|  |
| --- |
| Siemens Wind Power Inc. |
| 2017 Wind Analytics Contest |
| University of Central Florida, Statistics Dept. |
| Team ID: 6 |
| Team Members:   * Zachary Merritt – zmerritt@knights.ucf.edu * Paul Sarduy – psarduy@knights.ucf.edu |

|  |
| --- |
| Supporting Faculty Member: Dr. Mark Johnson  3-12-2017 |

Table of Contents

[Executive Summary 2](#_Toc477042281)

[Data Analysis 2](#_Toc477042282)

[Modeling Approach 4](#_Toc477042283)

[Associations 4](#_Toc477042284)

[Clickstreams 5](#_Toc477042285)

[Clustering Codestreams 7](#_Toc477042286)

[Predicting Visit Duration 7](#_Toc477042287)

[Results 7](#_Toc477042288)

[Conclusion 8](#_Toc477042289)

# Executive Summary

The main tasks of this project were to examine sequences of codes generated by wind turbines to find patterns, and to use the provided information to predict possible failures and duration of technician visits. To that end, we found several association rules that show associations between different codes. Most of these rules had high confidence and lift, but low support, so they only apply to a small subset of the data.

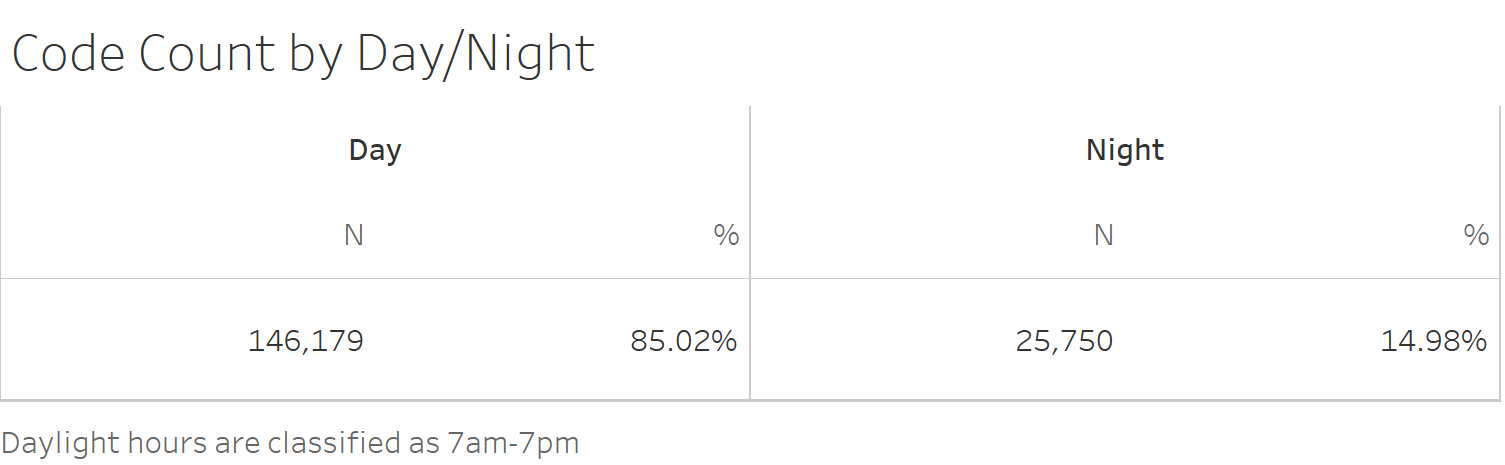
Next, we approached the code sequences from a web analytics standpoint, restructuring them as clickstreams. From that we could cluster the code sequences into two clusters, which we can use to predict turbine failure.

Lastly, we attempted a variety of regression methods to predict visit duration, first using factors A through D from the data set, then using code frequencies. The predictive power was not satisfactory, so instead we focused on ranking the most influential codes that contribute to visit duration based on the information provided.

