UNIVERSITY OF SKÖVDE

Assignment 3

Apriori Algorithm

ALEXANDER CROLL

Data Mining, IT721A

Data Science, University of Skövde

Alan Said, Niclas Ståhl

September 20, 2017

Table of Contents

1. Abstract

The purpose of this document is to provide documentation for the third programming assignment in Data Mining (IT721A) course.

This paper accompanies the Python code within the .zip file *DM2017_b16alecr_assignment3_code.zip*. There, code for an implementation of Apriori algorithm was written. In this document, the solution will be briefly described and the obtained results will be presented.

‘

2. Description of Solution

All relevant code to Apriori is contained in the file *Apriori.py*. There are no additional files, e.g. for data loading.

The library *numpy* is imported for handling data in Python, but no other machine learning libraries were used for this implementation.

The assignment is divided into several steps/functions which will be described in the subsequent paragraphs.

2.1. Apriori

The run_*apriori()* function is the main function of the apriori algorithm. It is responsible for the flow through the application with the following steps:

1. Loading data, which calls the function *read_basket_data()*
2. Generating frequent itemsets, which calls the function *generate_frequent()*. This step is also responsible for setting the minimum support for items in itemsets.
3. Generating association rules, which calls the function *generate_rules()*. This step is also responsible for setting the minimum confidence for any rules.
4. Printing association rules, which calls the function *print_rule()*.

2.2. Step 1 – Generate Frequent Itemset

The function *generate_frequent()* takes two arguments:

- X, which is the data (basket data)
- min_support, which is the value for the minimum support that a frequent item set should have

Basically, the *generate_frequent()* function is split into two parts: generating the k1-itemset first, before generating more complex item sets. This iterative generation continues until there are no more item sets to be generated.

For this purpose, two nested functions exist:

- *generate_k1()*
  This function takes two arguments: X (the dataset) and min_support which is the minimum support that an item set should have out of all transactions.
  Each transaction consists of a series of ones and zeros, indicating whether or not an item was bought in this specific transaction. The k1-itemsets represent an item set where only one item was bought. In order to create these k1-itemsets, the function

iterates through a range of the length of the transactions. At each iteration, an item set with only zeros is created first, before it is filled with exactly one “one” at the current range index. Before this k1-itemset is allowed to be added to the array with all k1-itemsets, it first has to fulfill the condition of minimum support.

- *generate_kx()*

  This function takes three arguments: X (the dataset), k (all k1-itemsets) and min_support which is the minimum support that an item set should have out of all transactions.

  The function then takes each k1-itemset and will combine it with each of the remaining k1-itemsets iteratively. Combinations are done by using numpy’s *logical_or*.

  For instance, the following two itemsets

  ```
  1    0    0    0    0    0
  0    1    0    0    0    0
  ```

  would be combined into

  ```
  1    1    0    0    0    0
  ```

  Once combination is done, support for each new item set is calculated. If minimum support is passed and the item set does not yet exist (*check_item()*), the new item set and its support are appended into corresponding arrays.

2.3. Step 2 – Generate Rules

The function *generate_rules()* takes three arguments:

- X, which is the dataset
- itemset, which is an array of all frequent item sets
- min_confidence, which is the minimum confidence a rule should have

The function loops through the array with all item sets and then tries to create rules by looping through the remaining item sets. Within the first loop, the antecedents (left side of the rule) and in the second loop the consequents (right side of the rule) are selected.

Associations are valid if both sides are not equal, the right-hand side includes the items from the left-hand side (that are used to draw associations from) and if a rule does not yet exist.

Further, each rule has to pass the minimum confidence.

