Final Project Report

Stats 202: Data Mining and Analysis

Miaomiao Zhang Stanford ID #006321250

August 15, 2018

Abstract

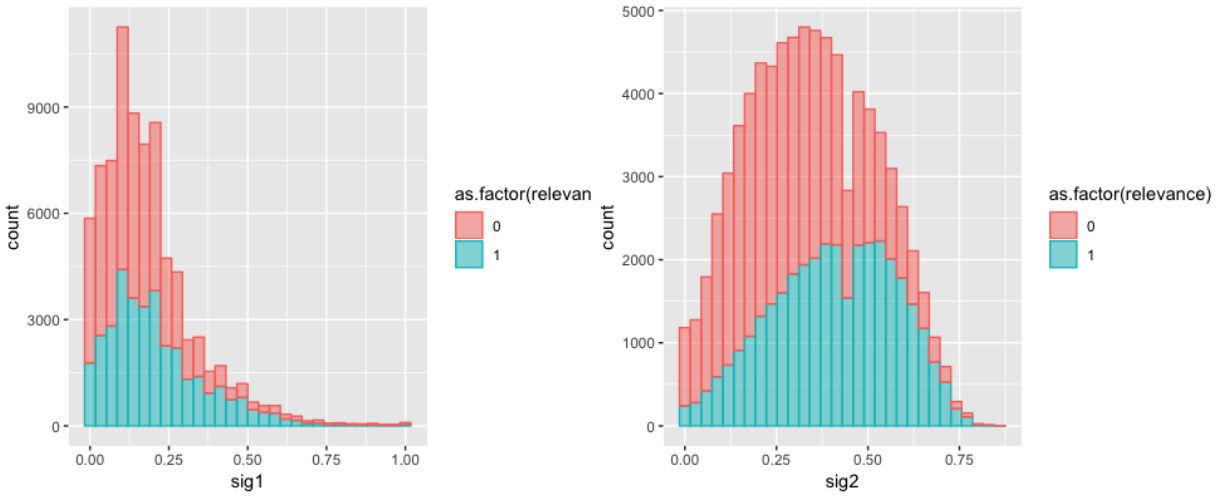
The goal of a search engine is to return relevant documents for search queries that users enter. Search engines use large amounts signals to determine the relevance of a url and then return a list of urls in order of relevance. It is essential to understand that different queries have different levels of difficulty in terms of finding relevant urls. This concept is carried through this project, including the process of data mining, model building, and parameter tuning. Two relevancy classes are labeled 1 and 0 for classification purpose. After engineering the features, I applied several different approaches to train the training set of the training data, tuned the models using validation set, or cross-validation approach, and tested the optimized models using a hold-out test set. Finally, I took a majority vote of my finest models' predictions to return responses for the test dataset.

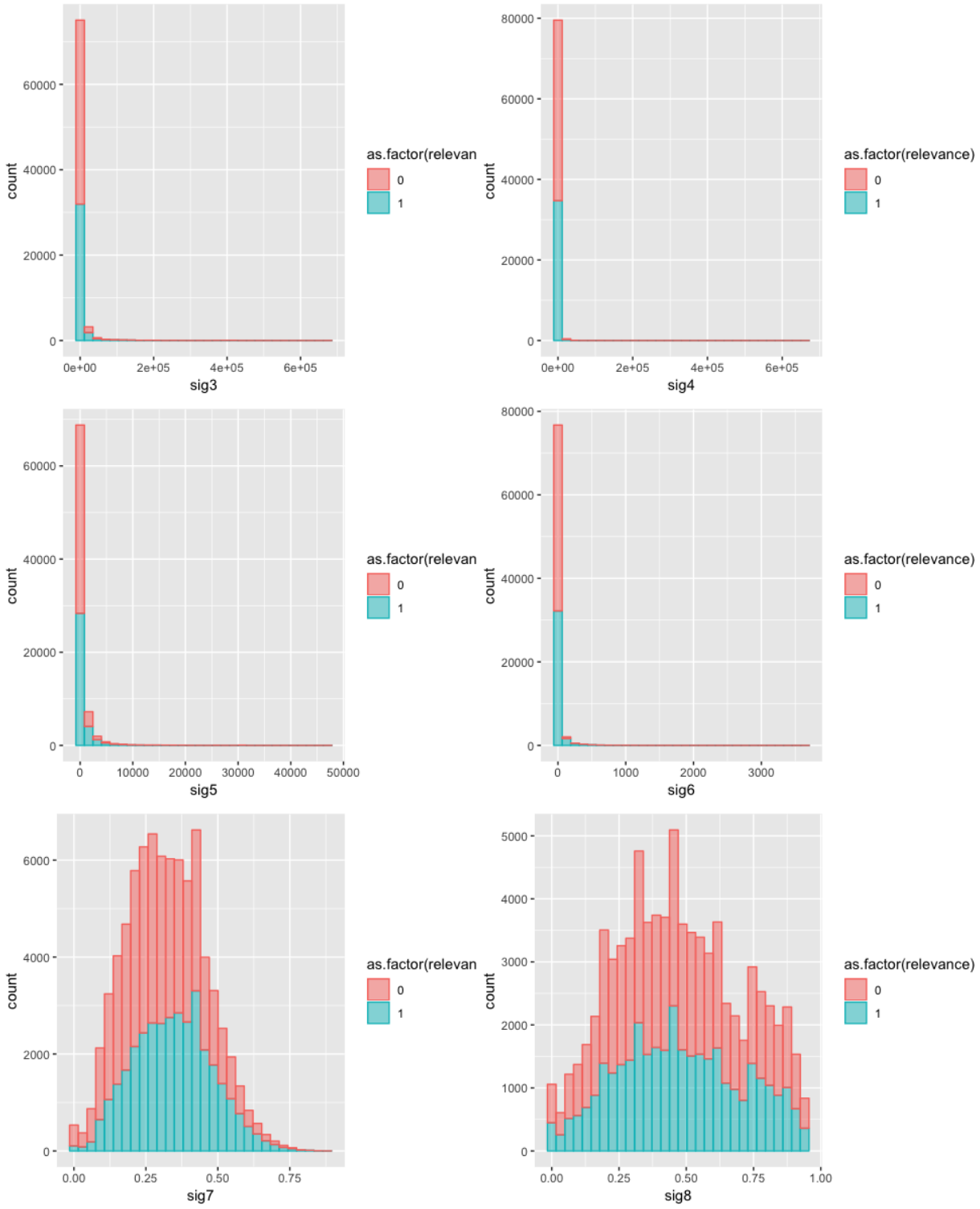
1 Introduction

I apply several binary model building approaches to the training dataset, in order to develop a best classifier that can distinguish the relevancy class. The training data includes 10 attributes and 80,046 observations from search engine query and url data. The 10 features could be used to help predict whether the url is relevant for the query. Additionally, another test data set is provided, which contains 30,001 observations with the same labeled features. My goal is to make relevance predictions for each row (a url for the queries) in the test data set.

