Data Mining Algorithms In R/Frequent Pattern Mining/The Apriori Algorithm

Introduction

In computer science and data mining, **Apriori** is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions. As is common in association rule mining, given a set of **itemsets**, the algorithm attempts to find subsets which are common to at least a minimum number C of the itemsets. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breadth-first search and a tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k-1$. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k -length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset S only after all $2^{|S|}-1$ of its proper subsets.

The quest to mine frequent patterns appears in many domains. The prototypical application is market basket analysis, i.e., to mine the sets of items that are frequent bought together, at a supermarket by analyzing the customer shopping carts (the so-called “market baskets”). Once we mine the frequent sets, they allow us to extract association rules among the item sets, where we make some statement about how likely are two sets of items to co-occur or to conditionally occur. For example, in the weblog scenario frequent sets allow us to extract rules like, “Users who visit the sets of pages main, laptops and rebates also visit the pages shopping-cart and checkout”, indicating, perhaps, that the special rebate offer is resulting in more laptop sales. In the case of market baskets, we can find rules like, “Customers who buy Milk and Cereal also tend to buy Bananas”, which may prompt a grocery store to co-locate bananas in the cereal aisle.

Algorithm

The following is a formal statement of the problem: Let $\tau = \{i_1, i_2, ..., i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq \tau$. Associated with each transaction is a unique identifier, called its TID. We say that a transaction T contains X, a set of some items in τ, if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset \tau$, $Y \subset \tau$, and $X \cap Y = \varnothing$. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if $c\%$ of transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has support s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$. Given a set of transactions D, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called minsup) and minimum confidence (called minconf) respectively.

The problem is usually decomposed into two subproblems. One is to find those itemsets whose occurrences exceed a predefined threshold in the database; those itemsets are called frequent or large itemsets. The second problem is to generate association rules from those large itemsets with the constraints of minimal confidence. Suppose one of the large itemsets is Lk, Lk = {I1, I2, … , Ik}, association rules with this itemsets are generated in the following way: the first rule is {I1, I2, … , Ik-1}⇒ {Ik}, by checking the confidence this rule can be determined as interesting or not. Then other rule are generated by deleting the last items in the antecedent and inserting it to the consequent,

further the confidences of the new rules are checked to determine the interestingness of them. Those processes iterated until the antecedent becomes empty. Since the second subproblem is quite straight forward, most of the researches focus on the first subproblem. The Apriori algorithm finds the frequent sets L In Database D .

Let X , $Y \subseteq$ I be any two itemsets. Observe that if $X \subseteq Y$, then sup(X) ≥ sup(Y), which leads to the following two corollaries:

- If X is frequent, then any subset Y ⊆ X is also frequent.
- If X is not frequent, then any superset Y ⊇ X cannot be frequent.

Based on the above observations, we can significantly improve the itemset mining algorithm by reducing the number of candidates we generate, by limiting the candidates to be only those that will potentially be frequent. First we can stop generating supersets of a candidate once we determine that it is infrequent, since no superset of an infrequent itemset can be frequent. Second, we can avoid any candidate that has an infrequent subset. These two observations can result in significant pruning of the search space.

- Find frequent set L_{k-1}.
- Join Step.
  - C_k is generated by joining L_{k-1} with itself
- Prune Step.
  - Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k -itemset, hence should be removed.

where

- (C_k : Candidate itemset of size k)
- (L_k : frequent itemset of size k)

Apriori Pseudocode

Apriori (T, ε)

$L_1 \leftarrow \{$ *large 1-itemsets that appear in more than* ε *transactions* $\}$

$k \leftarrow 2$

while $L_{k-1} \neq \varnothing$

$C_k \leftarrow$ *Generate* (L_{k-1})

for transactions $t \in T$

$C_t \leftarrow$ Subset (C_k, t)

for candidates $c \in C_t$

$\text{count}[c] \leftarrow \text{count}[c] + 1$

$L_k \leftarrow \{c \in C_k | \ \text{count}[c] \geq \varepsilon\}$

$k \leftarrow k + 1$

return $\bigcup_k L_k$

Implementation

The R package that implements association rule mining is called arules, and it can be found at http://cran.r-project.org/web/packages/arules/index.html (Details how to install R can be found here http://cran.r-project.org/bin/linux/ubuntu). In package arules we interface free reference implementations of Apriori and Eclat by Christian Borgelt (Borgelt and Kruse, 2002; Borgelt, 2003)3. The code is called directly from R by the functions apriori() and éclat() and the data objects are directly passed from R to the C code and back without writing to external files. The

implementations can mine frequent itemsets, and closed and maximal frequent itemsets. In addition, apriori() can also mine association rules.

