Table of Contents

> **My name is Aditya A Prasad, my Stanford ID number is 006163737. My Stanford webmail is aditya42@stanford.edu.**

Introduction

The objective here is to train a binary classifier with 10 signals which are *in some way* properties of "the query" as well as "the web page". For example the first two signals are **query_length** (a property solely of the query) and **is_homepage** (a property solely of the webpage), likewise the remaining 8 signals - $sig1, sig2, \ldots sig8$ are some unknown function of **both** "the query" as well as "the web page".

The boolean property called **relevance** for all pairs - $[Webpage_i, \quad Query_j]$ which is $True$ (1) if the i^{th} webpage is relevant to the j^{th} query else it's $False$ (0).

The motivation for making this sort of a binary classifier can be seen when we consider the problem of ranking a huge number of webpages with respect to how relevant they are to a particular webpage - MLR.

Because it's extremely computationally expensive to run any sort of ranking algorithm on all the webpages, one way to solve this problem is given a query we select those webpages which are relevant to it. After which we can run the computationally expensive algorithm on those "*relevant webpages*".

Analysis of the predictors

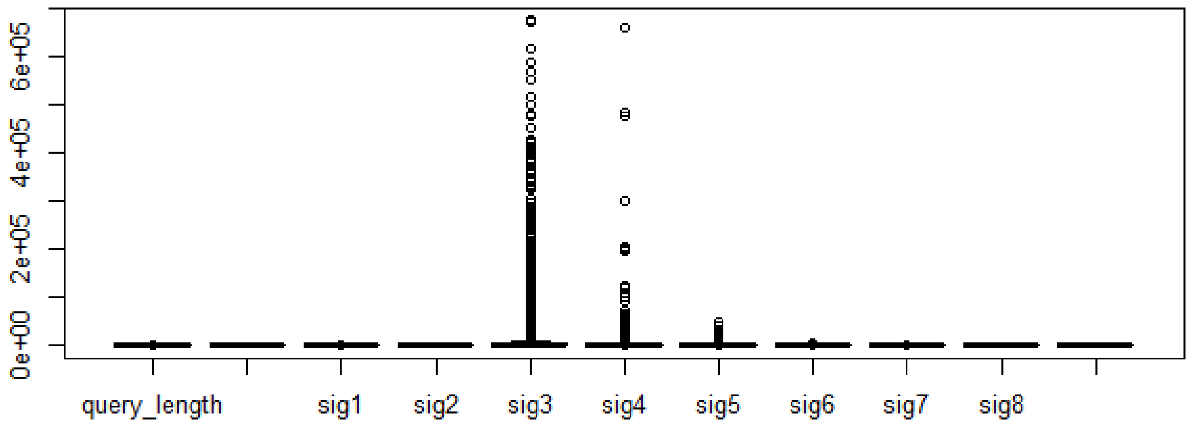
We know that none of the other signals are solely functions of query or web page because for any unique value of query_id/url_id we find that all other signals vary.

We can see that unlike what it might seem at first $query\_length$ doesn't seem to be the actual number of characters in the query because for most of the observations the value for $query\_length$ is 2 or 3 which makes it more likely that $query\_length$ represents the number of words or some other similar measure.

Note that I have converted the integer data types **is_homepage** and **relevance** into factor datatypes as the assumption of ordered values is false.

