Санкт-Петербургский государственный политехнический университет

Институт прикладной математики и механики

Кафедра «Прикладная математика»

Отчет

по лабораторной работе №1

по дисциплине

«Инструментальные средства обработки больших массивов данных»

Выполнил студент гр.23641/2 Бурков Э.А.

Санкт-Петербург

2017

Введение

Создание дисциплины «Большие данные» (Big Data) было необходимым и естественным шагом при анализе данных. Если в начале XX-ого века для поиска зависимостей и построения моделей аналитики оперировали малыми выборками с малым числом признаков, то в конце того же века размер как размер выборок, так и число их признаков увеличилось в сотни и тысячи раз. Этому поспособствовало появление больших вычислительных мощностей и средств хранения данных. Провальные опыты стали храниться в базе данных, и на их основе можно было строить более точные модели проведения экспериментов. Анализировать большие данные в дальнейшем стали не только в науке, но и в повседневной жизни, например для выявления группы товаров с наибольшим потребительским спросом, расчета банковского риска при выдаче кредита, поиска нагруженных транспортом участков дороги и другими задачами.

Одной из таких задач является поиск ассоциативных правил.

Постановка задачи

Имеется набор объектов (признаков) из множества X. Будем считать, что объекты были предварительно пронумерованы и само множество X содержит только номера исходных объектов. Также имеется выборка S размером n. Элементом этой выборки является подмножество номеров объектов из множества X. Таким образом:

$$\text{Набор номеров объектов: } X$$

$$\text{Выборка: } S = \{X_i\}, i \in |1..n|$$

$$\text{Набор: } Y = \{y_1, y_2, \dots, y_k\}, y_k \in X$$

$$\text{Элемент выборки: } X_i = \{x_{1i}, x_{i2}, \dots, x_{mi}\},\ x_{ij} \in X$$

Элемент выборки – набор, который принадлежит выборке.

Поддержка (частота встречаемости, support) набора Y – отношение числа вхождения всех объектов из набора в наборы из выборки к её размеру.

$$\text{Поддержка: } \delta(Y) = \frac{\#\{X \in S | Y \subset X\}}{n}$$

Минимальная поддержка обозначается, как δ_{min}.

Если набор Y: $\delta(Y) \geq \ \delta_{min}$, по такой набор называется частым.

Значимость (confidence) набора X к набору Y – отношение поддержки объединения этих наборов к поддержке набора Y.

$$\text{Значимость: } \gamma(X,Y) = \frac{\delta(X \cup Y)}{\delta(Y)}, X \cap Y = \emptyset$$

Минимальная значимость определяется как γ_{min}.

Ассоциативное правило это пара непересекающихся наборов X и Y, таких что:

1) Наборы X и Y совместно часто встречаются

$$\delta(X \cup Y) \geq \delta_{min}$$

2) Если встречается Y, то также часто встречается и X

$$\gamma(X,Y) \geq \gamma_{min}$$

Требуется решить задачу поиска ассоциативных правил для входной выборки S с помощью методов Apriori и FPGrowth.

Решение

Задача поиска ассоциативных правил делится на две подзадачи: поиск частных наборов и генерация на их основе ассоциативных правил. Алгоритмы Apriori[1] и FPGrowth[2] занимаются задачей поиска частых наборов, а задача генерации ассоциативных правил решается другим алгоритмом. Однако, при описании Apriori обычно имеют в виду комбинацию обоих алгоритмов (поиска частых наборов и генерации ассоциативных правил), так как в оригинальной работе автор описывал оба этих алгоритма. В данной задаче генерация ассоциативных правил будет производиться с помощью алгоритма, указанного в оригинальной статье.

Apriori основан на свойстве антимонотонности наборов ($X \subset Y \rightarrow \delta(X) \geq \delta(Y)$). Из свойства антимонотонности следует, что у нечастого набора все его подмножества тоже нечастые. Используя это следствие и стратегию поиска в ширину для нахождения частых наборов можно сформулировать алгоритм Apriori:

1) Построить множество частых наборов длины 1 (то есть содержащих только 1 элемент в наборе) - C_1, таких что их поддержка δ не меньше минимальной поддержки δ_{min}.

Инициализировать множество частых наборов L: $L = C_1$.

2) Для $k \in 2 : |X|$

1. Построить множество частых наборов длины k - C_k, используя C_{k-1}:

$$C_k = \{X \cup \{y\} | X \in C_{k-1} \ \ \& \ \ y \in C_1 \backslash X\}$$

2. Обновить множество частых наборов L:

$$L = L \cup \{X \in C_k \mid \delta(X) \geq \delta_{\min}\}$$

3. Если $C_k = \emptyset$, то выходим из цикла.