# Exploratory Data Analysis

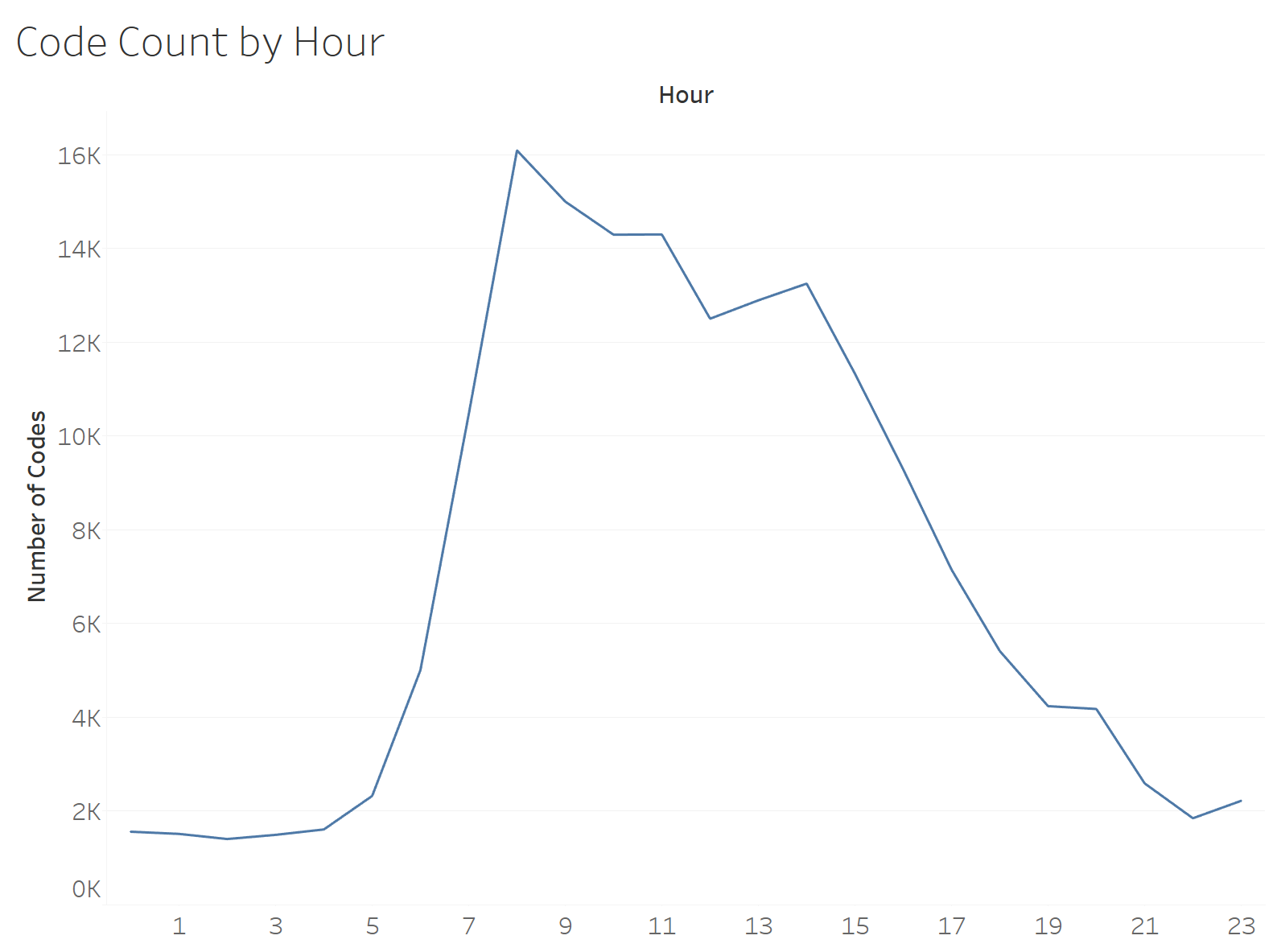
The main bulk of the information in this dataset is contained in the turbine codes. Therefore, we are mainly interested in exploring the distribution of codes under various conditions.

The first question to consider is when these codes are generated, based on TimeOn.