2 Data Observation

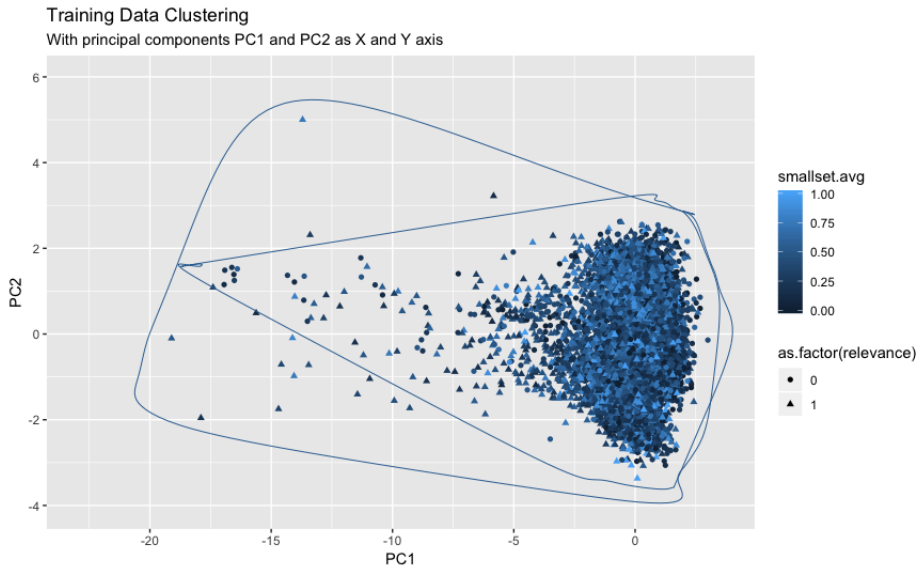
Both training and test data sets are examined and no missing data are found. There is not severe imbalance in the response of the training data – 43.7% labeled relevance equals to 1 (indicating a relevant url), and the rest 56.3% labeled relevance equals to 0 (indicating a irrelevant url). I implement `hist()` to visualize each feature's distribution, with colors indicating the relevancy.




count
sig3
0e+00
2e+05
4e+05
6e+05
as.factor(relevan
0
1
sig4
as.factor(relevance)
sig5
0
10000
20000
30000
40000
50000
sig6
1000
2000
3000
sig7
0.00
0.25
0.50
0.75
sig8
1.00

2.1 Selection

Features *query_id* and *url_id* are somewhat trivial, so I do not include them in my further anaylsis. But the fact that each query returns different urls enables me to transform these two attributes into another feature (see 2.3 for more detail). But before engineering the features, I graph the first two principal components in order to produce a low-dimensional view of this dataset and intend to pre-screen the attributes. As seen, a robust clustering shows that the two response classes are hardly separable from each other.

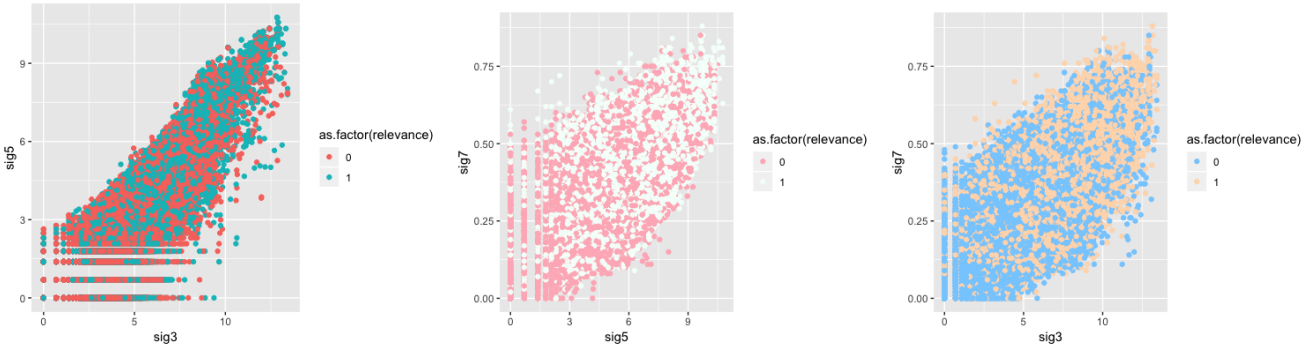


2.2 Preprocessing

Because *sig3* to *sig6* are severely skewed, I log-transformed these four features so that their distributions look like those for other signals and the patterns in the data are more interpretable. To avoid negative infinity value in my new dataset, I intentionally keep those with value 0 unchanged. In addition, I also use `caret` package to scale the eight signal features to have standard deviation one. This is essential for helping to meet the assumptions of model building later on (especially in 3.4 Support Vector Machines).

One of the variables, *is_homepage* is categorical, which only takes on value 0 and 1, so I turn the numerical values into factors. As we can see, none of any single feature plays a dominant role in each observation's relevancy value, as the 0's and 1's are evenly distributed in each bar, except for *sig6*. After passing over a certain threshold value of *sig6*, the urls become relevant all along. It is also worthy of attention that 52,165 out of 80,046 observations have a *sig6* value of 0.