Содержимое множества частых наборов L и будет требуемым результатом.

Алгоритм FPGrowth использует отличную от поиска в ширину стратегию поиска частых наборов. Чтобы их найти, в алгоритме используется префиксное FP-дерево (prefix frequent pattern (FP) tree). В узлах дерева находится признак (его номер), множество дочерних вершин, а также поддержка этого узла, выражаемая в поддержке набора, состоящего из признаков, находящихся в вершинах на пути от корня дерева до этого узла. Узлы, состоящие из одних и тех же признаков, объединены в уровни. Алгоритм делится на 2 этапа: построение исходного FP-дерева и рекурсивный поиск частых наборов.

Алгоритм построение FP-дерева состоит в проходе по выборке S и, в порядке уменьшения частоты поддержки признаков (не меньшей, чем $\delta_{\min}$) в наборе, добавления в дерево новых узлов или обновления поддержки текущих. После завершения построения FP-дерева его уровни будут упорядочены по убыванию поддержки признака уровня. Пример FP-дерева представлен на рисунке 1.

матрица	слова
a - - d - f -	d a
a - c d e - -	d c a e
- b - d - - -	d b
- b c d - - -	d b c
- b c - - - -	b c
a b - d - - -	d b a
- b - d e - -	d b e
- b c - e - g	b c e
- - c d - f -	d c
a b - d - - -	d b a

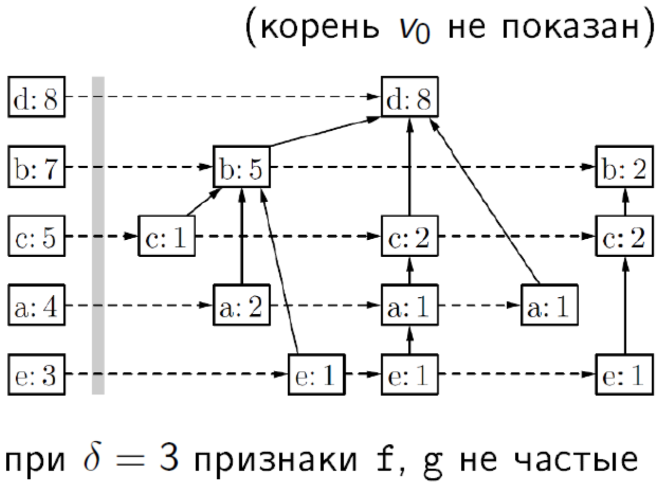

Рисунок 1. Пример FP-дерева и исходной выборки, по которой оно было построено.

В рекурсивном алгоритме поиска частых наборов используется условное FP-дерево по признаку x.

Условное FP-дерево (conditional FP-tree) по признаку x – FP-дерево, в котором удален уровень признака x и все потомки узлов, находящихся на этом уровне.

Построение условного FP-дерева можно разбить на несколько этапов:

1) Для каждого узла на уровне признака x:

1. Подняться до корня дерева, попутно сохраняя копии всех встреченных узлов или обновляя поддержки уже существующих.

2) Для каждого узла на уровне признака x:

1. Используя сохранённые копии узлов построить новое FP-дерево.

Полученное дерево будет условным FP-деревом по признаку x.

Теперь, имея исходное FP-дерево, можно провесим рекурсивный поиск частых наборов. Алгоритм поиска:

На вход алгоритма приходит: FP-дерево T, набор Y и список частых наборов L.

1) По уровням в порядке увеличения поддержки признака x, если его поддержка не меньше $\delta_{\min}$:

1. Добавить в список частых наборов L новый набор – $Y \cup \{x\}$.

2. Построить условное FP-дерево T’ по признаку x.

3. Рекурсивно найти частые наборы с параметрами T’, $Y \cup \{x\}$, L.

Итоговый список частных наборов L и будет являться требуемым результатом.

Тестирование

Алгоритмы были написаны на языке программирования Java. Для автоматической проверки алгоритмов были написаны юнит-тесты. Ниже приведен список тестов и их описание.