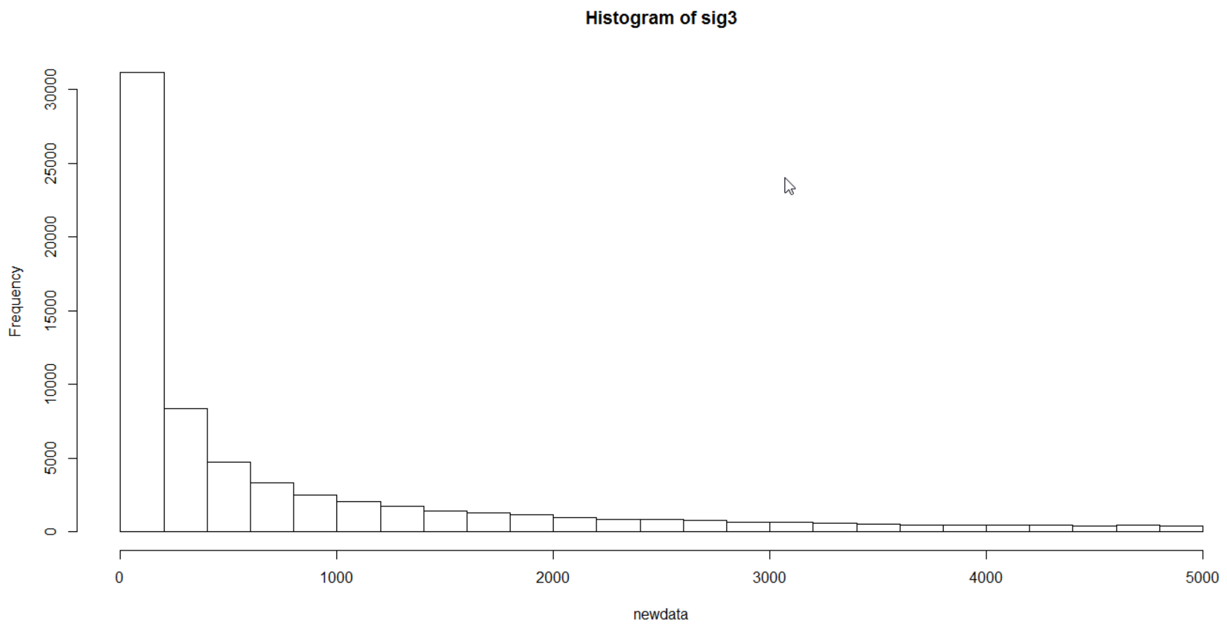
Coming to the output variable - $relevance$, 56.29% of observations are of queries which are relevant to that webpage. This means that we have enough observations of both classes so we can conclude that it's possible to train a pretty good binary classifier.

I saw that $sig3$, $sig5$, $sig4$ all had loads of "outliers", more than 10% of the values!

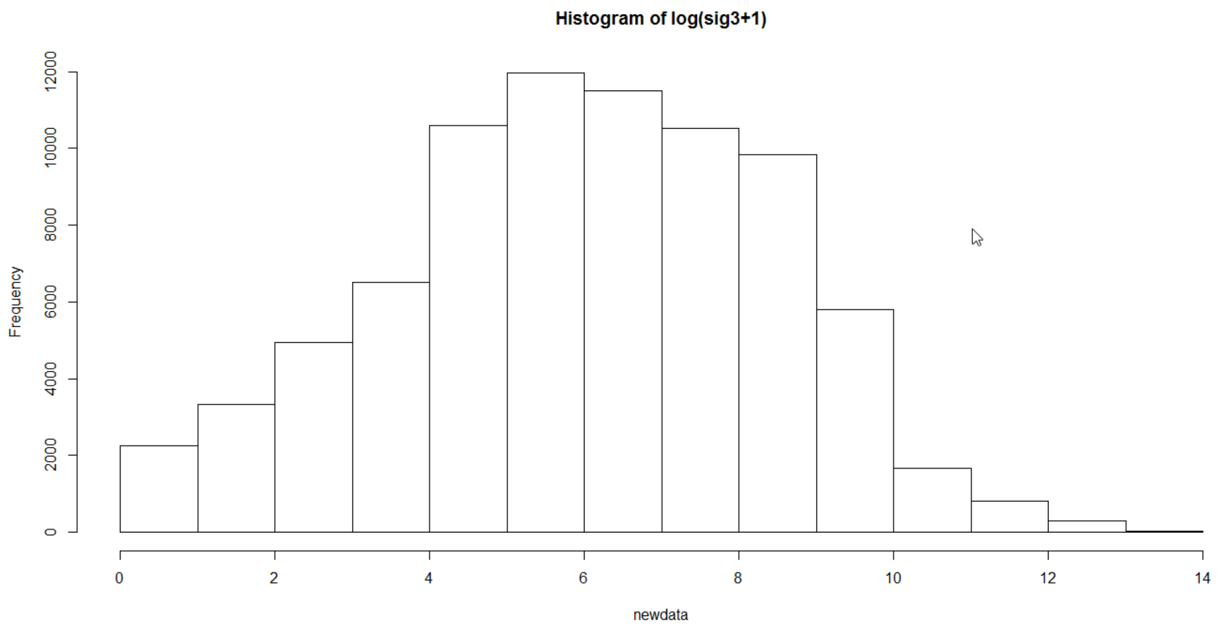


So when I plotted the distributions, I saw that all of them were highly skewed distributions. While the values were non-negative, we had zeros. I used the $x' = \log(x+1)$ transformation (which has the neat feature that 0 maps to 0 useful because $sig4$ and $sig5$ has a awful number of 0's). Another reason why I go with this transform is because the relationship is close to exponential.