Installation of the dependencies packages can be performed within the R environment using the function "install.packages("*package name*")". The name of the package in question is "*arules*". To use this package you need R environment with version 2.7.0 at least (details how to update R can be found in http://cran.r-project.org/bin/linux/ubuntu).

```
install.packages("arules")
```

After the user installed the necessaries packages, he must load them. This can be done using the function "library(*package name*)".

```
library(arules)
```

Visualization

Reading the data

Transactions can be read from files in the basket format, with the command read.transactions. The parameters for read.transactions are:

file

format read.transactions can get data structured in multiple formats, one of them being basket.

separator(sep)

One example of it's use would be:

```
tr<-read.transactions("teste",format="basket",sep=",")
```

The object "tr" is used to store the transactions read from the file named "teste", where each item is separated by a ",". "teste" could be, for example:

```
A,B,C
B,C
A,B,D
A,B,C,D
A
B
```

One way to visualize the data is inspect(object). For example:

```
inspect(tr)
```

```
items
1 {A,
   B,
   C}
2 {B,
   C}
3 {A,
   B,
   D}
4 {A,
   B,
```

```
   C,
   D}
5 {A}
6 {B}
```

Additionally, you can visually inspect binary incidence matrices, or plot the frequency of items sets:

```
image(tr)
itemFrequencyPlot(tr, support = 0.1)
```

To show the number of items in transactions read from the file named ”teste” do:

```
length(tr)
```

```
[1] 6
```

Rules

The function to mine frequent itemsets, association rules or association hyperedges, using the Apriori algorithm, takes 2 parameters:

Data: the object that contains the data

parameter: a multi-dimensional parameter to set up support and confidence

For example, using the dataset gathered in the previous section:

```
rules <- apriori(tr, parameter= list(supp=0.5, conf=0.5))
```

The rules can be visualized with the command inspect:

```
inspect(rules)
```

```
    rhs    support confidence lift
1 {} => {C} 0.5000000 0.5000000 1.0
2 {} => {A} 0.6666667 0.6666667 1.0
3 {} => {B} 0.8333333 0.8333333 1.0
4 {C} => {B} 0.5000000 1.0000000 1.2
5 {B} => {C} 0.5000000 0.6000000 1.2
6 {A} => {B} 0.5000000 0.7500000 0.9
7 {B} => {A} 0.5000000 0.6000000 0.9
```

To get a summary of the rules' characteristics, the function ”summary” can be used:

```
summary(rules)
```

```
set of 7 rules

rule length distribution (lhs + rhs):sizes
1 2
3 4

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   2.000   1.571   2.000   2.000

summary of quality measures:
```

```
    support         confidence          lift
 Min.   :0.5000   Min.   :0.5000   Min.   :0.900
 1st Qu.:0.5000   1st Qu.:0.6000   1st Qu.:0.950
 Median :0.5000   Median :0.6667   Median :1.000
 Mean   :0.5714   Mean   :0.7071   Mean   :1.029
 3rd Qu.:0.5833   3rd Qu.:0.7917   3rd Qu.:1.100
 Max.   :0.8333   Max.   :1.0000   Max.   :1.200

mining info:
 data ntransactions support confidence
   tr             6     0.5        0.5
```

Outher quality measures of the rules can be displaed with:

```
interestMeasure(rules, c("support", "chiSquare", "confidence",
"conviction",
      "cosine", "coverage", "leverage", "lift", "oddsRatio"), tr)
```

```
    support chiSquare confidence conviction    cosine  coverage    leverage
1 0.5000000       NaN  0.5000000  1.0000000 0.7071068 1.0000000  0.00000000
2 0.6666667       NaN  0.6666667  1.0000000 0.8164966 1.0000000  0.00000000
3 0.8333333       NaN  0.8333333  1.0000000 0.9128709 1.0000000  0.00000000
4 0.5000000       1.2  1.0000000        Inf 0.7745967 0.5000000  0.08333333
5 0.5000000       1.2  0.6000000  1.2500000 0.7745967 0.8333333  0.08333333
6 0.5000000       0.6  0.7500000  0.6666667 0.6708204 0.6666667 -0.05555556
7 0.5000000       0.6  0.6000000  0.8333333 0.6708204 0.8333333 -0.05555556
  lift oddsRatio
1  1.0       NaN
2  1.0       NaN
3  1.0       NaN
4  1.2       Inf
5  1.2       Inf
6  0.9         0
7  0.9         0
```

