heaton, Comparing Dataset Characteristics that Favor the Apriori, Eclat or FP-Growth Frequent Itemset Mining Algorithms, 2017

[5] http://fimi.uantwerpen.be/data/

[6] https://github.com/evandempsey/fp-growth/