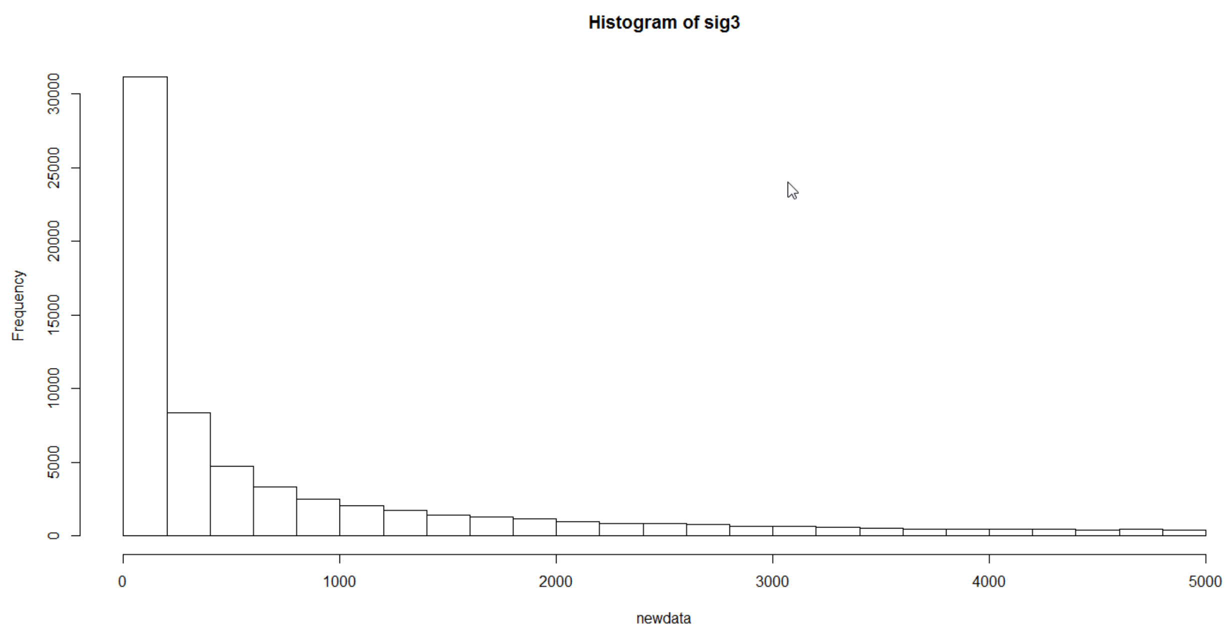
So for $sig3$ we initially had the skewed distribution shown below. I am showing elements below 5000 because otherwise the hist function has a single bucket for all numbers below 1000.



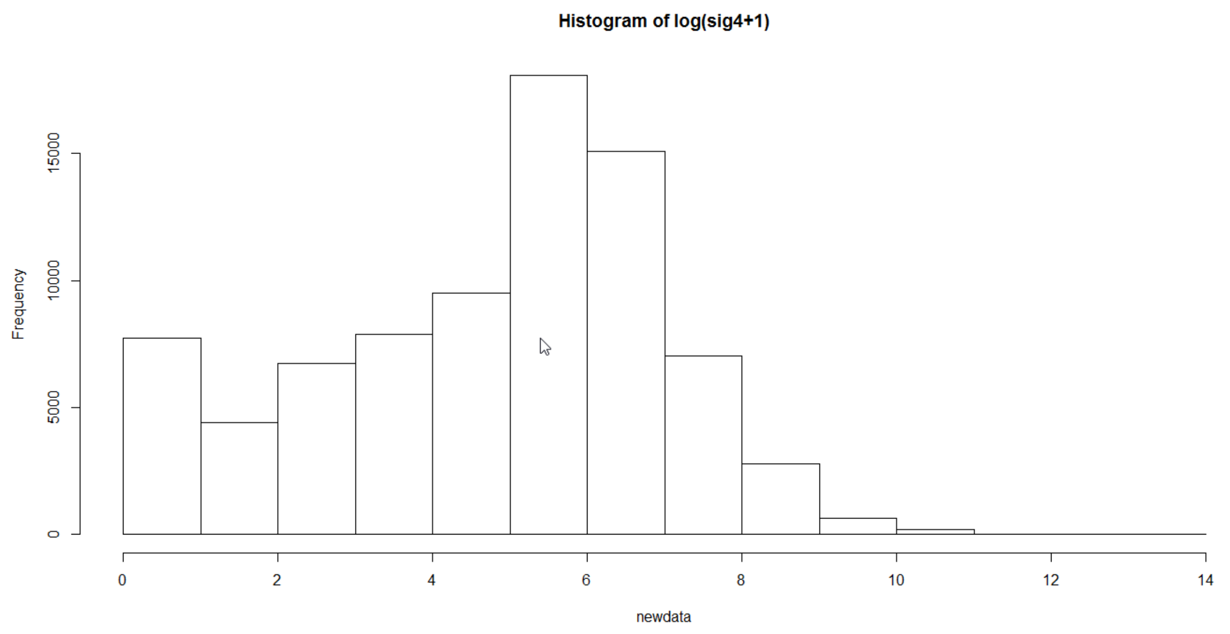
After log transforming it, I get a beautiful distribution.