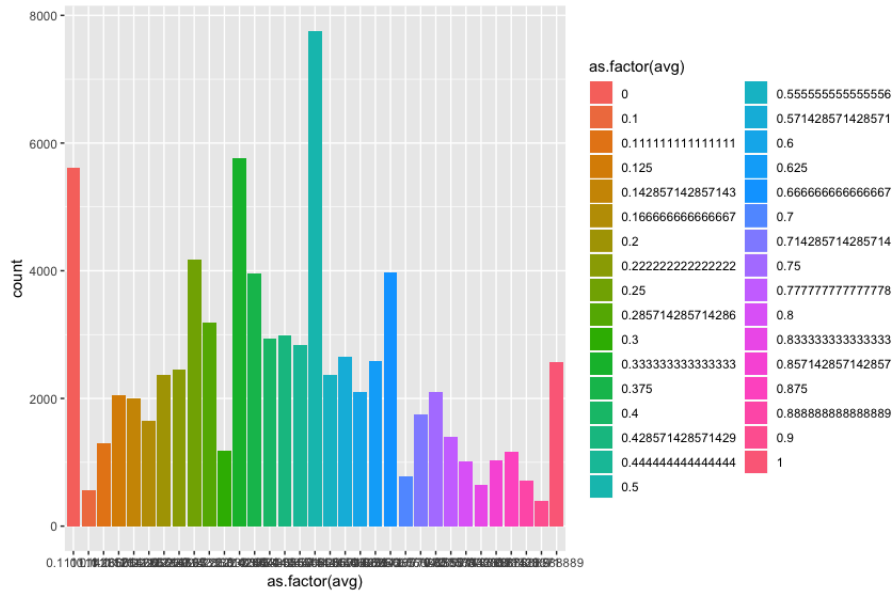
Finally, I look into each feature and examine the correlation of each pair using `cor()` command. As expected, none of the variable has a strong correlation with the outcome relevance. Strong correlations bewtween features are noticed – a correlation of 0.9043 between *sig3* and *sig5*, a correlation of 0.7005 between *sig3* and *sig7*, and a correlation of 0.7153 between *sig5* and *sig7*.



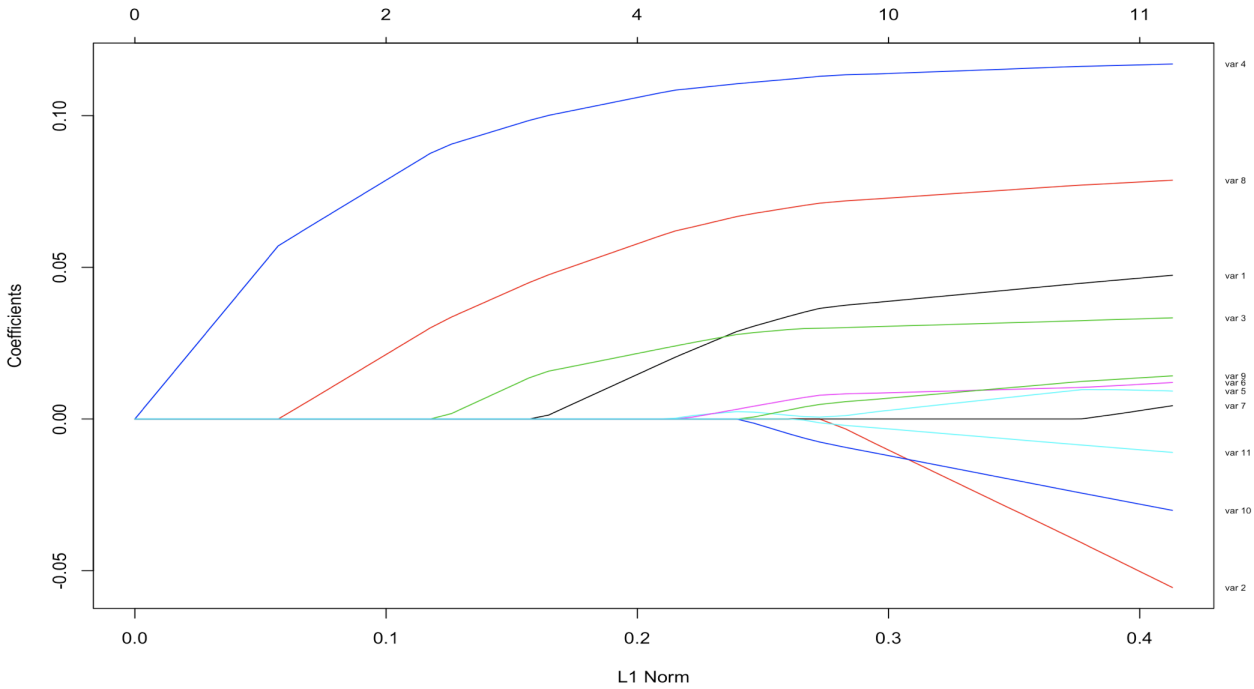
In the end, I also visualize the distribution of all features on the test data and fortunately, those distributions are similar.

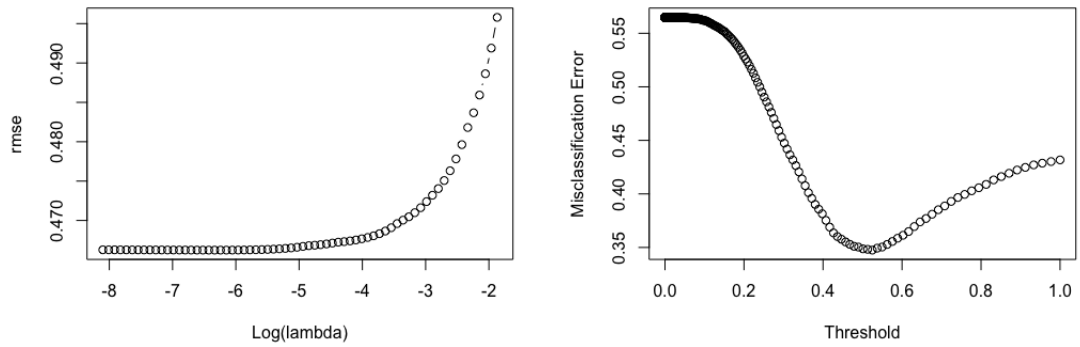
2.3 Transformation

As mentioned in 2.1, features *query_id* and *url_id* are somewhat trivial, but it is also understood that queries themselves must have some inherent property in terms of difficulty of finding relevant urls. I leverage the *query_id* and *url_id* in a way of counting how many urls returned for each query and averaging its relevancy to quantify the insight. As a result, I add column *id* and *avg* into my original training dataset. The url count per query id can be used as a new feature later, but the average will only be used in lasso regression below.