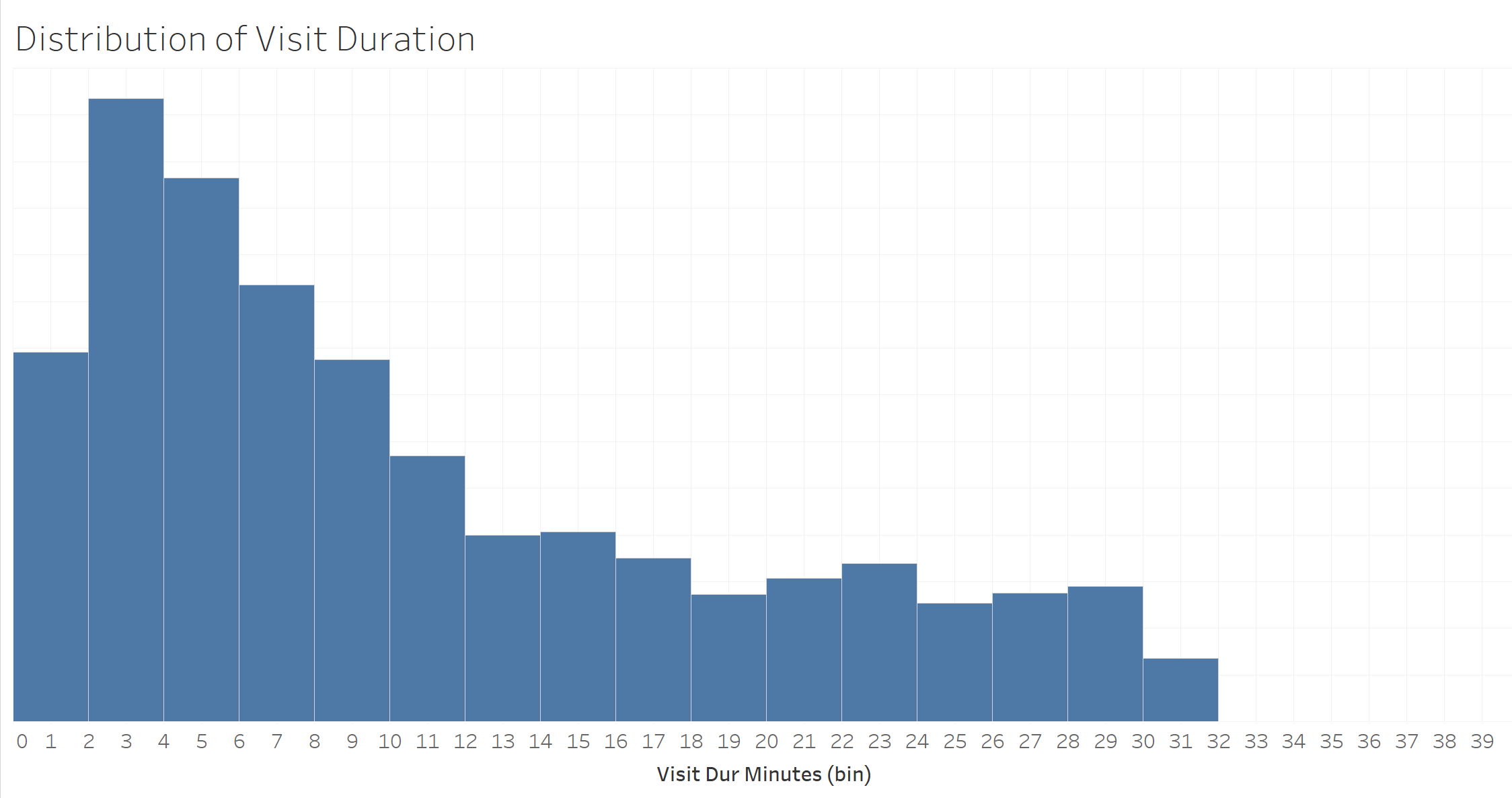
We can see that about 85% of the codes were generated during the day, which is likely a relief to the technicians.

If we want a more fine-grained breakdown of code generation time, we can also look at it by hour.



The code count hits its peak around 8am and its nadir at 2am. It appears to be approximately normally distributed.

One of the central tasks in our analysis concerns predicting visit duration, so it is useful to know the distribution of code counts by visit duration.



The distribution is skewed to the right, so a log transformation may be appropriate to coerce visit duration to be closer to a normal distribution.

The codes are not the only information in the data set. We also have Factors A through D, so we can look at the average visit duration by each level of these factors.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

If we were to use these factors to predict visit duration, factors B and D would likely not be useful, since there is so little difference in average visit duration between the levels.

# Modeling Approach

## Associations

We can perform an association analysis, more colloquially known as a market basket analysis, of the codes generated. In this case, each “basket” is a visit ID, and the codes generated correspond to the items that a customer bought.

This type of analysis lets us discover which codes are associated with each other, e.g. if a visit contains code A and B it is highly likely that it will also contain code C. Association analysis is a useful starting point for us, but we can take it further.

## Clickstreams

One of the main differences between the wind turbine code logs and market basket analysis is that in association rules any ordering data is lost. We examine combinations of codes, but we should look at permutations to preserve any information encoded in code order. Thus, association analysis may be unsuitable for this application.

However, a more apt analogy can be found in web analytics. Consider a user on a website for an online store like Amazon. Every element on the site that the user clicks on is recorded, and the series of clicks will lead to one of two outcomes: buy something or leave the site. So, for each user session, a queue of clicks called a clickstream is captured and capped with one of our available outcomes. One can view this as a stochastic process where each click is a state, which eventually ends in some absorbing state. This process can be modeled as a Markov chain, which can then be used for several purposes, including prediction and clustering.

In fact, the choice of end states can be selected based on the variable of interest. Since the absorbing states should be finite, any continuous variable that one wishes to measure should be binned.

In our application, each user session is a visit ID, like the association analysis, and the sequence of codes, or codestream, is analogous to a clickstream. This lets us fit a Markov chain that encapsulates the information in the codestreams. Selection of absorbing states is not quite as straightforward as Buy or Leave in the website example above.

We decide to set the stop urgency related to the last code in the visit as the absorbing state. This should let us predict stop urgency based on observed code sequences. The structure of the chain is simple: each vertex is a code, and directed edge weights between vertices A and B are the probability of seeing code B immediately after code A.

## Clustering Codestreams

In many situations, it may be more useful to know what types of codestreams are related to each other rather than predicting the next code in a sequence. We can cluster the clickstreams via the k-means clustering algorithm, but we need to specify the number of clusters. That will depend mainly on the probabilities of ending in each absorbing state.