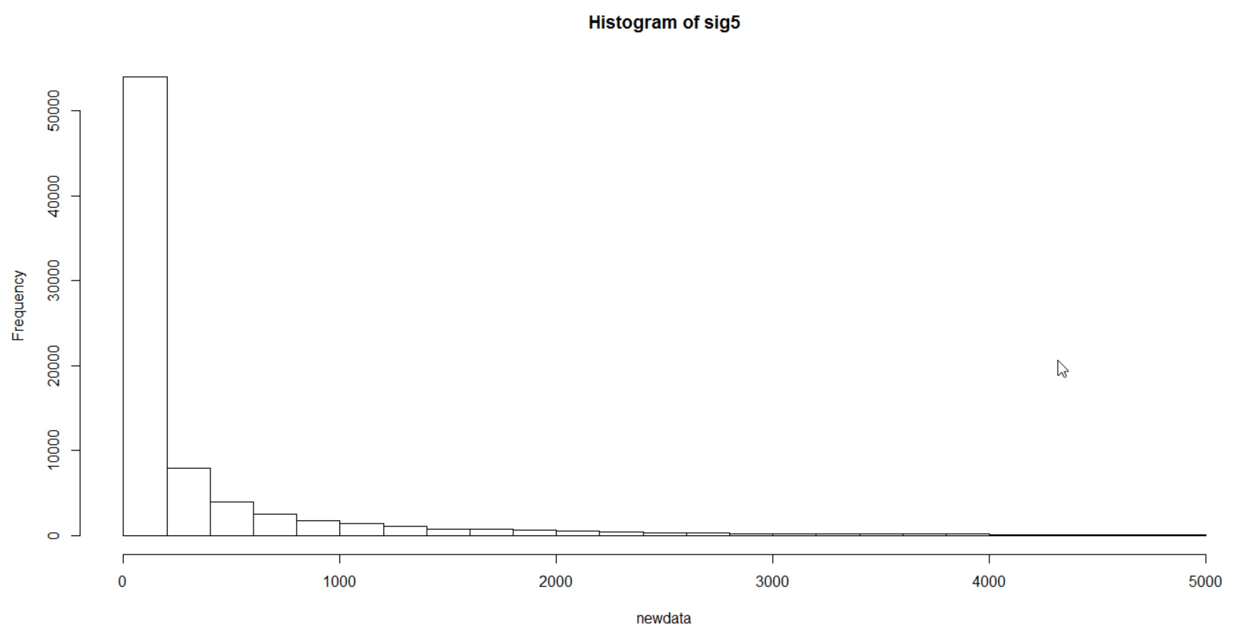
Similarly for the signals $sig4$ and $sig5$.