I performed lasso regression on the training data set and intended to select features. A threshold of 0.00177 with the lowest validation set RMSE and a probability threshold of 0.5248 were chosen. Even though some of the coefficient estimates of the variables are shrunk down to zero quickly, the optimal λ has a really small value 0.00177, which is a small penalty to put on the least squares regression. In addition, *sig5*(var7 in the plot) is the first variable shrunk to zero, but it has high correlation with other variables in the dataset so its effect might be masked. It is worthy of attention that *sig2* (var4) and *sig6* (var8) are the last two kept in feature selection.





3 Data Mining

Seven different models' performances and evaluations are provided in the following subsections. First, I divide the training data into three parts – one for training, one for validation, and another for testing. To avoid disrupting the correlation among the same queries, I randomize data and then split the data based on *query_id*. Second, I use the training set to train my model, the validation set to tune my parameter, and select the model that yields the lowest validation error rate. Finally, I run the selected model on the hold-out test set and record the test error rate.

3.1 Logistic Regression Model

Logistic regression is a classification parametric model and is very popular when k = 2. When all 11 predictors are included in the model, the result shows that *sig3* and *sig5* are not significant, but defining variables to enter in the model, adding, or removing explanatory variables can be complicated and must be carefully planned in logistic regression.

Given that there are important correlations between the two variables, I decide to remove *sig3* and *sig5* one at a time and test my model's performance accordingly in case of over-adjustment. Removing one of them makes the other a little more significant (as expected), but the test error rate does not change too much. I also add an interaction term *sig3*sig5* and surprisingly enough, all of the predictors including *sig3* and *sig5* become important. In the end, because I have seen *sig2* and *sig6* are two of the most important features in the LASSO model, I suspect that the addition of an interaction term *sig2*sig6* would have a positive impact on my response output, and therefore improve my model performance. The table below shows the test results. Unfortunately, not a significant improvement on the test error rate is seen.

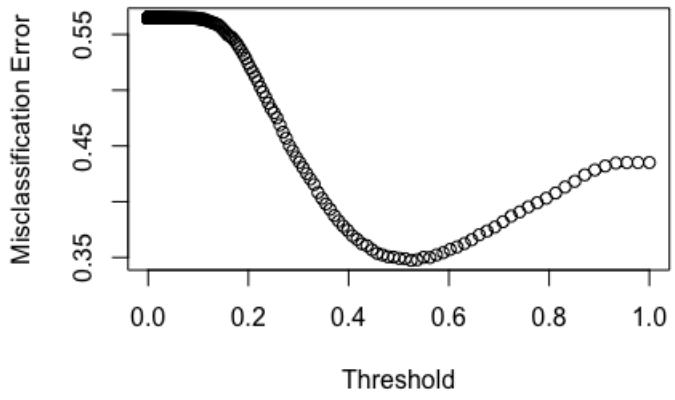


Figure 1: Probability Threshold vs. Validation Error Rate

similar plots were produced for parameter probability tuning in all five models

Predictors	Probability Threshold	Test Error Rate
ALL	0.5248	0.3387
-sig3	0.5128	0.3379
-sig5	0.5248	0.3388
-sig3 - sig5	0.5129	0.3371
+ sig3 * sig5	0.5248	0.3362
+ sig3 * sig5 + sig2 * sig6	0.5129	0.3355*

Note: LASSO-regularized logistic regression could also be performed in a similar fashion by an addition of $l1$ penalty term.

3.2 Naive Bayes

Naive Bayes is one of the simplest classifier which assigns each observation to the most likely class, given its predictor values. At a high level, Naive Bayes tries to classify instances based on the probabilities of previously seen attributes, assuming complete attribute independence. The *R* built-in `naiveBayes()` command is used here. The reason why we get a high test error rate of **0.3707** might be that it makes a very strong assumption about the data having features independent of each other while in reality, as we have seen in 2.2 before, there are indeed correlations. In addition, Naive Bayes is more suited to categorical variables, but not so much for numeric features: it makes another strong assumption that the numerical variables are normally distributed.

3.3 K-Nearest Neighbor Model

Built on Bayes rule and thereby classifying a given observation to the class with highest probability, KNN approach instead *estimates* the conditional distribution of Y given X. Because the KNN

classifier first identifies the k points in the training data that are closest to a given observation, the choice of k has a drastic effect on the KNN classifier obtained. However, before tuning parameter k, I need to first scale and center all the data because "closest" is often in terms of the distance between points. Different measurement scales on the features would otherwise introduce unwanted bias into our model. `Caret` package performs knn for me using a 10-fold cross validation. Accuracy was used to select the optimal model using the largest value. The final value used for the model is $\mathbf{k = 43}$. The test error rate is **0.3393**.

