James Taylor

Exercise 3

Data 630 Spring 2020

**1) Introduction**

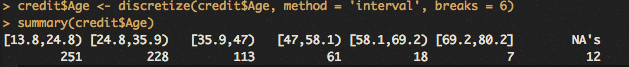
I expect the Apriori method to show patterns/correlations that are associated with an application being approved for a credit card. Conversely, the method would also show correlations with being denied a credit card. This could be used to quickly access if yourself have a good or bad chance at approval, or there is obvious prejudice in their approval system.

**2) Data Pre-Processing**

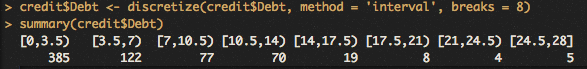
The Apriori method requires all variables to be discrete or factors. The provided Credit Approval dataset has numerical values that need to be discretized for this reason. The first step of pre-processing, however is removing a unnecessary variable Key which is just an numerical identifier. The code is below.

../../Desktop/1.png

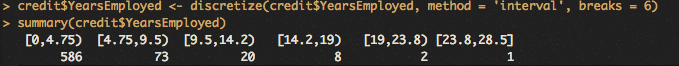
There are six numerical variables that need discretization. The below figures are the process to do that for each of these variables.



Above is the command that discretized the Age variable. The output shows the distribution across the bins and their width.



The above figure is a command that discretized the Debt variable. The assumption is the value is in the thousands of dollars, and the applications tend to have little debt.



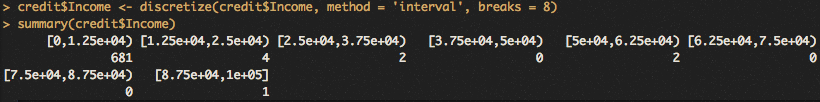
The command above is the discretization of the Years Employed variable. There were 6 bins created because the distribution of the values is very centered on low values with skew towards larger values.

../../Desktop/5.png

The command used for the Credit Score created an ordered factor. This was done because there were only 23 unique values and a very important determinant in someone’s interests rates and ability to handle credit, so the distribution was better preserved without discretization.

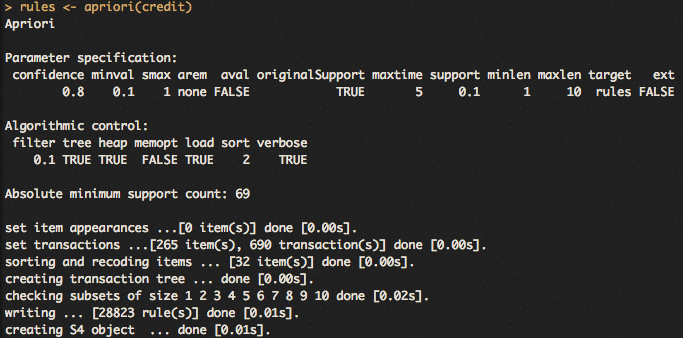
../../Desktop/6.png

The above command converts the numerical ZIP Codes variable to factors. This variable doesn’t make to place into bins because they are identifiers and their magnitude has no meaning.

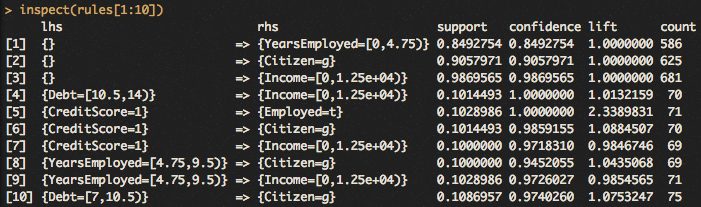


Income was discretized into eight separate bins. The distribution is heavily skewed to the right. All the discretization is to allow for the Apriori method package to work properly.