## Predicting Visit Duration

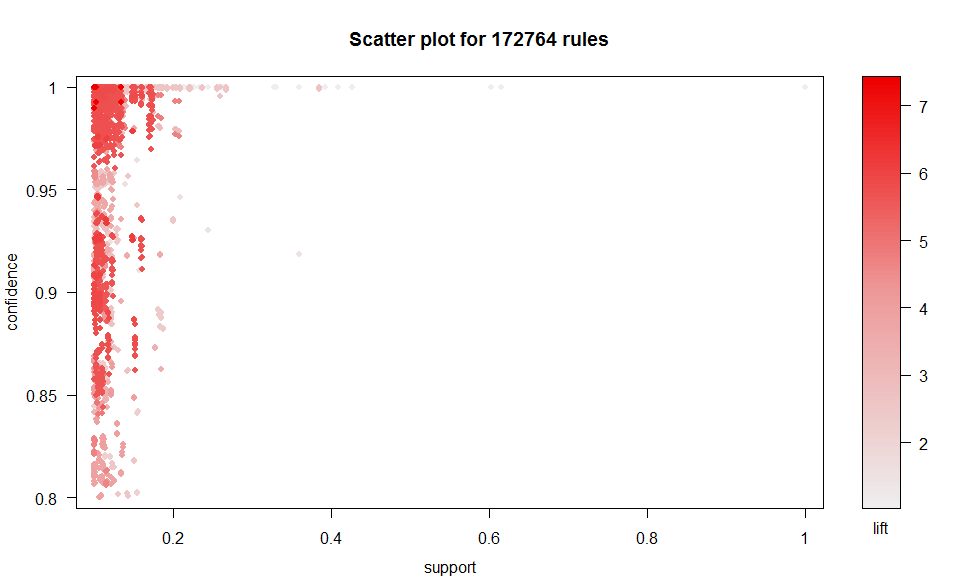
We tried several different structures for predicting visit duration. At first, we tried a structure using the given non-code factors and a few engineered variables. We soon found out that the factors given are not very good predictors of visit duration by themselves. In fact, you are probably better off assuming the average visit duration. The next analysis we did was using the codes as predictors of visit duration. We used the frequency matrix from the clickstream analysis (where each code is a variable with the count of the codes frequency and the rows are a single visit ID). The sparse code matrix performed better than the factor data set by a significant margin, but it still only had an adjusted R-squared of 0.24 in the best model. A combined data set of the factors and the code matrix was explored, but did not produce any significant variables not found in the code matrix data set.

We employed several different modeling techniques on our dataset. At first, we started with a simple ordinary least squares linear regression model. Various variable selection and dimension reduction techniques were applied, but the model never performed better than an adjusted R-squared of 0.23. Graduating from the simple regression, we also tried to capture more variance by introducing polynomial degrees to the predictors. Overall, the adjusted R-square improved very slightly (about a 0.002 difference), but not enough to warrant their inclusion. Finishing our regression attempts on the visit duration were several neural networks that were trained using both regression and classification nodes. Overall the neural networks simply overfit the data, no matter how many transformations were applied to the variables. Overall, the linear regression model seemed to fit the best.  
We had one more trick up our sleeve: we decided to try binning the visit duration in five minute increments to turn our regression problem into a classification one. Our bins were 0-5, 6-10, 11-15, 15-20, 25-30, and 30+ minutes. This is an ordinal categorical response. Again, we fit several regression models, a few CART decision trees, and a neural network. The model we ended up choosing from this analysis was the full simple linear regression model. This was not based so much on the predictive power, but on the ability to rank the codes by significance.

# Results

## Associations

After restructuring the data as a list of many transactions, indexed by visit ID, we obtain over 170,000 different association rules.



We can see from the plot that most rules have very low support, since there are so many different codes. Even if the intersection of two codes shows up relatively often in the numerator, there are so many transactions in the denominator that support is driven down. Consequently, association rules with high lift and confidence are still good rules, but they will only apply to a relatively small subset of the data.

With that disclaimer, we can still look at the top association rules sorted by lift to see if the results are useful.