Тесты Apriori:

Тест	Описание
`testMaxSupportResultIsEmpty`	Проверка, что задание поддержки > 1 возвращает пустой результат.
`testMaxConfidenceResultIsEmpty`	Проверка, что задание значимость > 1 возвращает пустой результат.
`testProvedData`	Проверка ассоциативных правил, полученных алгоритмом, с правилами, которые были пред посчитаны (mock test)
`testAllData`	Проверка ассоциативных правил, полученных алгоритмом, факта наличия во всех правилах признака, повторяющегося в каждом элементе выборки
`testBigDataAndCompareFreqSetsWithFPGrowth`	Проверка факта того, что множество частых наборов, полученное Apriori, содержится во множестве частых наборов, полученных FPGrowth (размер выборки - 20000 записей)
`testBigDataAndCompareRulesWithFPGrowth`	Проверка факта того, что множество правил, полученное на основе частых наборов из Apriori, содержится во

	множестве правил, полученное на основе частых наборов из FPGrowth (размер выборки - 20000 записей)

Набор тестов для FPGrowth зеркально идентичен набору тестов для Apriori, с поправкой на то, что сравнение в последних двух тестах делается с Apriori, а не FPGrowth.

Результаты

На основе проведённых тестов можно построить зависимость времени от минимальной поддержки (рисунки 2 и 3).

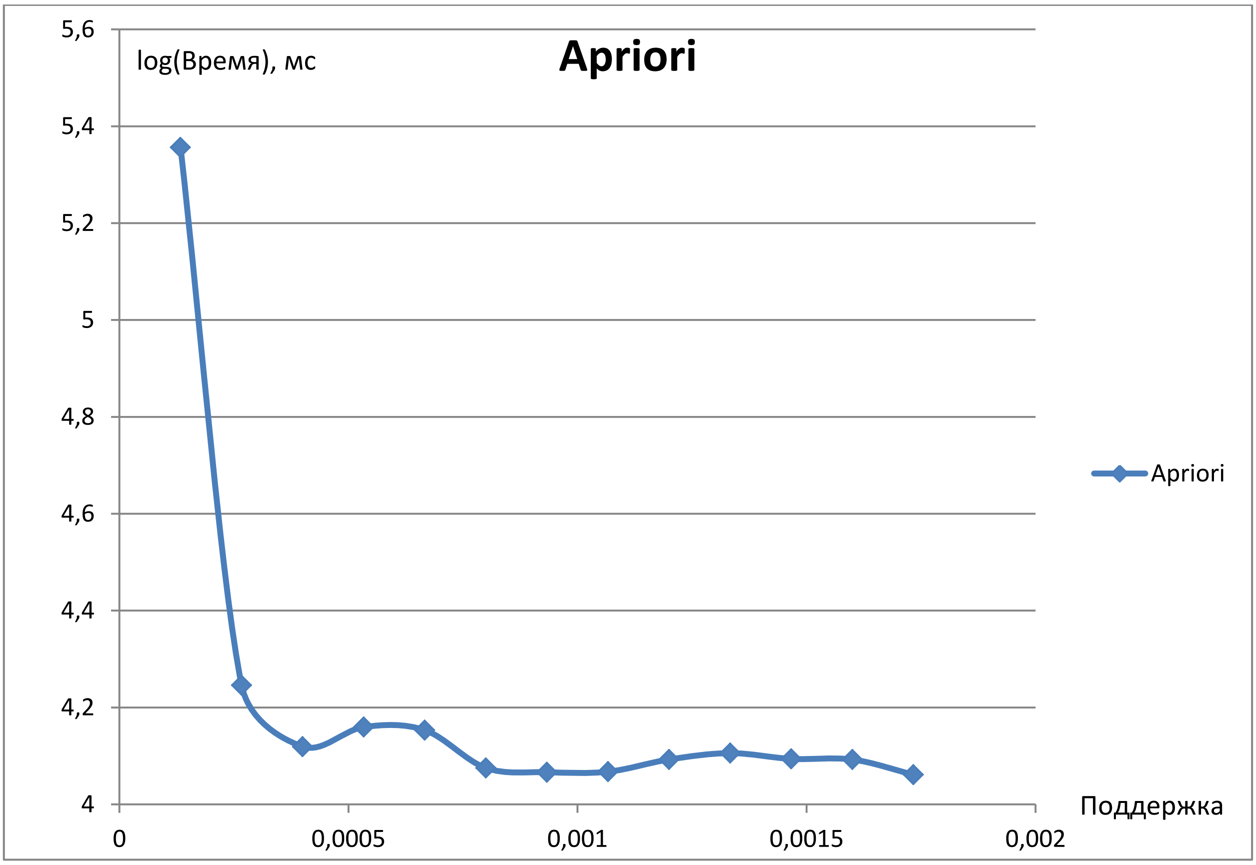

Рисунок 2. Зависимость времени работы алгоритмов от минимальной поддержки для алгоритма Apriori