To calculate a single measure and add it to the quality slot:

```
quality(rules) <- cbind(quality(rules), hyperConfidence = interestMeasure(rules, method = "hyperConfidence", Income))

inspect(head(SORT(rules, by = "hyperConfidence")))
```

```
  lhs     rhs    support confidence lift hyperConfidence
1 {C} => {B} 0.5000000  1.0000000  1.2             0.5
2 {B} => {C} 0.5000000  0.6000000  1.2             0.5
3 {}  => {C} 0.5000000  0.5000000  1.0             0.0
4 {}  => {A} 0.6666667  0.6666667  1.0             0.0
5 {}  => {B} 0.8333333  0.8333333  1.0             0.0
6 {A} => {B} 0.5000000  0.7500000  0.9             0.0
```

Finally, to send the output to a file use:

```
sink("sink-examp.txt")
inspect(head(SORT(rules, by = "hyperConfidence")))
...
```

Case study

We now present a brief case study to illustrate the use of **apriori** and the **package** arules in a real data set.

Scenario

Spam is the abuse of electronic messaging systems (including most broadcast media, digital delivery systems) to send unsolicited bulk messages indiscriminately. Many filters are designed to stop spam in its tracks, but such filters are often left behind by the spammers' obfuscating techniques. One of such filters is SpamAssassin, a well known filter that relies on machine learning and fixed rules for filtering. SpamAssassin, however, doesn't consider the possible relations between its rules.

Input Data

22406 spams were used in this case study, ranging from 1998 to 2009, collected from the public spam corpus SpamArchive (http://untroubled.org/spam/).Each spam was analyzed by SpamAssassin, and the rules found were analyzed, as well its year. For example, a particular spam would be: YEAR=1998 INVALID_DATE INVALID_TZ_EST meaning this was a spam from 1998 in which SpamAssassin accused two rules: INVALID_DATE and INVALID_TZ_EST.

Implementation

The output from each spam filtered through SpamAssassin was saved in a file named output. The following commands were issued:

```
library("arules"); #Using the package
tr<-read.transactions("output",format="basket",sep=" ") #Reading each spam from the output file, separated by ' ' (single spaces).
rules <- apriori(tr, parameter= list(supp=0.05, conf=0.3)) #Running apriori
inspect(rules) #Visualizing the association rules found.
```