|  |
| --- |
| > inspect(sort(rules, by='lift')[1:20])  lhs rhs support confidence lift  [1] {1023,13} => {14} 0.1019208 0.9923664 7.3949  [2] {1020,1023,13} => {14} 0.1019208 0.9923664 7.3949  [3] {14} => {13} 0.1341957 1.0000000 7.3942  [4] {13} => {14} 0.1341957 0.9922705 7.3942  [5] {1023,14} => {13} 0.1019208 1.0000000 7.3942  [6] {1001,14} => {13} 0.1000915 1.0000000 7.3942  [7] {1020,14} => {13} 0.1341957 1.0000000 7.3942  [8] {1020,13} => {14} 0.1341957 0.9922705 7.3942  [9] {1020,1023,14} => {13} 0.1019208 1.0000000 7.3942  [10] {1001,1020,14} => {13} 0.1000915 1.0000000 7.3942  [11] {1001,13} => {14} 0.1000915 0.9896641 7.3747  [12] {1001,1020,13} => {14} 0.1000915 0.9896641 7.3747  [13] {5112,5122,63003} => {5113} 0.1047955 0.9468713 6.2094  [14] {5110,5112,5122,63003} => {5113} 0.1047955 0.9468713 6.2094  [15] {1018,5112,5122,63003} => {5113} 0.1047955 0.9468713 6.2094  [16] {1020,5112,5122,63003} => {5113} 0.1047955 0.9468713 6.2094  [17] {1018,5110,5112,5122,63003} => {5113} 0.1047955 0.9468713 6.2094  [18] {1020,5110,5112,5122,63003} => {5113} 0.1047955 0.9468713 6.2094  [19] {1018,1020,5112,5122,63003} => {5113} 0.1047955 0.9468713 6.2094  [20] {1018,1020,5110,5112,5122,63003} => {5113} 0.1047955 0.9468713 6.2094 |

To interpret the top rule, we will not see codes 1023 and 13 together very often due to the low support, but when we do, we will very likely see 14 as well due to the high confidence and lift.

We can also show a summary for the association rules.

|  |
| --- |
| > summary(rules)  set of 172764 rules  rule length distribution (lhs + rhs):sizes  1 2 3 4 5 6 7 8 9 10  1 105 1283 6381 17827 32393 41058 37429 24692 11595  Min. 1st Qu. Median Mean 3rd Qu. Max.  1.000 6.000 7.000 7.166 8.000 10.000  summary of quality measures:  support confidence lift  Min. :0.1001 Min. :0.8000 Min. :1.000  1st Qu.:0.1026 1st Qu.:0.9746 1st Qu.:2.597  Median :0.1057 Median :0.9987 Median :3.045  Mean :0.1095 Mean :0.9777 Mean :3.792  3rd Qu.:0.1122 3rd Qu.:1.0000 3rd Qu.:5.657  Max. :1.0000 Max. :1.0000 Max. :7.395 |

The summary tells the same story as the scatter plot above, namely that the mean confidence and lift are high, but mean support is much lower.

## Clickstreams

The Markov chain is a directed edge graph, where vertices are codes and edge weights are the probability of transitioning from one code to another. When we plot the entire graph, we get the following.