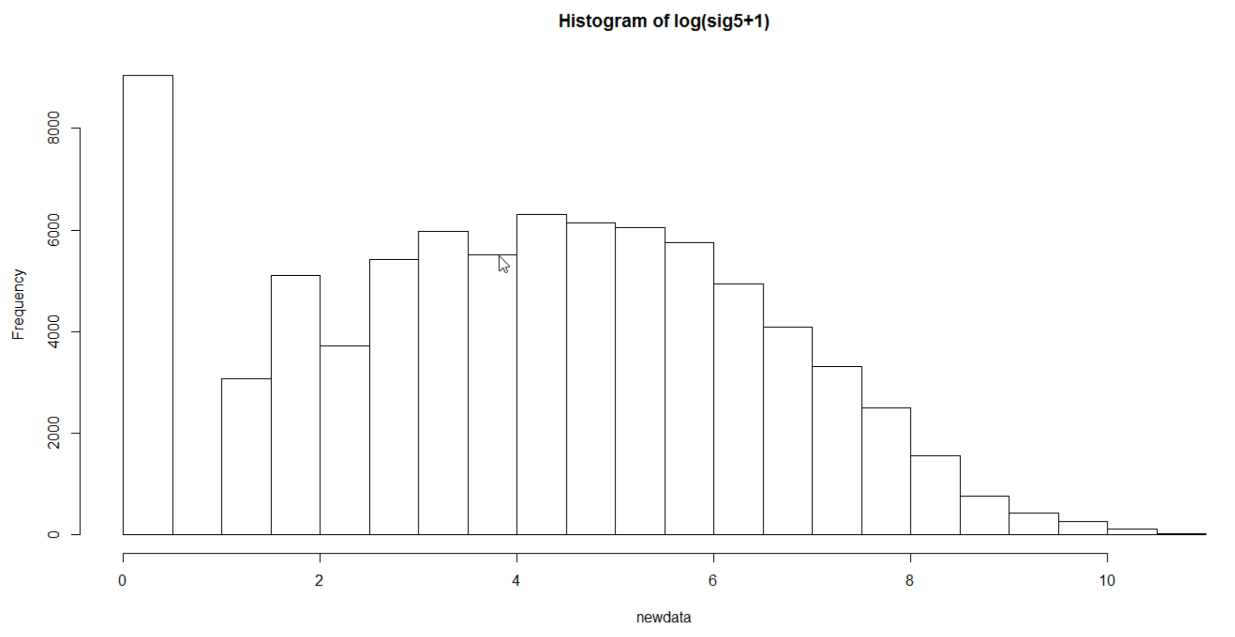
which becomes,



and,

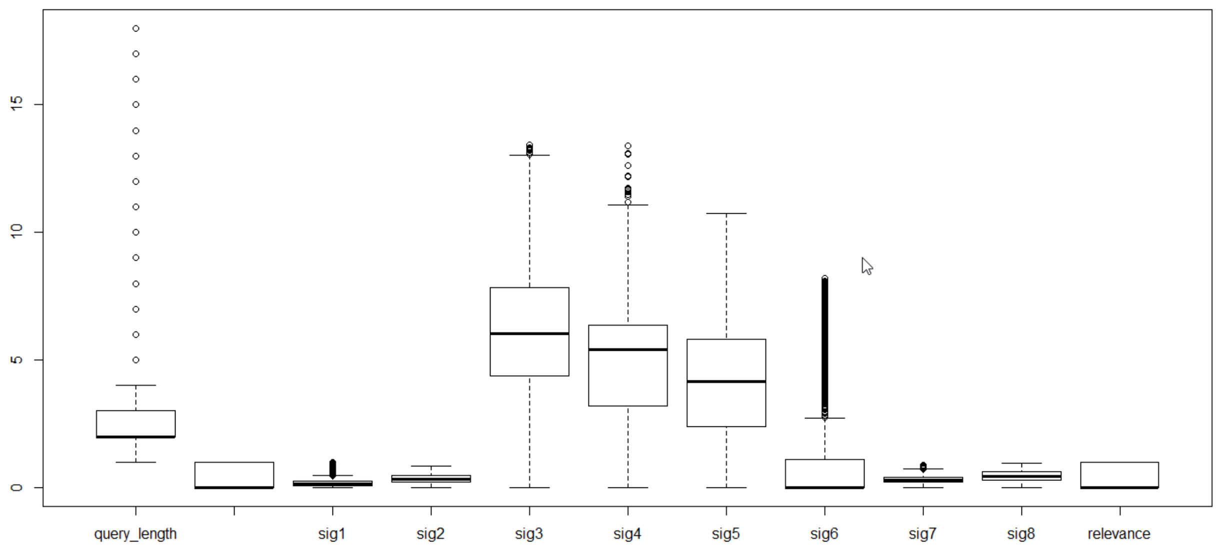


which becomes,



When it came to $sig6$ its distribution does not follow the same exponential relationship and even after the log transform the distribution still i

After log transforming $sig6, sig3, sig5, sig4$ we find that the box plot for the predictors look much better are as seen below,



So I am removing the first two columns from the training dataset to get the inputs used in the model as both **query_id** and **url_id** are just unique identifiers for queries and web pages respectively.

```
input = train[,3:13]
```

Selection of predictor variables

We usually perform variable selection when we have a large number of predictors, because it's hard to draw inference from a large number of predictors. Because we have only 10 predictors, this argument doesn't really have much merit.