Output Data

We displayed only the itemsets of size 1, for simplicity:

```
    lhs                            rhs                            support    confidence   lift
1   {}                          => {RDNS_NONE}                    0.40828350  0.4082835  1.0000000
2   {}                          => {MIME_HTML_ONLY}               0.30509685  0.3050968  1.0000000
3   {}                          => {HTML_MESSAGE}                 0.51347853  0.5134785  1.0000000
4   {YEAR=2008}                 => {RDNS_NONE}                    0.06663394  0.7465000  1.8283864
5   {YEAR=2009}                 => {RDNS_NONE}                    0.07007052  0.7850000  1.9226836
6   {YEAR=2009}                 => {HTML_MESSAGE}                 0.05877890  0.6585000  1.2824295
7   {RATWARE_OUTLOOK_NONAME}    => {RATWARE_MS_HASH}              0.05404802  0.9991749 12.9482436
8   {RATWARE_MS_HASH}           => {RATWARE_OUTLOOK_NONAME}       0.05404802  0.7004049 12.9482436
9   {MISSING_DATE}              => {MISSING_MID}                  0.05538695  0.7231935  4.0540087
10  {MISSING_MID}               => {MISSING_DATE}                 0.05538695  0.3104829  4.0540087
11  {HELO_DYNAMIC_IPADDR}       => {FH_HELO_EQ_D_D_D_D}           0.05400339  0.8916728 10.0648972
12  {FH_HELO_EQ_D_D_D_D}        => {HELO_DYNAMIC_IPADDR}          0.05400339  0.6095718 10.0648972
13  {YEAR=2007}                 => {RDNS_NONE}                    0.05543158  0.6210000  1.5210020
14  {YEAR=2007}                 => {HTML_MESSAGE}                 0.05346782  0.5990000  1.1665532
```

```
15  {FORGED_OUTLOOK_TAGS}    => {HTML_MESSAGE}           0.05163795  0.9974138  1.9424644
16  {SUBJ_ILLEGAL_CHARS}     => {SUBJECT_NEEDS_ENCODING} 0.05038829  0.9982317 17.8645195
17  {SUBJECT_NEEDS_ENCODING} => {SUBJ_ILLEGAL_CHARS}     0.05038829  0.9017572 17.8645195
18  {RATWARE_MS_HASH}        => {MSGID_OUTLOOK_INVALID}  0.06667857  0.8640833  8.7604752
19  {MSGID_OUTLOOK_INVALID}  => {RATWARE_MS_HASH}        0.06667857  0.6760181  8.7604752
20  {YEAR=2002}              => {HTML_MESSAGE}           0.05636883  0.6315000  1.2298469
21  {DATE_SPAMWARE_Y2K}      => {INVALID_DATE}           0.05511916  1.0000000  5.8394579
22  {INVALID_DATE}           => {DATE_SPAMWARE_Y2K}      0.05511916  0.3218660  5.8394579
23  {RCVD_HELO_IP_MISMATCH}  => {RCVD_NUMERIC_HELO}      0.06190306  0.9992795  7.4982777
24  {RCVD_NUMERIC_HELO}      => {RCVD_HELO_IP_MISMATCH}  0.06190306  0.4645010  7.4982777
25  {MIME_QP_LONG_LINE}      => {HTML_MESSAGE}           0.07033830  0.8602620  1.6753612
26  {FH_HELO_EQ_D_D_D_D}     => {HTML_MESSAGE}           0.05819870  0.6569270  1.2793660
27  {FORGED_OUTLOOK_HTML}    => {MIME_HTML_ONLY}         0.06538427  1.0000000  3.2776477
28  {FORGED_OUTLOOK_HTML}    => {HTML_MESSAGE}           0.06538427  1.0000000  1.9475011
29  {MSGID_OUTLOOK_INVALID}  => {MISSING_MIMEOLE}        0.06261716  0.6348416  4.1301572
30  {MISSING_MIMEOLE}        => {MSGID_OUTLOOK_INVALID}  0.06261716  0.4073751  4.1301572
31  {MSGID_OUTLOOK_INVALID}  => {MIME_HTML_ONLY}         0.05752923  0.5832579  1.9117140
32  {MSGID_OUTLOOK_INVALID}  => {HTML_MESSAGE}           0.06364367  0.6452489  1.2566229
33  {YEAR=2003}              => {MIME_HTML_ONLY}         0.05766313  0.6460000  2.1173604
34  {YEAR=2003}              => {HTML_MESSAGE}           0.06895474  0.7725000  1.5044446
35  {YEAR=2004}              => {HTML_MESSAGE}           0.07042756  0.7890000  1.5365784
36  {FORGED_MUA_OUTLOOK}     => {HTML_MESSAGE}           0.05681514  0.5912680  1.1514951
37  {MIME_HTML_ONLY_MULTI}   => {MPART_ALT_DIFF}         0.06583058  0.9886059  9.9285987
38  {MPART_ALT_DIFF}         => {MIME_HTML_ONLY_MULTI}   0.06583058  0.6611385  9.9285987
39  {MIME_HTML_ONLY_MULTI}   => {MISSING_MIMEOLE}        0.05315540  0.7982574  5.1933086
40  {MISSING_MIMEOLE}        => {MIME_HTML_ONLY_MULTI}   0.05315540  0.3458188  5.1933086
41  {MIME_HTML_ONLY_MULTI}   => {MIME_HTML_ONLY}         0.06658931  1.0000000  3.2776477
42  {MIME_HTML_ONLY_MULTI}   => {HTML_MESSAGE}           0.06658931  1.0000000  1.9475011
43  {MISSING_MID}            => {RDNS_NONE}              0.07060609  0.3957968  0.9694167
44  {MISSING_MID}            => {MIME_HTML_ONLY}         0.07730072  0.4333250  1.4202867
45  {MISSING_MID}            => {HTML_MESSAGE}           0.09515308  0.5334001  1.0387972
46  {MPART_ALT_DIFF}         => {MISSING_MIMEOLE}        0.05337856  0.5360825  3.4876492
47  {MISSING_MIMEOLE}        => {MPART_ALT_DIFF}         0.05337856  0.3472706  3.4876492
48  {MPART_ALT_DIFF}         => {MIME_HTML_ONLY}         0.06591984  0.6620350  2.1699174
49  {MPART_ALT_DIFF}         => {HTML_MESSAGE}           0.09957154  1.0000000  1.9475011
50  {RCVD_NUMERIC_HELO}      => {MISSING_MIMEOLE}        0.05369098  0.4028801  2.6210603
51  {MISSING_MIMEOLE}        => {RCVD_NUMERIC_HELO}      0.05369098  0.3493031  2.6210603
52  {RCVD_NUMERIC_HELO}      => {RDNS_NONE}              0.07752388  0.5817147  1.4247812
53  {RCVD_NUMERIC_HELO}      => {MIME_HTML_ONLY}         0.05855574  0.4393838  1.4401453
54  {RCVD_NUMERIC_HELO}      => {HTML_MESSAGE}           0.07877354  0.5910918  1.1511518
55  {RDNS_DYNAMIC}           => {MIME_HTML_ONLY}         0.06257253  0.4054367  1.3288786
56  {RDNS_DYNAMIC}           => {HTML_MESSAGE}           0.09975007  0.6463274  1.2587232
57  {INVALID_DATE}           => {MIME_HTML_ONLY}         0.05467286  0.3192598  1.0464213
58  {INVALID_DATE}           => {HTML_MESSAGE}           0.06712488  0.3919729  0.7633676
59  {MISSING_MIMEOLE}        => {MIME_HTML_ONLY}         0.10381148  0.6753775  2.2136494
60  {MIME_HTML_ONLY}         => {MISSING_MIMEOLE}        0.10381148  0.3402575  2.2136494
61  {MISSING_MIMEOLE}        => {HTML_MESSAGE}           0.10648933  0.6927991  1.3492269
```

```
62  {MIME_HTML_ONLY}          => {RDNS_NONE}             0.11867357  0.3889702  0.9526963
63  {RDNS_NONE}               => {HTML_MESSAGE}          0.23216995  0.5686489  1.1074443
64  {HTML_MESSAGE}            => {RDNS_NONE}             0.23216995  0.4521512  1.1074443
65  {MIME_HTML_ONLY}          => {HTML_MESSAGE}          0.30505222  0.9998537  1.9472162
66  {HTML_MESSAGE}            => {MIME_HTML_ONLY}        0.30505222  0.5940895  1.9472162
```