**3) Generate rules in variable called ‘rules’**

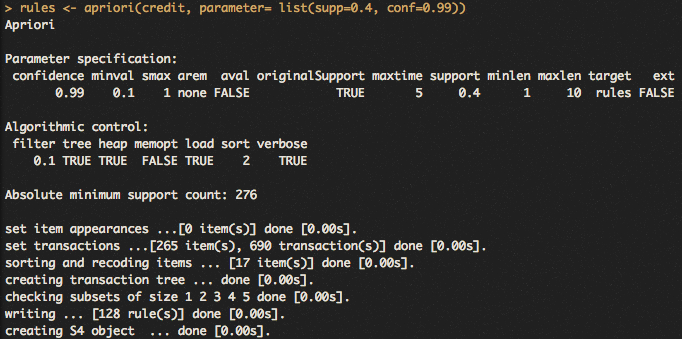
****

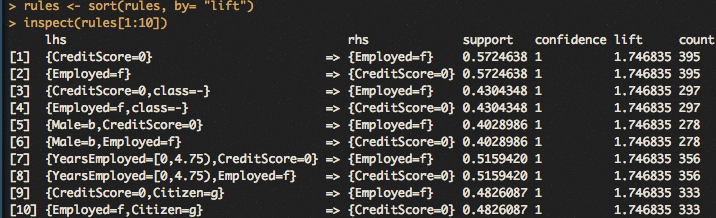
The above is the output from the command which is a Apriori method approach to the dataset. There are specifications listed there which show threshold on when rules are omitted. Above, its shown this algorithm omits rules with confidence below .8 and support below 0.1. The method ran through 690 observations and wrote 28,823 rules.



The above output shows the first 10 rules of this algorithm. The ‘lhs’ (left hand side) and ‘rhs’ (right hand side) columns is analogous to an IF-THEN rule. In this dataset, it is generally true that given the left-hand statement, the right-hand statement is true. The support column shows what percent of observations in the dataset that meets both conditions. Confidence is the accuracy of the IF-THEN statement. Lift is a measure to detect independence between the two statements. If the lift is close to one, they are more independent. The more the lift value is the more dependent they are, or a stronger relationship between the two. The count is the number of observations the conditions occur in the dataset.

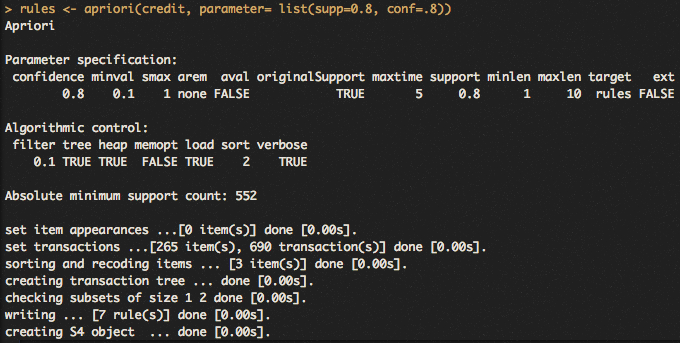
**4) Run method with different combinations**

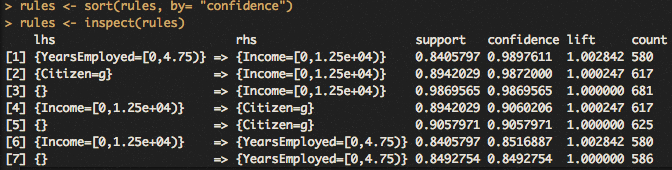
****

****

The above figures show another Apriori method but with different parameters for support and confidence. The confidence threshold was increased to .99 and the support to .4. This reduced the amount of rules written to only 128, compared to 28,823 with the default parameters.

The lower figure shows the rules sorted by lift value. This can loosely be considered the strength of relationship. It is unlikely to be randomness that associates on with the other.



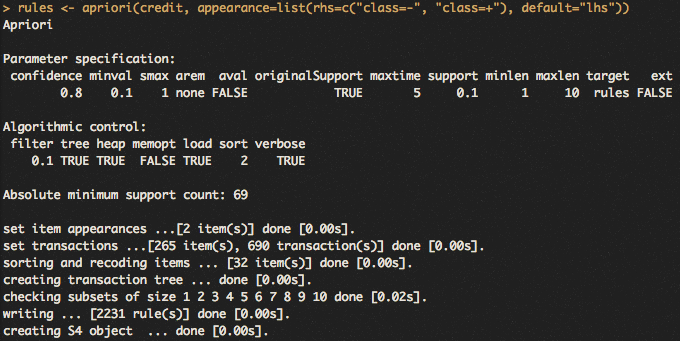


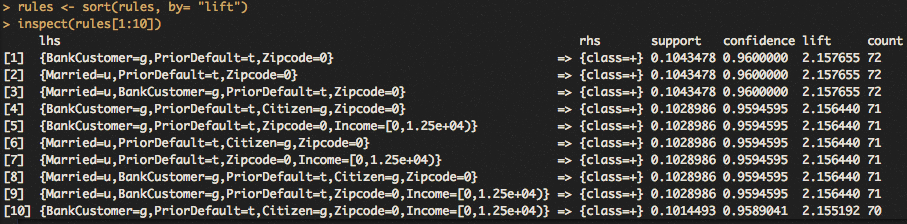
Above is another Apriori method with again different parameters, where rules with confidence below .8 and support below .8 were omitted. These specifications allowed for seven rules, which are sorted by confidence above.