When we have collinearity (highly correlated predictors) then some models fare worse with the redundant variables. So I used `cor` and found that $sig3$ and $sig5$ are highly correlated with each other. The covariance is0.8147, after which the next highest value is 0.479 between $sig5$ and $sig6$.

In all the models I trained removing $sig5$ never caused any significant improvement in the test error. In most cases the error even increased.

Preprocessing

I used functions in the caret library to check for some common problems that some datasets have.

Near Zero-Variance Predictors

I used `nearZeroVar` to check if among the predictors any exist which have only a handful of unique values that occur with very low frequencies. These can cause problems when I split the training data into sub-samples (for cross validation). This function takes care of this problem neatly by using two different metrics. But in our data we do not have this problem.

Linear Dependencies

I tried to find if any of the signals are linear combinations of the other signals using the function `findLinearCombos` which uses the QR decomposition of a matrix to enumerate sets of linear combinations (if they exist). For each linear combination, we can remove that predictor.

I found that there was no need to remove any signals due to this reason. This is expected as this sort of linear dependence is usually found in binary chemical fingerprints etc.

Imputation

We don't have missing values (`na` 's), so we don't have to do any imputation.

Outliers

I feel that outliers are never something that need to be removed. They indicate something about the underlying relationship between the web page and the query. So trying to mess with them will just cause the model to be overly simplified - which may be needed if I needed to perform inference, but as I need to get a good test error, all I care about is making a robust binary classifier which provides me with a good performance.

Different models are based on different assumptions (at different resolutions) to model outliers. Also outliers are usually due to a incorrect measurement etc.

Outliers are rare (unless the method of populating the dataset is flawed). When I used Tukey's method to identify the outliers ranged above and below the 1.5*IQR.

I also find other that some models require specific preprocessing which I will be discussing along with the model itself in Data Mining section of this report.

Data Mining

I am using the `train` of the the caret library in order to train most models (except knn where I use `class`). It also allows us to use cross validation to choose the tuning parameters for each model.

I use 80% of the data (I refer to this as the Training set) to tune any chosen model via some form of cross validation after which I compute the test error on this model using the remaining 20% (which I will refer to as the Validation set). I use this sort of splitting due to the Pareto principle.

I used $set.seed(2325)$ wherever any randomness was needed.

The following is a table of errors I got,

Model	**Error from validation set**
$Naive\ Bayes$	37.01899%
$Bagging\ and\ Random\ Forest$	34.12%
KNN	34.01624%
SVM	33.74563%
$Boosting$	33.25422%
.	.

NaiveBayes

So I started with Naive Bayes as it’s a popular baseline classifier for similar classification problems. It runs pretty fast (unlike the iterative algorithms which we will work with later) and is simple to understand.

I obtained a error rate of 40.11% when I train the model with all 10 predictors. While I get 39.73% when I remove $sig5$ which is just a slight improvement.

