Final Project Report

STATS202 Wentao Zhu

Stanford ID: 006321396

Abstract

The task of this project is to perform webpage relevance analysis, which is essentially a binary classification problem. We first analyzed the dataset and evaluated the importance of variables, and then applied multiple classification methods. Fine-tuning methods are performed under cross validation. Finally, we tried to ensemble different models to produce a more comprehensive classifier.

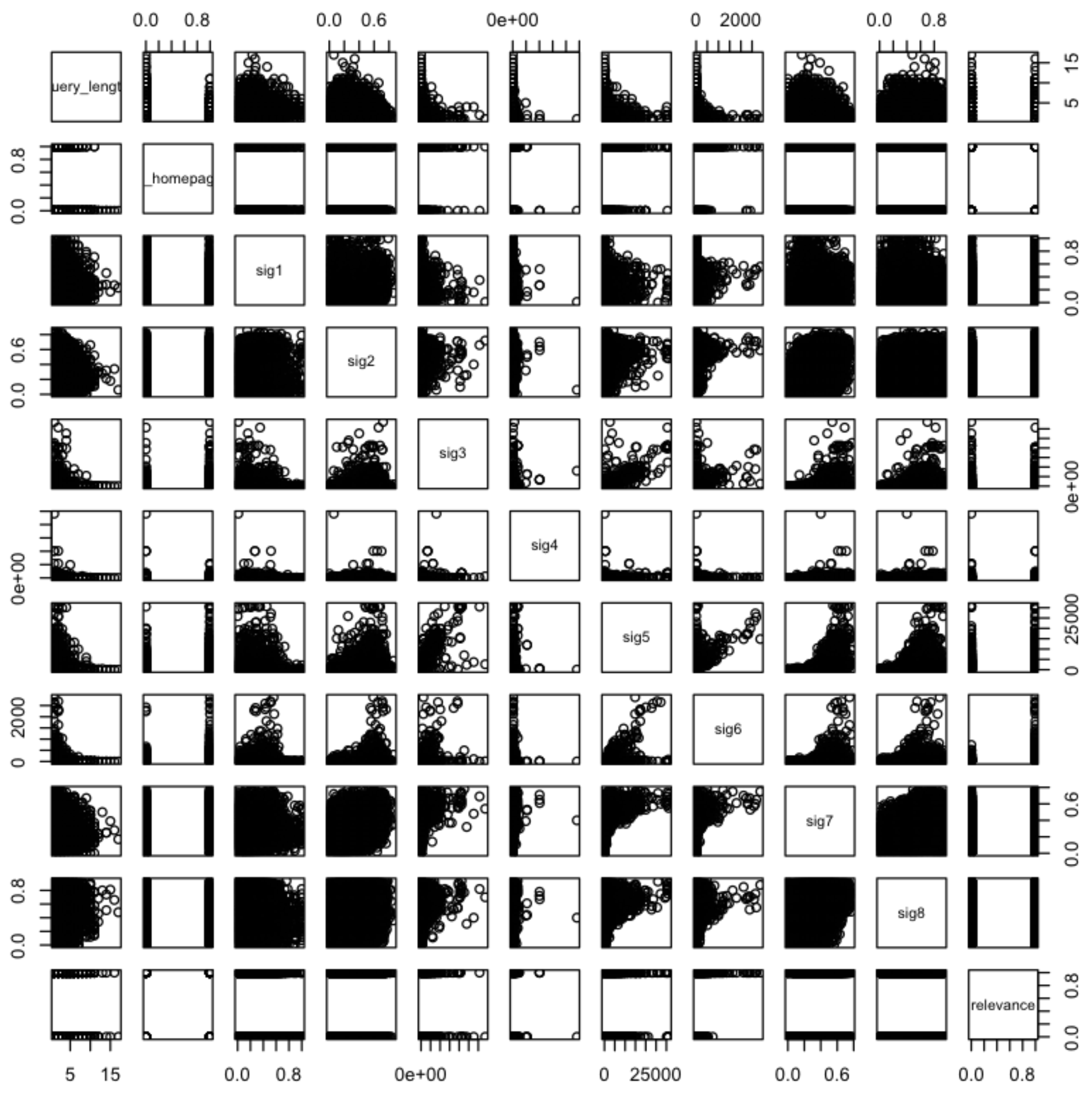
Introduction

The goal of the project is to make relevance predictions for each row (urls for a query) in the test data set. The predictions are binary: 1 for irrelevant and 0 for relevant. A training set of 10 attributes and 80046 observations is provided. As the meaning of the predictors are unknown, it is necessary to perform feature selection and transformation while applying classification methods.