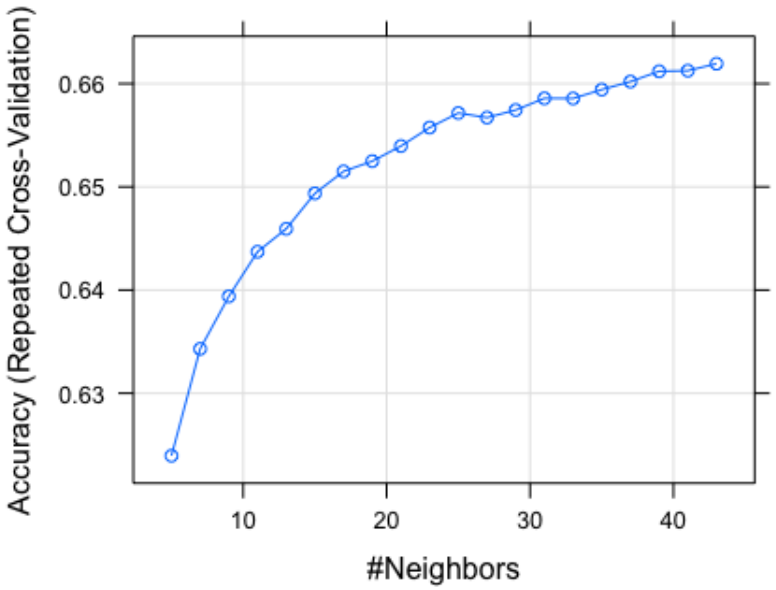


Figure 2: Parameter K Neighbors vs. Cross-Validated Accuracy

`caret` package performs cross-validation by separating the folds randomly – this could cause some drawbacks and could be improved upon (See 4 Model Improvements)

3.4 Support Vector Machines

Support vector machines tend to do better if predictors are quantitative and each of which has a roughly linear or quadratic relationship with the outcome. It is a parametric method, which is different from decision-tree-based methods (3.5 - 3.7), which do not assume a functional relationship between the predictors and the outcome. However, it is a lot more flexible than logistic regression so it tends to do well on huge amount of observations and low dimensionality of predictor space.

Here I consider setting `scale = TRUE` since I assume that each variable does not have the same unit measurement. If one variable is on a vastly different scale than the others, then this variable is going to inordinately influence the final model. Again, the optimized parameter C is selected based on the minimum mean misclassification error from the validation set.

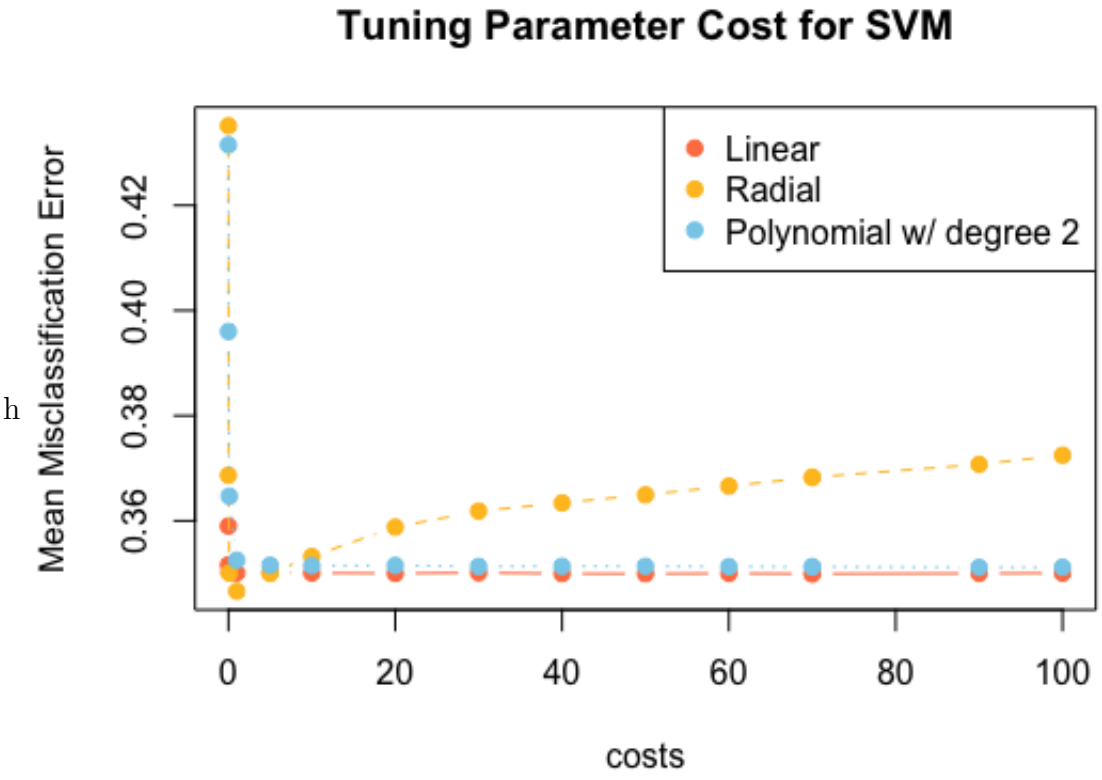


Figure 3: Parameter C Cost vs. Validation Misclassification Error

Kernel	Cost (*budget*)	Test Error Rate
Linear	50	0.3404
Radial	1	0.3291*
Polynomial	90	0.3390

3.5 Decision Trees

One of the biggest advantages of decision trees is that they easily handle qualitative predictors. If there are interactions between predictors, we do not have to pre-specify them in decision trees. Lastly, I do not have to assume linear relationship between my predictors and response.

I first build a large "bushy" classification tree on the training set of the training data using `rpart.control(cp=0.00001)`, which could have too much variance. In general, more levels in the tree mean that it has lower classification error on the training. However, it also runs the risk of overfitting. Often, the cross-validation error will actually grow as the size of the tree grows (at least, after the *optimal* level). Therefore, I use built-in cross-validation in `rpart()` to prune the tree optimally. The output `plotcp()` shows me the lowest level where cross-validation error falls into one standard deviation of error minimum occurs, i.e., the level whose `xerror` is at or below horizontal line.