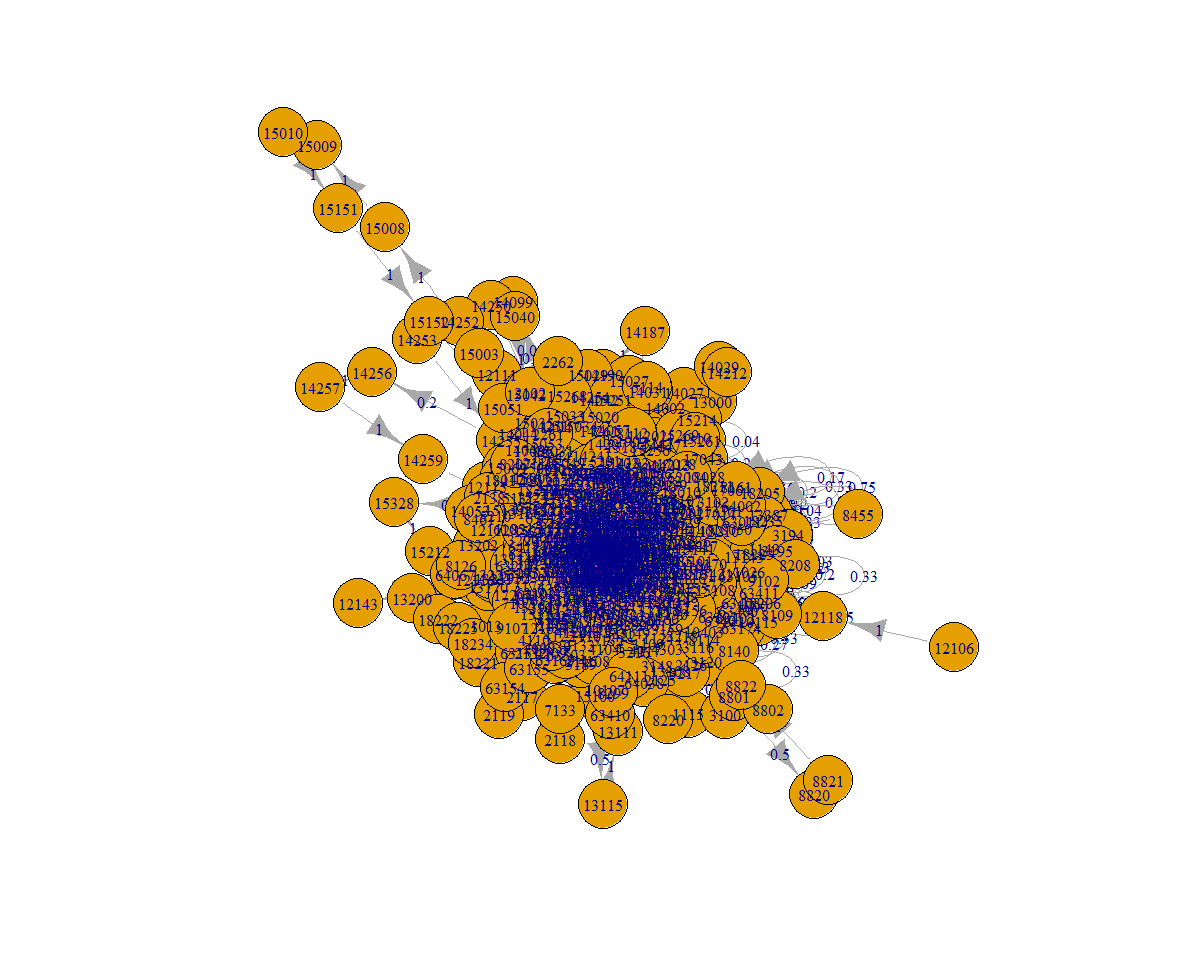


Figure 2: Visualization of full Markov chain for codestreams.

Obviously, the graph is too dense to be of much use. Luckily, we have some options to increase clarity. For starters, the graph above shows every single edge, even if the weight is miniscule. We can cut down on a good deal of the clutter by filtering out any edges with weight below a threshold, then removing vertices that become disconnected, i.e. have degree 0. We can also clean the graph up by removing the circle around vertex labels.