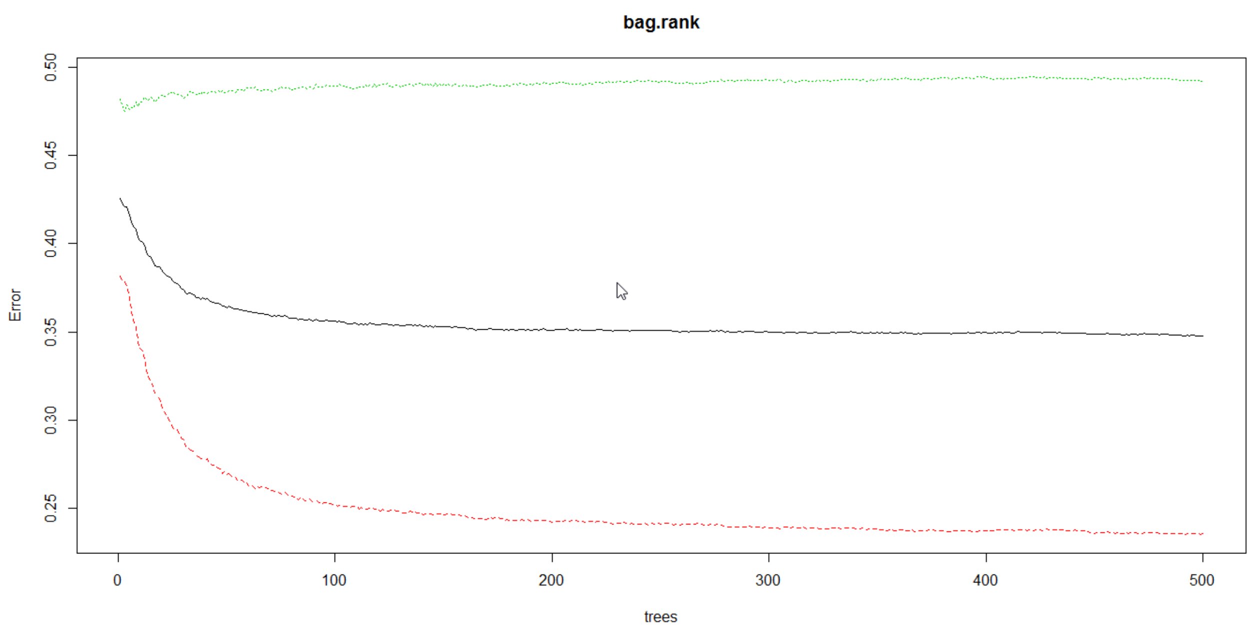
I tried scaling the predictors but the error did not change, this makes sense as algorithms such as Linear Discriminant Analysis and Naive Bayes do feature scaling by design and performing one manually would cause no effect.

I attribute the significant classification error due to the independence assumptions for the predictors which is highly unlikely to be true.

After I log transformed the signals $sig3, sig5, sig4$ the Naive Bayes error dropped to 37.01899%.

Bagging and Random Forest

I used the `randomForest` library with `mtry` equal to all 10 predictors. I tried more than 1000 trees yet found that after 200 trees the OOB error converges so I use `ntree` equal to 500.



When talking about preprocessing, one of the benefits of decision trees is that ordinal input data does not require any significant preprocessing. In fact, the results should be consistent regardless of any scaling or translational normalization, since the trees can choose equivalent splitting points.

That's why even after the log transform the error just slightly drops from 34.87% to 34.78125%.

In random forest we get pretty similar results as the number of predictors are so small that choosing a subset doesnt really change things much.

I get the OOB error to be 34.12%

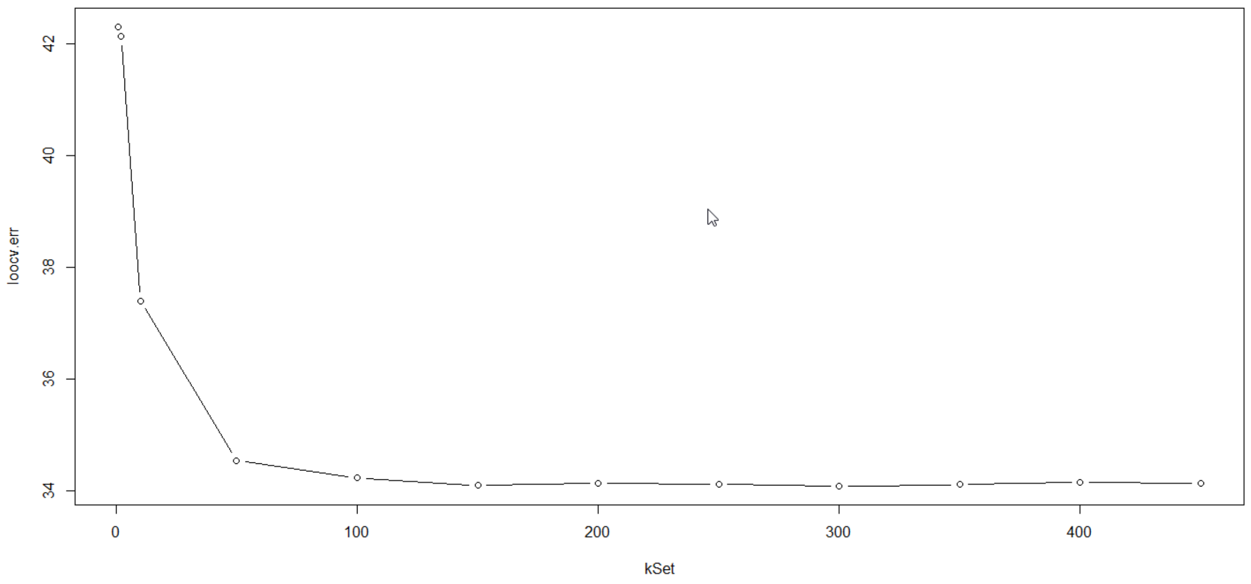
KNN

The k-Nearest Neighbors algorithm gave me a error of 41.53654%, but this large error is understandable because the data was **not scaled**!