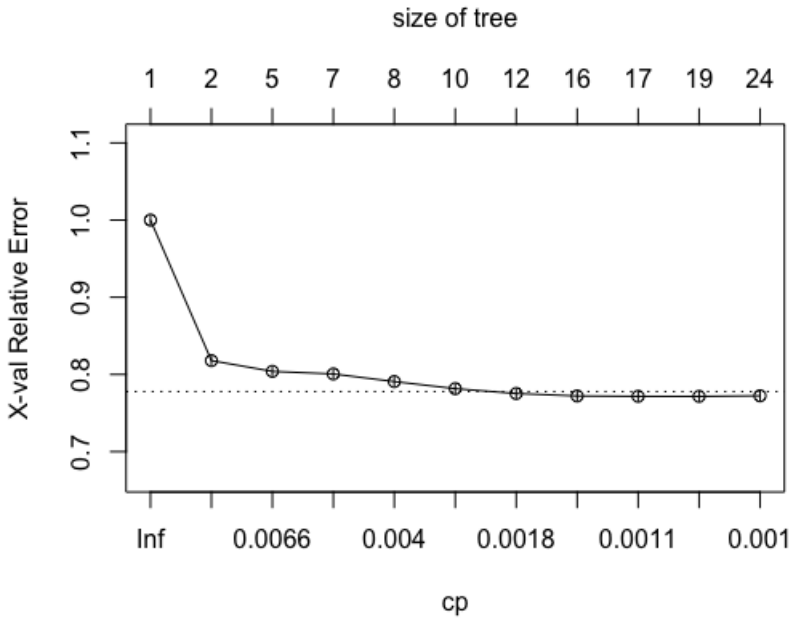
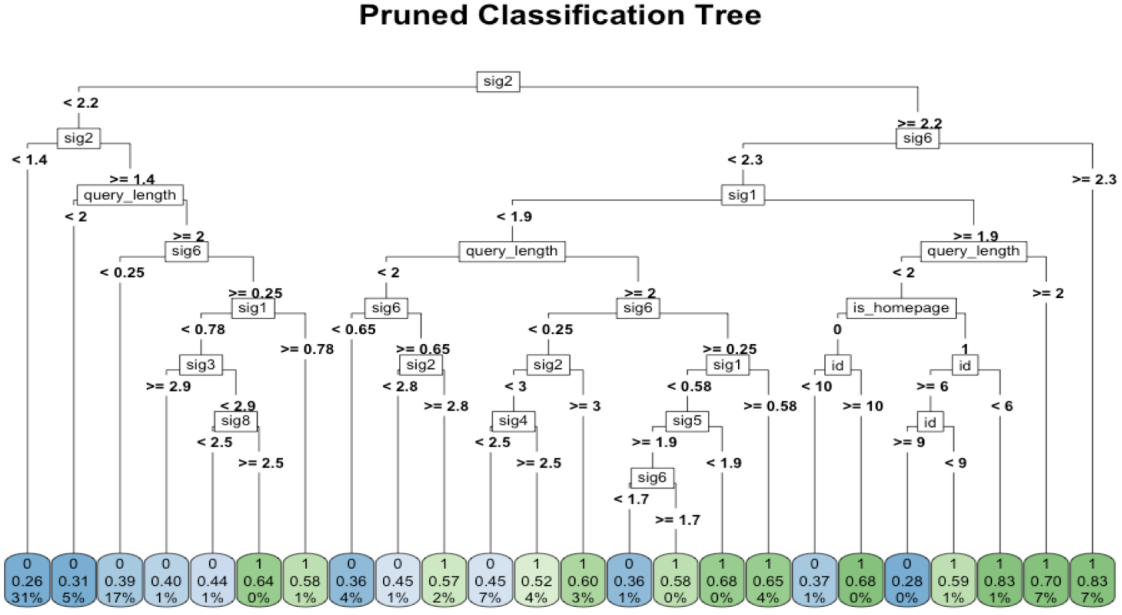


Figure 4: Tree Size vs. Cross-Validation Error

As `caret`, `rpart` package performs cross-validation by randomly selecting observations to each fold, which could be improved upon for this dataset.

I choose this method as the basis for doing the pruning because it also takes into account the variability of `xerror` resulting from cross-validation.



Finally, I evaluate my pruned tree on the test set. There is a significant improvement from

pruning. In general, decision trees are simple and interpretable models for classification. However, decision trees are often not competitive with other methods in terms of prediction errors. Bagging, random forests and boosting, on the other hand, tend to outperform trees as far as prediction and misclassification errors are concerned.

Decision Tree	Training Error Rate	Test Error Rate
Unpruned	0.1962	0.4045
Pruned	0.3307	0.3439*

3.6 Random Forests

Bagging, random forests and boosting all work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees. Random forests provide an improvement over bagged trees by de-correlating the trees. A random selection of m predictors chosen at each split in a tree reduces the variance when we average the trees. Therefore, I skip bagging and only apply random forests.

To tune the parameter m, the number of splitting variables, I fit a series of random forests by setting a loop for `mtry` in 1 to 11 (there are 11 variables in total). As with bagging, random forests do not overfit if I increase the number of trees, so I just use a value of B sufficiently large for the error rate to have settled down.

By default of `randomForest()`, the mean of squared residuals and % variance explained are based on OOB or *out-of-bag* estimates (a very clever device in random forests to get unbiased error estimates, so each observation was predicted using the average of trees that did not include it).

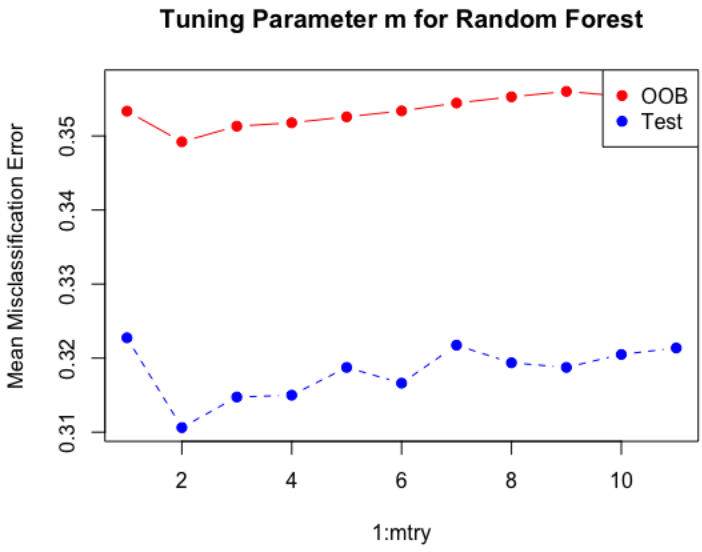


Figure 5: # of Predictors Selected m vs. Validation Set Mean Misclassification Error

We can see `mtry` at 2 is the best, both for the test error and for the OOB error. Even if the minimum occurs at different m, it would not be a good idea to choose the tuning parameter

value based on the model OOB error estimates, since these trees are repeatedly fit to bootstrapped subsets of the observations. We see that the OOB error is always higher than the test error: this further confirms that boostrapping destructs the inherent correlation within the same *query_id* group. The averaged prediction using each of the trees for each observation that is left out does not return good error rate result and has less interpretability power.