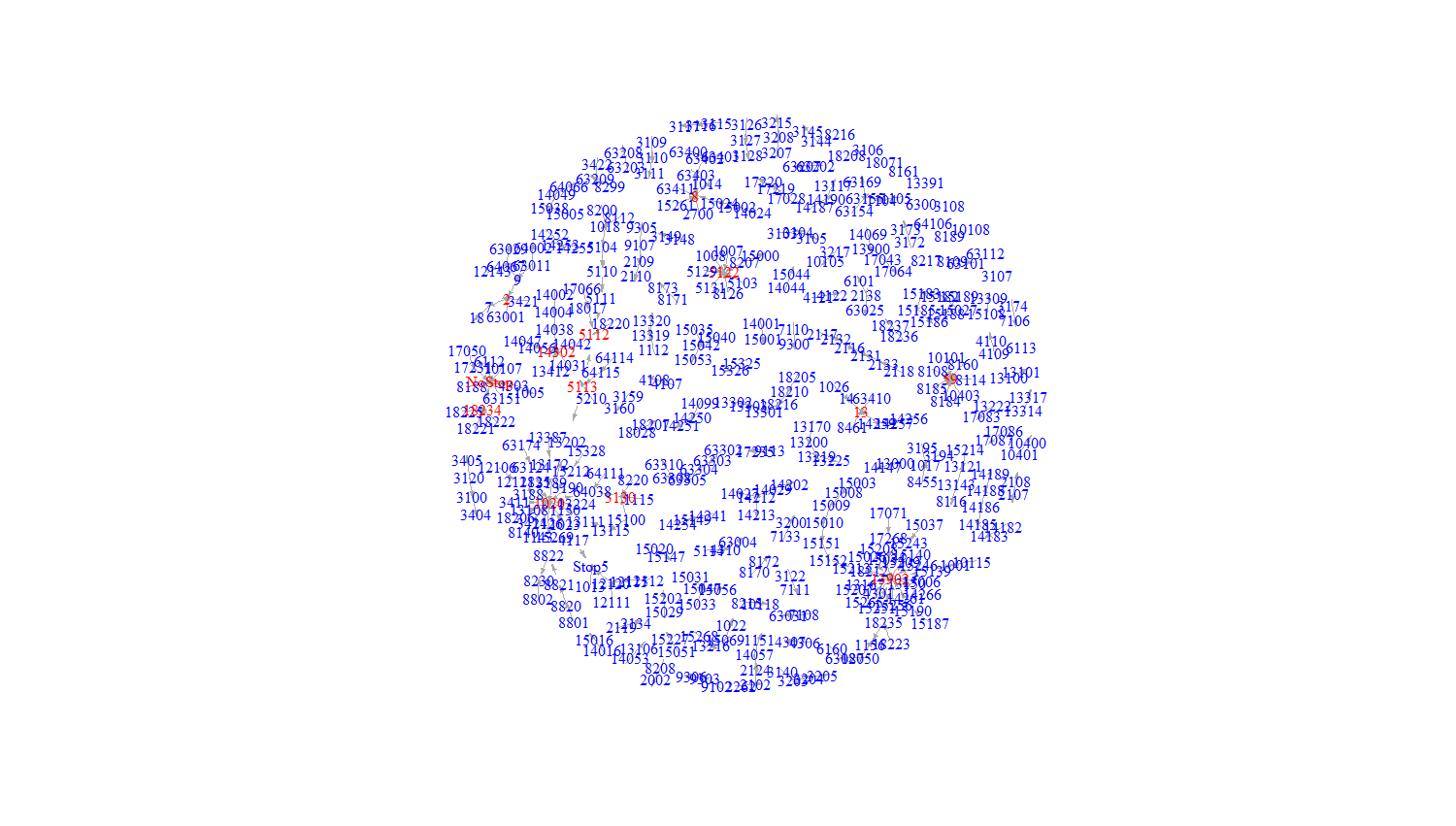


Figure 2: Markov chain with edge weights < 0.4 removed. Codes with degree at least 4 are highlighted in red.

From this visualization, it is easier to see that code sequences are arranged not in long chains, but in short associations, usually only two or three codes long. The results from the association analysis support this result. Most of the association rules have high confidence and lift, but low support, which implies strong associations that only apply to small portions of the data. We can also determine that certain codes, such as 13902 and 59, have several codes that lead to them, since they have a high in-degree.

This is bad news for using the Markov chain for predicting the next code in a sequence, unfortunately. Since most sequences with high edge weights are very short, predicting based on several codes in a sequence yields little more information than seeing only one or two codes. However, we can still glean useful information from the Markov chain by looking at the absorbing states.

|  |
| --- |
| End Probabilities:  NoStop Stop1 Stop2 Stop3 Stop4 Stop5  0.5180974781 0.0020906834 0.0047040376 0.0082320659 0.1142035803 0.3525414870 |

There were no sequences that ended with a stop of urgency 6. When we look at the probability of ending up in each of the absorbing states, we see that most codestreams end with stop urgency 0 or 5. In other words, a sequence either does not lead to a stop, or it leads to a very abrupt stop, with very little in between. This implies a few things:

1. Codestreams fall into one of two broad categories: no stop or sudden stop.
2. Having six levels of granularity for stop urgency may be unnecessary.

## Clustering Codestreams

As stated in the previous section, codestreams tend to end in either no stop or a rapid stop based on the probabilities of each absorbing state. Therefore, it makes sense to attempt to choose clusters for our k-means clustering algorithm.

After getting the two clusters, we can fit a Markov chain to each to see how predicted probabilities for absorbing states differ. We find that the first cluster leads to an urgent stop (StopUrgency = 5) with very high probability, and the second leads to no stop (StopUrgency = 0) with very high probability. This lets us loosely predict the outcome of any arbitrary code sequence as stop or no stop based on which cluster it would be assigned to.

We can test this capability using some example sequences from our data. Take the codestream with visit ID 100312. It consists of seven codes, ending with one that denotes a stop of urgency 5.

If we take the first portion, say the first five elements, of the code sequence and predict which cluster it falls into, we should get the cluster that corresponds to an urgent stop, namely cluster 1.

|  |
| --- |
| > click\_test <- clicks$`100312`  > click\_test.fragment <- new('Pattern', sequence = click\_test[1:5])  > predict(clusters, click\_test.fragment)  [1] 1 |

As a second example, consider the codestream corresponding to visit ID 1000863. This one has eight codes and ends without a stop, i.e. StopUrgency = 0.

Since this codestream is slightly longer, we take the first six codes and predict which cluster it goes in.

|  |
| --- |
| > click\_test2 <- clicks$`1000863`  > click\_test2.fragment <- new('Pattern', sequence = click\_test2[1:6])  > predict(clusters, click\_test2.fragment)  [1] 2 |

