Machine Learning in Data Science Project Week4

Eva Aßmann, Paul Vogler

November 2019

1 Introduction

The Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The Apriori algorithm has been designed to operate on databases containing transactions, such as purchases by customers of a store. An item set is considered as "frequent" if it meets a user-specified support threshold. Support is an indication of how frequently the item set appears in the data set and is defined as the proportion of transactions from the item set in the data set. Finding all frequent item sets in a database is difficult since it involves searching all possible item sets (item combinations). Although the size of the power-set grows exponentially in the number of items k, efficient search is possible using the downward-closure property of support (also called anti-monotonicity or Apriori rule) which guarantees that for a frequent item set, all its subsets are also frequent and thus no infrequent item set can be a subset of a frequent item set.
The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in a database. Association rules are required to satisfy a minimum confidence constraint. Confidence indicates how often a rules has been found to be true and can be interpreted as the conditional probability of one frequent item set following another frequent item set.
Tree-Based Association Rule (TBAR) algorithms: In the first pass, the algorithm counts the occurrences of items (attribute-value pairs) in the transaction data set, and stores these counts in a 'header table'. In the second pass, it builds the Frequent-Pattern-tree (FP-tree) structure by inserting transactions into a trie. Items in each transaction have to be sorted by descending order of their frequency in the data set before being inserted so that the tree can be processed quickly. Items in each transaction that do not meet the minimum support requirement are discarded. If many transactions share most frequent items, the FP-tree provides high compression close to tree root. Recursive processing of this compressed version of the main data set grows frequent item sets directly, instead of generating candidate items and testing them against the entire database as in the Apriori algorithm.
Growth begins from the bottom of the header table i.e. the item with the smallest support by finding all sorted transactions that end in that item. Call this item **I**. A new conditional tree is created which is the original FP-tree projected onto **I**. The supports of all nodes in the projected tree are re-counted with each node getting the sum of its children counts. Nodes (and hence subtrees) that do not meet the minimum support are pruned. Recursive growth ends when no individual items conditional on **I** meet the minimum support threshold. The resulting paths from root to **I** will be frequent item sets. After this step, processing continues with the next least-supported header item of the original FP-tree. Once the recursive process has completed, all frequent item sets will have been found, and association rule creation begins.

2 Description of the data

The Online Retail Data Set (http://archive.ics.uci.edu/ml/datasets/Online+Retail) downloaded from the UCI Machine Learning Repository is a transnational data set which contained all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. The company mainly sold unique all-occasion gifts. Many customers of the company were wholesalers. The data set

comprised 541.909 transaction, each assigned 8 attributes describing id, date, time and price of a transaction, id, description and quantity of the purchased items, as well as customer id and country of residence (see Table 1). Transaction ids (InvoiceNo) starting with the letter 'A" labelled 'Adjust bad debt' transactions (and StockCode B), ids starting with the letter 'C' labelled cancelled transactions. There were StockCode values deviating from the predefined format that still labelled products. The Description field contained nan entries, upper case and lower case values. Mosly, upper case values denoted product names and lower case values denoted various and inconsistently formatted values on issues with the product or transaction process. However, there were intersecting values between upper and lower case descriptions. The Quantity field contained values ranging from 0 to 80.995 and negative values from -80995 to 0. This was processed in the Preprocessing section. The UnitPrice field contained positive values ranging from to and two negative values ranging from -11062 to 0. The two transactions with negative unit prices showed an 'Adjust bad debt' label and nan customer ids. The CustomerID field contained nan entries.

Name	Description	Data type
InvoiceNo	Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction.	string
StockCode	Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.	string
Description	Product (item) name. Nominal.	string
Quantity	The quantities of each product (item) per transaction. Numeric.	int
InvoiceDate	Invice Date and time. Numeric, the day and time when each transaction was generated.	string
UnitPrice	Unit price. Numeric, Product price per unit in sterling.	float
CustomerID	Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.	float
Country	Country name. Nominal, the name of the country where each customer resides.	string

Table 1: Schema of Online Retail Data Set.

3 Preprocessing steps

Data Cleaning. Since a association rule analysis requires transactions with known product labels, all transactions containing nan Description values were dropped. All transactions labelled as cancelled were removed as well. The transactions with negative Quantity values that remained in the data set up to this point all showed lower case Description values. Since the lower case Description values were not of good quality and did not seem to qualify for an informative association rule analysis, these transactions were removed from the data set. The two transactions with negative unit prices were affected by this removal. After missing values in the Description field, cancelled transactions, negative quantities and negative unit prices were handled, there remained multiple entries with missing customer ids. Since customer ids are not relevant in the process of association rule analysis, missing values were replaced by "no_data". After the cleaning process, 530.693 transactions were still in the data set.
Preprocessing. Transactions done in France were extracted only considering relevant attributes, which were transaction id (InvoiceNo), purchased item(s) (Description) and quantity of each item (Quantity). The resulting french transaction table comprised 461 transactions as rows and 1564 items as columns. The french transaction table was binary encoded, such that each row stored one values (1) for every item that was purchased and zero values (0) for all items that were not purchased within the respective transaction.

4 Methods

The binary encoded french transaction table was given as input to the 'apriori' function from the mlxtent library. The function processes the most frequent item-sets from the columns with value 1, and returns all

sets with their support, where the support is larger than the given 0.07%, as a dataframe. The resulting frequent item dataframe was processed by the 'association_rules' function from the same package, to find association rules from the frequent items, as well as calculating the support and confidence (alongside other metrics) for each of these rules. The resulting rule set was then filtered, to only consist of rules with a confidence larger than 80% and a lift larger than 6.
The TBAR algorithm was run by using a tree-based apriori algorithm on the transaction table. The transaction table needed to be transformed from being binary encoded, so that for each row in the transaction table all columns with 0 would be omitted and for all columns with value 1 the column number would be in that row. The transformed transaction table was given to the tree-based apriori as a space-separated '.csv' file. The function returned the frequent items with their support in a dictionary, which was adjusted to hand the output to the mlxtend 'association_rules' function again.

5 Results

The runtime for the apriori algorithm was 0.045 seconds for our dataset, which was about double the duration that the TBAR algorithm took, with a runtime of 0.022 seconds. Therefore considering runtime, the TBAR algorithm is preferable algorithm. Concerning the results from each run, both algorithms resulted in a list of 50 frequent itemsets, but apriori resulted in a lot of one, and a few two or more elements in each item set, while the run of the TBAR algorithm only returned frequent item sets with two or more elements, so the results differed strongly. Also the found itemsets that contain two or more items stronly differ between the two algorithms. The association rule function could find multiple rules for the apriori produced itemsets, but couldn't find any for the TBAR sets, probably because of the missing 1-item itemsets.

6 Discussion

The data set was poorly documented and contained many unexplained attribute values. The runtime advantage of the Tree based approach was expected, because of the easier to process data structure for multiple iterations through the dataset. It was not clear, why the two algorithms showed different results, but the pure apriori approach seems to be the correct one, because the frequent itemsets contain the 1-item itemsets.