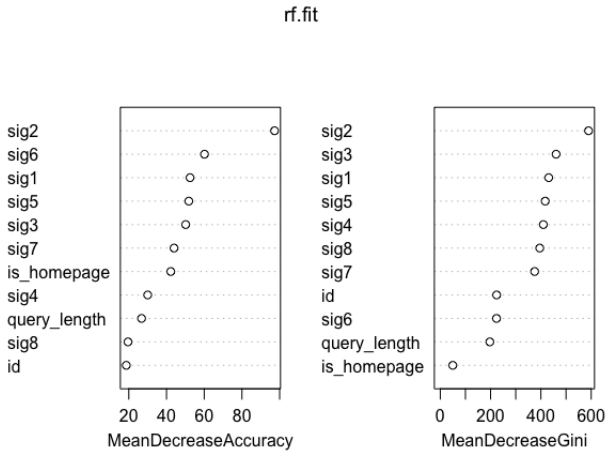


Figure 6: Importance of Variables in Random Forests

Gini index on the right of Figure 6 measures node impurity. Here because I am more interested in prediction accuracy, the mean decrease accuracy on the left is preferred to see which features are important. Random forests use out-of-bag (OOB) samples to measure prediction accuracy. Again, those are not good measurements for this dataset as we discussed above. It is also worthy of attention that *sig3*, *sig5*, and *sig7* have similar level of relatively high importance. Note that this might not be due to the fact that all of them hold true predictive power, but due to their high correlations among one another. The reason is as follows:

Since each split only considers a subset of the possible variables, a variable that is correlated with an "important" variable may be considered without the "important" variable. This would cause the correlated variable to be selected for the split. The correlated value does hold some predictive value, but only because of the truly important variable.

As a result, after I remove *sig 8, query_length*, and *id*, the least important predictors shown in Figure 6, run random forests again, and tune my `mtry` parameter, the test error rate becomes worse.

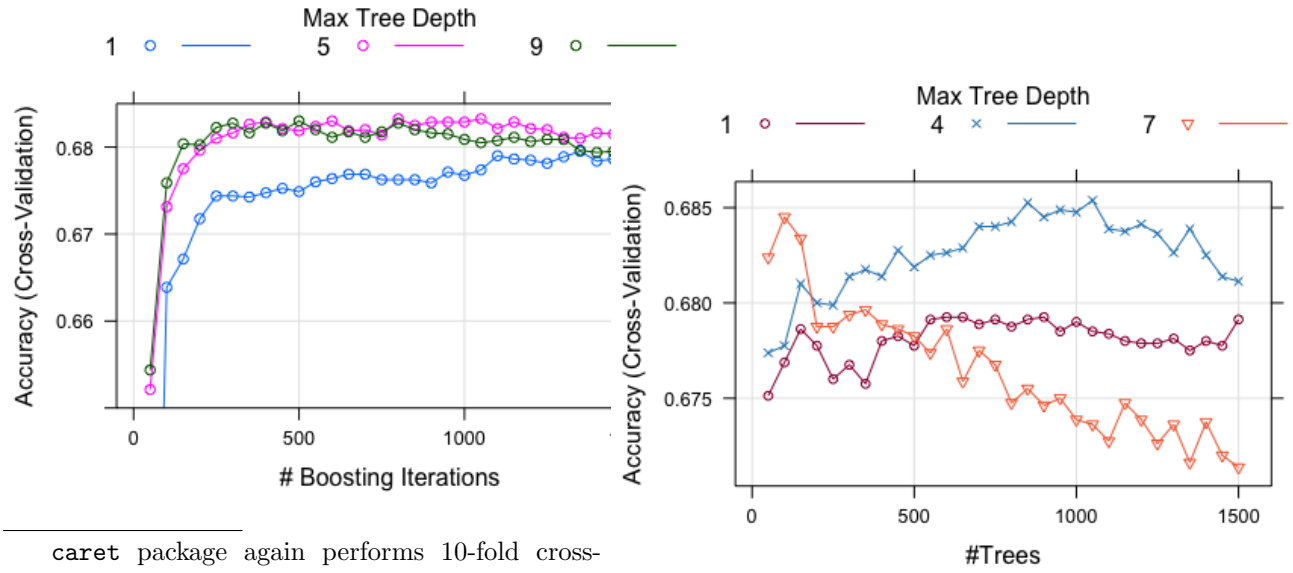
Features	Optimized m	Test Error Rate
ALL	2	0.3368*
-id - sig8 - query_length	2	0.3546

3.7 Boosting

Trees of random forest are built on a bootstrapped dataset, independent of other trees. In "slow learner" boosting, however, trees are grown sequentially: the growth of each tree uses information

from previously grown trees. In gradient boosting, each tree is growing to the residuals left over from the previous collection of trees and fitting small trees to the residuals. In adaptive boosting, it works by upweighing points at each iteration which are misclassified. Both methods slowly improve the model in cases where it does not perform well.

In order to perform boosting, we need to select 3 parameters: number of trees B, tree depth d, and step size λ (α in the lecture). The last parameter λ is just a function of misclassification error that we defined in the algorithm. Therefore, I do not have to tune it. However, unlike random forests, the number of trees is a tuning parameter because if we have too many we can overfit. The plots below show my tuning results of performing `gbm` (left) and `adaboost` (right).



`caret` package again performs 10-fold cross-validation for boosting. Randomly picking subsets of training observations in this dataset may not be the best choice.