2.4. Additional Functions

- *read_basket_data()*, which takes the file name to be loaded as argument. Here, data from the file "data.txt" which contains shopping basket data is loaded.
- *print_rule()*, which takes four arguments: names (of the items), left and right (sides of association rules) and confidence. The function will use the arguments to print each rule in the format X -> Y.
- *check_item()*, which takes two arguments, a newly generated item set and an array of all the already generated itemsets and then checks the new item set against the list of all existing item sets to see if it should be added to the list or not.
- *check_rule()*, which takes four arguments: two single item sets (that would make up a rule, l and r) and the left-and (lhs) right-hand (rhs) side of the existing rules. The function essentially takes an item set (l) and checks if this item set is already existing within the rules. If so, it will check, if the other part of the rule (r) is matching the same rule on the right-hand side. The function returns a Boolean.
- *calculate_support()*, which takes two arguments, some item set and all of the item sets from the basket data set. It will calculate the support of x in X, meaning it will return a value that indicates how often the item set x occurs in the entire data set X.
- *calculate_confidence()*, which takes three arguments, the left- and right-hand side of a rule and the entire dataset. It then calculates the confidence of any rule, by checking how many transactions in the dataset match the left-hand side of the rule and how many associations can be made from it (right-hand side), e.g. customers bought apples in 6 transactions and within these transactions customers also bought bananas 4 times. Hence, the confidence is calculated by dividing associations by transactions: 4/6 = 0.67.

3. Results

In this chapter, the effects of changing values for minimum support and minimum confidence as well as the resulting association rules will be briefly discussed.

3.1. Different values for minimum support and minimum confidence

Changing the value for minimum support will essentially determine how many frequent item sets the apriori algorithm will find. An increased minimum support level means that any item set that is considered to be a valid item set has to occur within the dataset more times than with lower minimum support. However, this is less likely, because it requires k1-itemsets to occur more often and in return to also have more complex item sets occur more frequently, but since there will be less k1-itemsets when increasing the minimum support level, there will also be less more complex item sets. This can be explained by the principle that if a simple item set is not frequent, any more complex item set that includes the non-frequent simple item set can also not be frequent.

In practice, this can be seen in the following examples

Min_support	Item sets generated
0.015	149
0.05	23
0.1	6
0.2	2
0.3	0

As you can see, marginal changes in minimum support level can already have significant impacts on the number of generated item sets.

As for the confidence level, changing the minimum value will impact the number of rules that are generated from the frequent item sets. Obviously, the following rules apply:

Parameter inputs	**Outcome**
Low min_support & low min_confidence	Many item sets, Many rules
Low min_support & high min_confidence	Many item sets, few rules
High min_support & low min_confidence	Few item sets, few rules
High min_support & high min_confidence	Few item sets, no rules

In practice, the above can be observed when tweaking the parameters as previously done for minimum support, first for min_confidence = 0.6:

Min_support	**Item sets generated**	**Rules generated**
0.015	149	12
0.05	23	0
0.1	6	0
0.2	2	0
0.3	0	0

then for min_confidence = 0.2:

Min_support	**Item sets generated**	**Rules generated**
0.015	149	982
0.05	23	71
0.1	6	13
0.2	2	1
0.3	0	0

The lower the confidence level, the more rules are generated. However, lowering the confidence level is not always good, because it can lead to rules that do not make much sense.

3.2. Analysis of association rules

The below association rules were drawn from the apriori algorithm with

- min_support = 0.015
- min_confidence = 0.6

These settings provide both a decent number of frequent item sets as well as rules that provide sufficient confidence to be considered valid.

```
ham -> whole milk with confidence: [ 0.66666667]

sliced cheese -> rolls/buns with confidence: [ 0.72727273]

bottled beer, other vegetables -> whole milk with confidence: [ 0.625]

brown bread, whole milk -> yogurt with confidence: [ 0.625]

citrus fruit, rolls/buns -> whole milk with confidence: [ 0.7]

curd, yogurt -> whole milk with confidence: [ 0.7]

frozen vegetables, other vegetables -> whole milk with confidence: [ 0.625]

margarine, other vegetables -> whole milk with confidence: [ 0.66666667]

margarine, rolls/buns -> whole milk with confidence: [ 0.88888889]

rolls/buns, sliced cheese -> sausage with confidence: [ 0.625]

rolls/buns, sugar -> whole milk with confidence: [ 0.625]

tropical fruit, yogurt -> whole milk with confidence: [ 0.66666667]
```

Some conclusions:

- Whole milk is bought quite frequently
- Some rules clearly indicate that customers intend to use items together, e.g. *sliced cheese* -> *rolls/buns* or *rolls/buns, sliced cheese* -> *sausage*. These items could be used for breakfast or some other meal.
- There are no really surprising or weird rules that do not really make sense.
- There are no rules that associate an item with itself, e.g. yoghurt -> yoghurt.