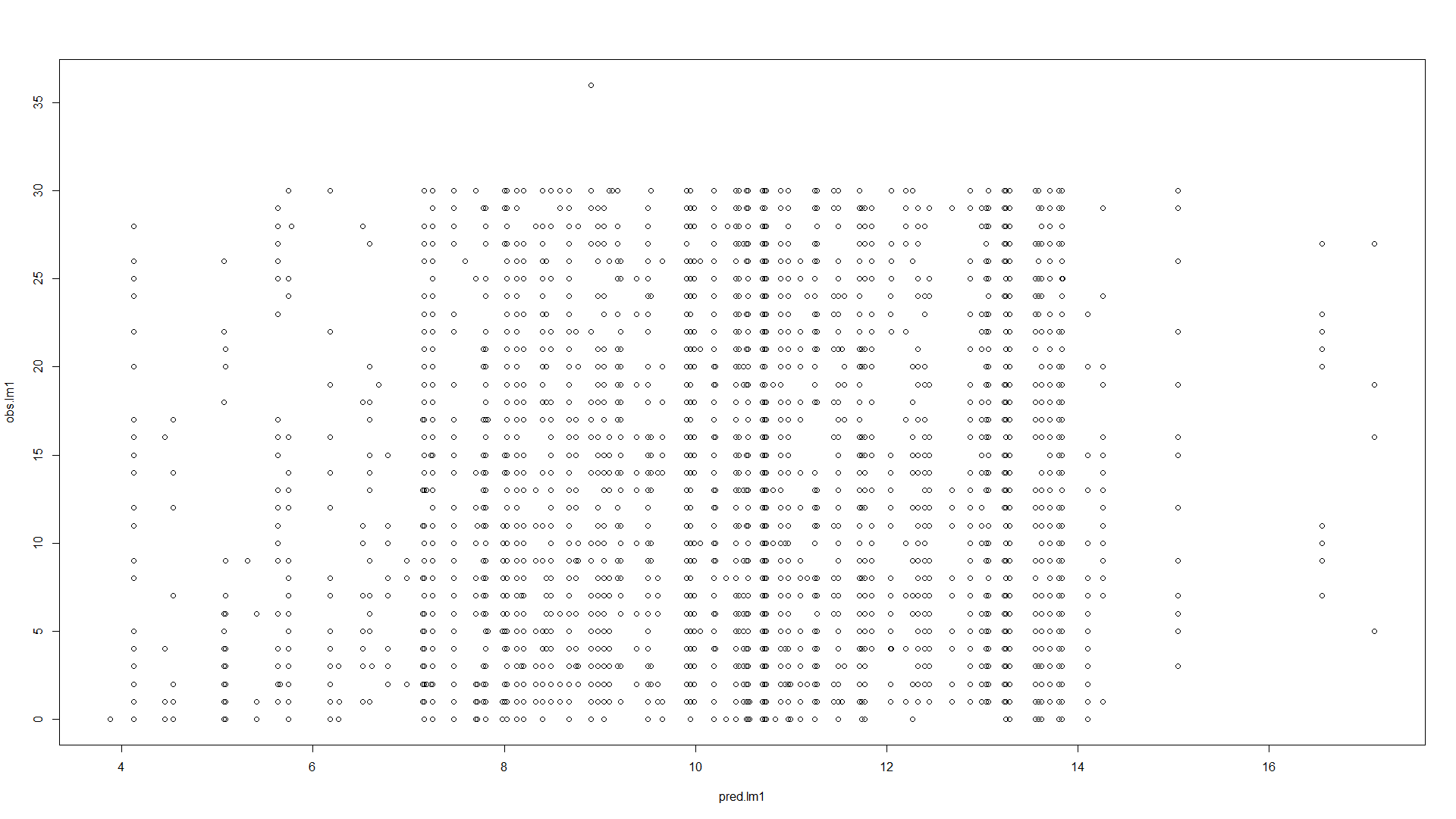
The partial stream is correctly classified in the second cluster, which corresponds to a high probability of ending with StopUrgency = 0.

One use case of these clusters could be for early failure detection. If a partial code sequence falls into cluster 1, it may be worth a visit from a technician before it fails. On the other hand, if it falls into cluster 2, it can probably be safely ignored for a while. This should be mentioned with the caveat that trying to place a codestream that has only a few codes into a cluster would likely yield untrustworthy results.

## Predicting Visit Duration

Before discussing the final model results, it is worth exploring the model with the original factors, just to show that they are not very good predictors of visit duration. Below is the model and a plot of the predicted response values versus the observed response values.

|  |
| --- |
| Model:  VisitDurMinutes ~ FactorA + FactorB + FactorC + FactorD + VisitType |

If the model was doing a perfect job at predicting the visit duration, we would expect to see a 45 degree line passing through the origin. We expect a good model to have a general trend along that line. As we can see above, that is about as far as you can get from the positive linear trend. This model does not do a good job at predicting visit duration. The adjusted R-square was 0.24.

As mentioned in the modeling approach section, our final model was chosen for its ability to rank the codes to find the most influential. Our final model was a full fit of all the code variables. We are not interested in their absolute estimates here, but more in their relative significance. Using our ranking package in R, we can find the variables with the highest variation of the response explained. Here we decided to pick the top 19 codes to report as most influential, that is because with these 19 codes alone we capture 70% of the variance explained by the full model. After that the return of adding a new code variable is heavily diminished. Below is a list of the top 19 most influential codes.

|  |
| --- |
| Most influential Codes:  1001  |  13294  |  1007  |  7103  |  10100  |  12120  |  10118  |  7106  |  1025  |  63029  |  1023  |  13311  |  13140  |  1015  |  9304  |  1024  |  3411  |  13106  |  7111 |

# 

# Conclusion

One of the tasks in the project was more feasible than the other, given the information provided. Visit duration is likely dependent on information that is not in the data set, perhaps something like technician skill. On the other hand, finding patterns in the code sequences was much more successful, and the association and clickstream analyses yielded information that may help Siemens wind turbine technicians be more efficient in their work.