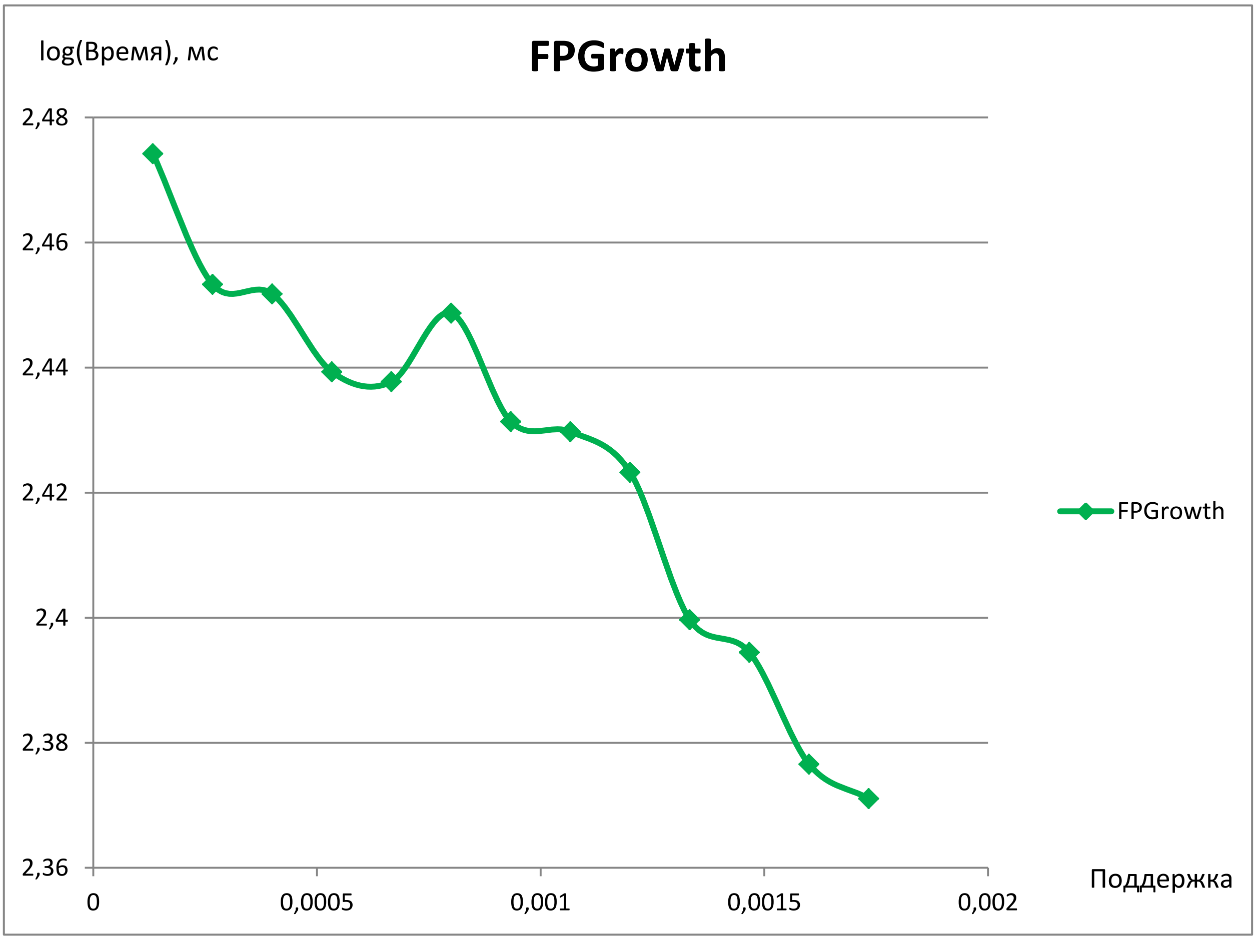

Рисунок 3. Зависимость времени работы алгоритмов от минимальной поддержки для алгоритма FPGrowth

С помощью алгоритмов найдём ассоциативные правила в выборке «Traffic Accidents Data Set», полученные Национальный Институтом Статистики (NIS) для региона Фландерса (Бельгия) в период с 1991 по 2000 годы. Описание данных находится в документе [3], а сама выборка хранится в файле “acc.txt”.