After scaling I saw that I am getting more than a 7.286% improvement in the error. This illustrates how important feature scaling is to KNN, this is because the usual distance metric used to decide the similarity between two observations is Euclidean distance, so only by scaling will we avoid penalising some predictors just because of its units.

The only tuning parameter for this model is k the number of neighbours we consider when classifying a new data point. A small value of k results in a highly flexible decision boundary, while a large k gives a more smoother boundaries (less prone to overfitting).

I used `knn.cv` to perform LOOCV on the training set which showed me that the optimal value of k is 300 (34.08551% error rate). Check out the graph I plotted which shows how the model converges after $k = 150$.



When I used the optimal $k = 300$ on the remaining 20% validation set data I got a error of 34.01624%. After $k = 150$ it's pretty close to 34%.

SVM

As far is preprocessing is concerned, SVM needs numeric data, I converted the categorical attributes into numeric data. Manually scaling the data didn't really affect my results because the `caret` package scales the data internally.

I couldn't conduct extensive tuning by trying various kernels, and doing a proper 9 fold cross validation because every time I tried to do that R ended up running for hours straight (record was 16hrs).

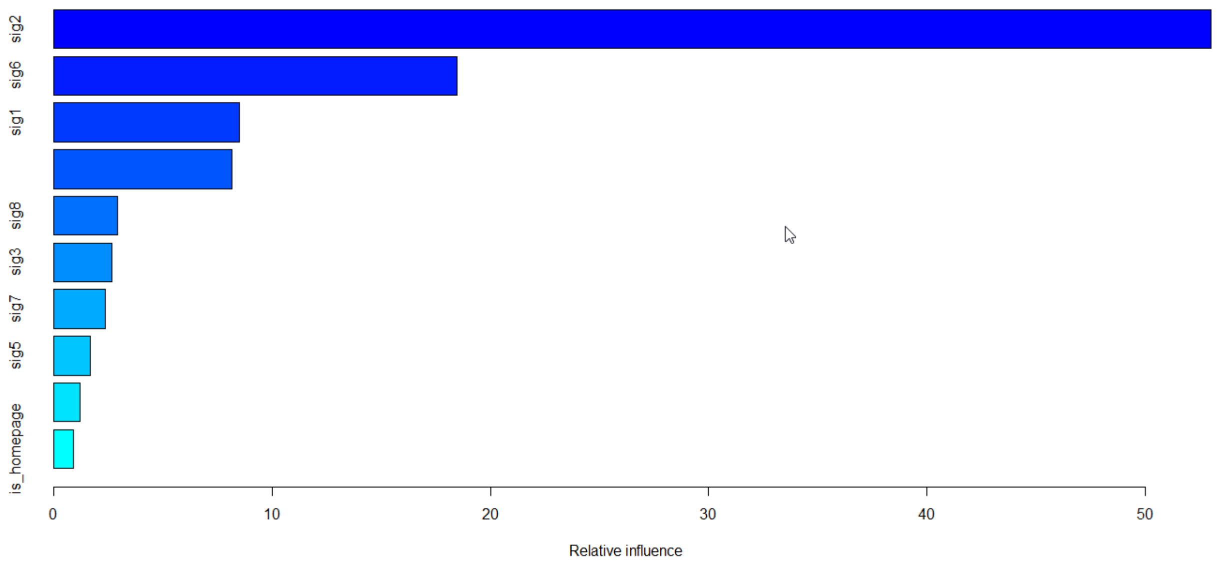
The best I got was using Radial Basis Function where I obtained a validation error rate of 33.74563%.

Boosting

So I used 9 fold cross validation to select the optimal tuning parameters for boosting. I used the Area Under the Curve (AUC) of the ROC as the metric to minimize.
The tuning parameters I finally obtained are,

$n.trees$	$interaction.depth$	$shrinkage$	$n.minobsinnode$
150	3	0.1	10

We find that $sig2$ is a highly important predictor for this model. After which $sig6$ and $sig1$ are next.



So I attribute the success of this model to the fact that it reduces bias while not increasing variance of the model.

Code

I am maintaining the code in my Stats202 github repository.

> Written with StackEdit.