Analysis

It takes a critical eye to spot the interesting information. Rules such as YEAR=2009 => HTML_MESSAGE carry information that is not surprising, deeming it uninteresting. However, some interesting patterns can be found:

MISSING_DATE => MISSING_MID

> Spammers who don't set up a date usually 'forget' about the message ID as well. Maybe spam assassin could score differently when they are found together.

HELO_DYNAMIC_IPADDR => FH_HELO_EQ_D_D_D_D

> When HELO is done using a suspicious hostname, it is usually in the d-d-d-d format. This characterizes a spam more than just one of the two rules separately.

SUBJECT_NEEDS_ENCODING => SUBJ_ILLEGAL_CHARS

> Typically this means a spam is in Chinese. These rules obviously have a strong correlation (confidence= 90%).

Many more interesting patterns can be found and studied, in order to improve the quality of the filter. Using apriori with "arules" is an easy and straightforward task.

References

- [1]http://en.wikipedia.org/wiki/Apriori_algorithm
- [2]http://rss.acs.unt.edu/Rdoc/library/arules/html/apriori.html
- [3]http://untroubled.org/spam/

Article Sources and Contributors

Data Mining Algorithms In R/Frequent Pattern Mining/The Apriori Algorithm *Source*: http://en.wikibooks.org/w/index.php?oldid=2064531 *Contributors*: -

License