Figure 7: # of trees vs. Cross-Validated Accuracy

In Figure 7, I display the training accuracy as a function of the total number of trees and the interaction depth d. For this particular test set, `gbm` and `adaboot` have similar performance. Due to the speed limit, I only selected 8,000 out of my training set to train the model and 10-fold cross validation is performed on the 8,000 observations to select the optimal parameters. This might result in weaker model performance and some bias of my error rate.

The final values used for the two boosting models are listed in the table below. Note that the standard errors of test error rate among each d are around 0.02, making none of the differences significant, so interaction depths may not need to be tuned too hard either. In Figure 8, *sig2*, *sig6*, and *sig1* again are ranked as the most important features.

Boosting	# of Trees B	Interaction Depth d	Shrinkage λ	Test Error Rate
Gradient Boosting	800	5	0.01	0.3352
Adaptive Boosting	1050	4	0.01	0.3375

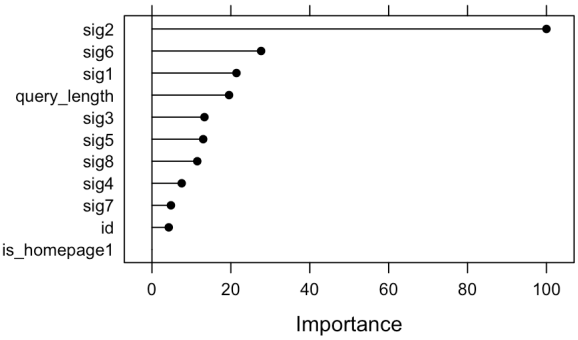


Figure 8: Importance of Variables in Boosting

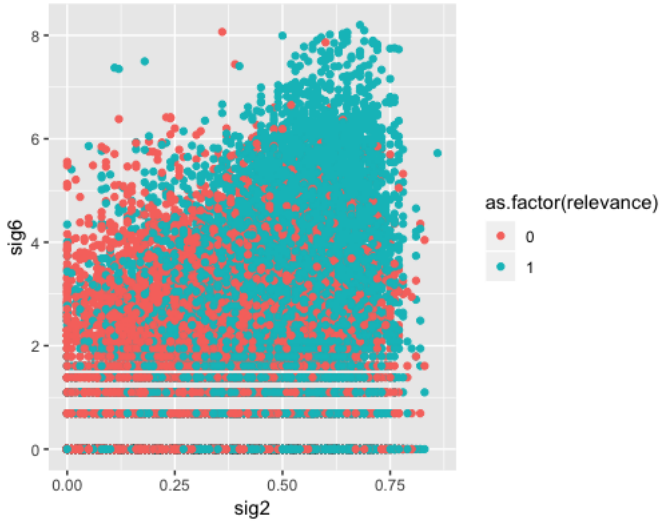


Figure 9: Among all the tree models, they tend to agree that *sig2* and *sig6* are of the most important features. This is not hard to visualize here. The blue dots are separable, though far from perfectly, from the red dots.

4 Model Improvement – "Tricks" in *query_id*

Validation-set approach applied in the project has several drawbacks:

1. The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and the validation set.

2. Only a subset of observations – those included in the training set – are used to fit the model.

3. Validation set error may tend to overestimate the test error for the model fit on the entire dataset, because in general, the more training data we have the more info our model gets, and the lower error we should expect to get.

The 10-fold cross-validation performed in some of the models also have drawbacks for this particular dataset. `R` has packages that randomizes the samples and then tune models based on the minimum cross-validated error rate. However, such cross-validation separates the same queries into different groups, which breaks original setting of grouping the same queries together. Therefore, one area that may improve the accuracy rate further is to manually split the training data into 10 folds based on *query_id* (or code a function), and then perform training/validation accordingly.

Additionally, I also intended to perform clustering on my training and test set so as to find homogeneous subgroups among the observations. For each query in the training set, the urls' average relevancy above 0.8 can be categorized as "easy" query since most of the urls returned are relevant in this case. When mixing the those and test data set together, clustering could potentially help me to identify which queries in the test data are similar to the "easy" ones in

training data by only looking at their features. The same clustering method can also be applied to the "medium" and "hard" queries. The queries are thus grouped and trained separately – a potential improvement on overall dataset.

Lastly, when selecting the best model, I used misclassification rate (accuracy) as a criterion. However, other measures of classifier performance like **recall** and **precision** can also be used. ROC curve that captures areas under the curve AUC is also useful for comparing different classifiers.

5 Conclusion

Classifiers		Test Error Rate[1]
Logistic Regression	$p \approx 0.5$	0.3355*
Naive Bayes	—	0.3707
K-Nearest Neighbors	$k = 43$	0.3393*
Support Vector Classifier	Linear Kernel	0.3404
	Radial Kernel	0.3291*
	Polynomial Kernel	0.3390*
Decision Trees	Unpruned	0.4045
	Pruned	0.3439
Random Forests	$m = 2$	0.3368*
Boosting	Gradient Boosting	0.3352*
	Adaptive Boosting	0.3375*
Majority Vote	—	0.3301*

Both logistic regression, support vector classifiers, random forests, and boosting are robust to observations far from the boundary or the hyperplane. Decision trees, on the other hand, can be non-robust. Small changes in the data causes large change in finding estimated tree, so I did not include it when I took the majority vote of the models' predictions.

With support vector machines, one disadvantage is that it uses all the features, and it does not easily select which features are important, and thus lack the power of interpretation of the solution. As compared to logistic regression, even though we are not interested in the exact class probabilities, the probability of returning relevant documents for each queries that users enter being 0.51 as opposed to 0.99, the classification is the same, but the implications are very different.

As noticed, trees in general deal with messy, real data well. Extraneous predictors do not affect too much of their performance and there are no assumptions that the response has a linear (or even smooth) relationship with the predictors. Although random forests and boosting are among the "state-of-the-art" methods for supervised learning, their results can be difficult to interpret. Here in this project, because I am more interested in prediction, and more specifically, just the prediction performance, they are very good techniques.

In the end, we should note that a significant improvement on accuracy does not come too much from the choice of methods, but more from pre-processing and engineering the features. Because several of the models have similar performance, I take a majority vote and the test set returns a relatively low error rate. The same is thus done on the test data for my final submission.

[1]Test error rates less than 34 % are starred