Найдём ассоциативные правила с параметрами (0.9, 0.7): (support, confidence). Их число равно 218. Для анализа выберем признак, который мы будем исследовать. Выберем признак под номером 21, faulty lighting. Наибольшей значимостью (все первые 3 цифры после запятой - 9) обладают правила:

[12, 18, 21] -> [17]
[12, 21] -> [17]
[18, 21] -> [12, 17]
[21] -> [17]

Если перевести числовые значения в смысловые, то получится набор правил:

[intersection traffic signs, roundabout, faulty lighting] -> [railroad]
[intersection traffic signs, faulty lighting] -> [railroad]
[roundabout, faulty lighting] -> [intersection traffic signs, railroad]
[faulty lighting] -> [railroad]

Из этих правил видно, что аварии вблизи железнодорожных путей случались при нерегулируемом перекрёстке и недостаточном освещении, иногда при наличии кругового движения или съезда с него. На основе этих правил можно сделать предположение, что на выходе с кругового движения, который переходил в перекрёсток, неподалёку от железной дороги было недостаточное освещение, что приводило к авариям. При дальнейшем анализе этих данных видно, что они взаимозависимы (находятся в правилах друг друга), что сильнее подкрепляет это предположение.

Выводы

Алгоритм Apriori был впервые предложен в 94-м году, на заре развития дисциплины «Data Mining». Время его работы в несколько раз превосходило алгоритмы, которые были предложены годом ранее (AIS и SETM). Помимо него в оригинальной работе было упомянута его модификация - AprioriTid – и AprioriHybrid, являющаяся комбинацией Apriori и AprioriTid. В 2000-м году был предложен алгоритм FPGrowth, время работы которого превышало время работы уже Apriori в несколько раз, что видно при сравнении рисунков 2 и 3. Время работы уменьшилось из-за использования структуры дерева для хранения выборки и стратегии поиска частых наборов в виде прохода по дереву.

С точки зрения производительности FPGrowth превосходит Apriori. Но это не отменяет исторической значимости этого алгоритма для дисциплины «Data Mining».

Литература и источники

1. R. Agrawal and R. Srikant. «Fast algorithms for mining association rules in large databases». Research Report RJ 9839, IBM Almaden Research Center, San Jose, California, June 1994

2. Han, J., Pei, J., & Yin, Y. «Mining frequent patterns without candidate generation», ACM SIGMOD Record (Vol. 29, No. 2, pp. 1-12), May 2000

3. «Traffic Accidents Data Set», Karolien Geurts, Research Group Data and Modelling, Limburgs Universitair Centrum, Universitaire Campus, B-3590 Diepenbeek, BELGIUM

Приложение

Результаты работы алгоритма для параметров (0.9, 0.7), для элемента с номером 21.

Rule: [12] -> [21], conf = 0.9055942304802925

Rule: [12] -> [17, 21], conf = 0.9054940652075926

Rule: [12] -> [18, 21], conf = 0.9032904292081935

Rule: [12, 17] -> [21], conf = 0.9055847733533684

Rule: [12, 17] -> [18, 21], conf = 0.9032807412972702

Rule: [12, 18] -> [21], conf = 0.9056035348463547

Rule: [12, 18] -> [17, 21], conf = 0.9055031130749146

Rule: [12, 21] -> [17], conf = 0.999889392766287

Rule: [12, 17, 18] -> [21], conf = 0.9055940544340665

Rule: [12, 17, 21] -> [18], conf = 0.9974557522123894

Rule: [12, 18, 21] -> [17], conf = 0.9998891106675538

Rule: [17] -> [21], conf = 0.9056405640564057

Rule: [17] -> [12, 21], conf = 0.9040904090409041

Rule: [17] -> [18, 21], conf = 0.9022902290229022

Rule: [17] -> [12, 18, 21], conf = 0.9017901790179018

Rule: [17, 18] -> [21], conf = 0.9055959849435383

Rule: [17, 18] -> [12, 21], conf = 0.9050941028858218

Rule: [18] -> [21], conf = 0.9056054599287399

Rule: [18] -> [12, 21], conf = 0.9051036282430873

Rule: [18] -> [17, 21], conf = 0.9055050935916094

Rule: [18] -> [12, 17, 21], conf = 0.9050032619059567

Rule: [18, 21] -> [12, 17], conf = 0.9993350326942259

Rule: [21] -> [12], conf = 0.9982885220559818

Rule: [21] -> [17], conf = 0.9998895820681278

Rule: [21] -> [18], conf = 0.9963009992822834

Rule: [21] -> [12, 17], conf = 0.9981781041241098

Rule: [21] -> [12, 18], conf = 0.9957489096229227

Rule: [21] -> [17, 18], conf = 0.9961905813504113

Rule: [21] -> [12, 17, 18], conf = 0.9956384916910507