Specifying the threshold levels of confidence, support and minimum length alters which rules get omitted. The higher the standards one specifies the rules to be, the less rules will be outputted.

The strongest rule depends on ones definition of strong. A high confidence means this rule will always be correct. This may not be helpful at all however because two variables in the dataset may have the same value for all observations. That rule will have 100% confidence and support but it’s not interesting or useful. A high lift value means that meeting both conditional statements are dependent and not random. This definition of strength is about the relationship between the two statements, and how likely they are to not be a random association. For example, take these two conditions; being very tall (7ft), and having very large shoes. Each one is very rare to occur on their own, but very likely to occur together. A very tall person certainly not have small feet, making the lift value high for this kind of relationship.

**5) Generate rules with class = +/-**

****

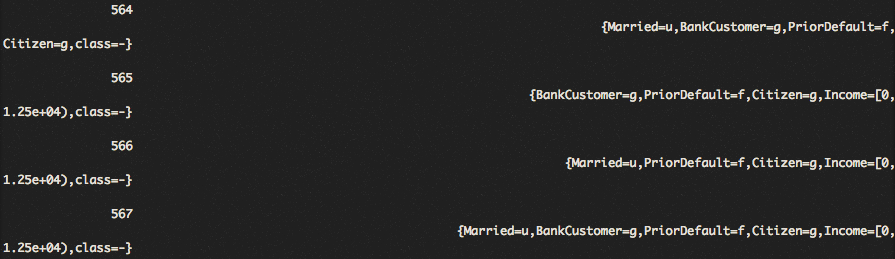
****

The above commands and outputs made a specification to have the right hand right-hand side (rhs) be either a ‘+’ or ‘-‘ for this Apriori method. It returned 2,231 rules. The rules were sorted

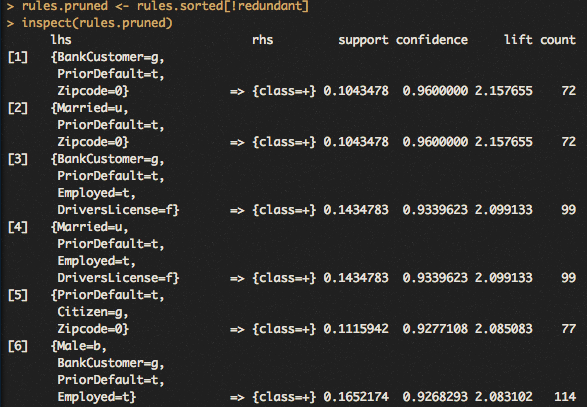
so the strongest were at the top. The left-hand sides are more specific with three or four conditions. Many of the top 10 strongest rules have prior default as ‘t’ which assumed to be true. Assuming the ‘+’ value in class means they were approved for credit, this is unexpected. Having prior defaults shows a history of delinquency. The working explanation is either ‘t’ for prior defaults doesn’t mean true, or ‘+’ for class doesn’t mean approved for credit.

**6) Prune returned values**

Pruning removes the redundant rules (for example a->b is redundant with b->a).



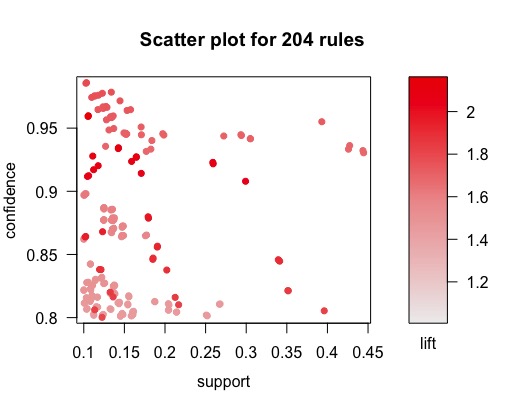
Above is a sample of the output for the which(redundant) command which produces many lines of output. These rules are the redundant rules that will be removed.



The commands and output shows a sample of the outputted unique rules. The ‘!’ character gets all the rules that are not redundant.

**7) Rules visualization**

**../../Desktop/19.png**

****

The above is a plot of rules where confidence, support and lift are displayed for each. It shows how the different definitions of strength of a rule. There are trade-offs with each strength, sacrificing strength in other categories.

**8) Summary**

More than one strength metric matters because they all matter. If one is chosen as a priority over the others, they sacrifice the strength of that rule in the other categories. A good rule would be strong in all categories.

The most challenging part is understanding the redundant cleaning of the rules. Getting rules to compare them to other rules, changing all below the diagonal as NA, this technique was new to me. It took some time reading through the steps to figure out what each step as doing.