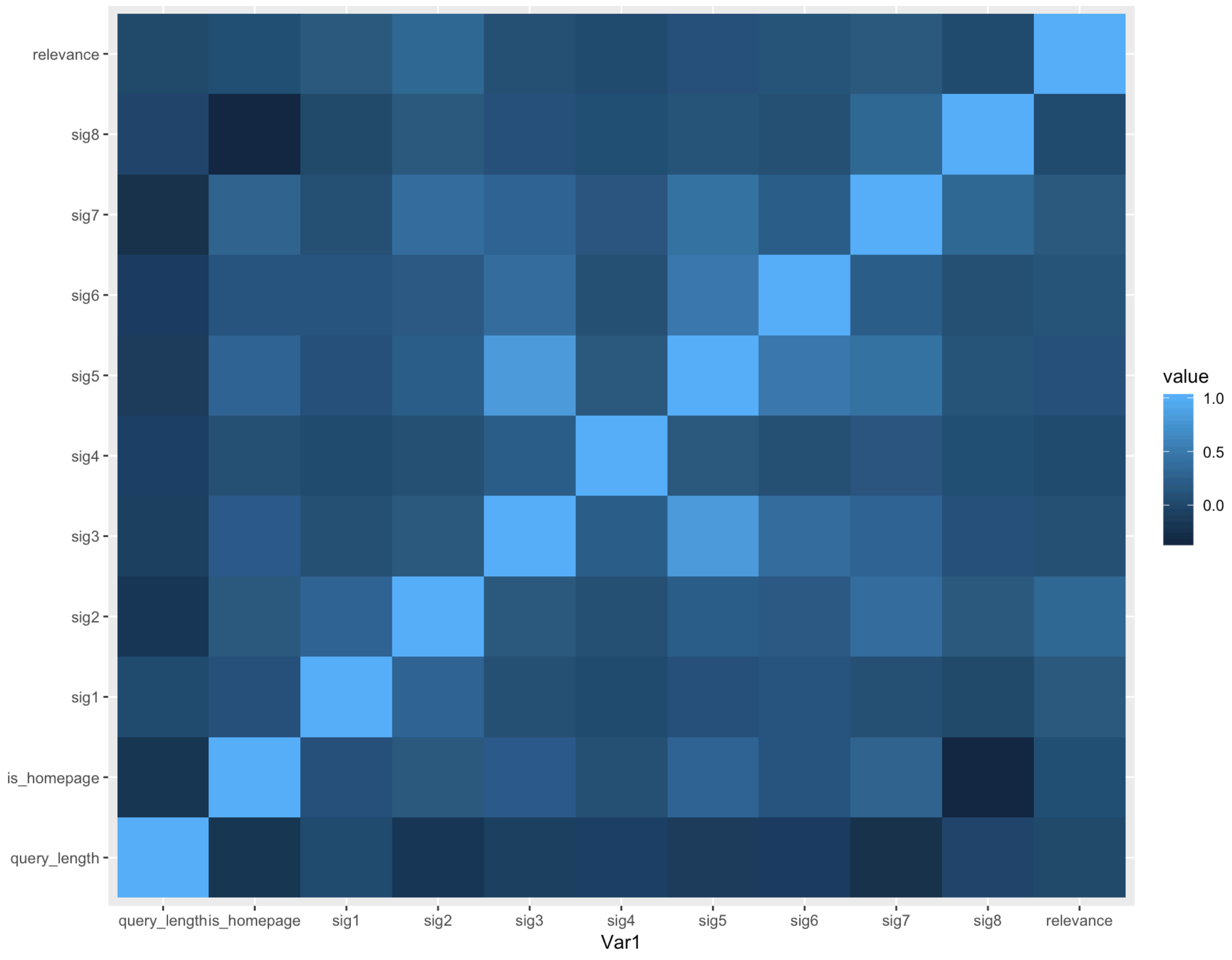
Data Observation

First of all, we observed the dataset and measured the importance of attributes. The training data are well balanced with 43.71% positive (relevant) samples and no missing entries. There are 10 predictor candidates, among which `is_homepage` is binary.

We then visualized the pattern of distribution between pairs of variables and computed the correlation matrix.

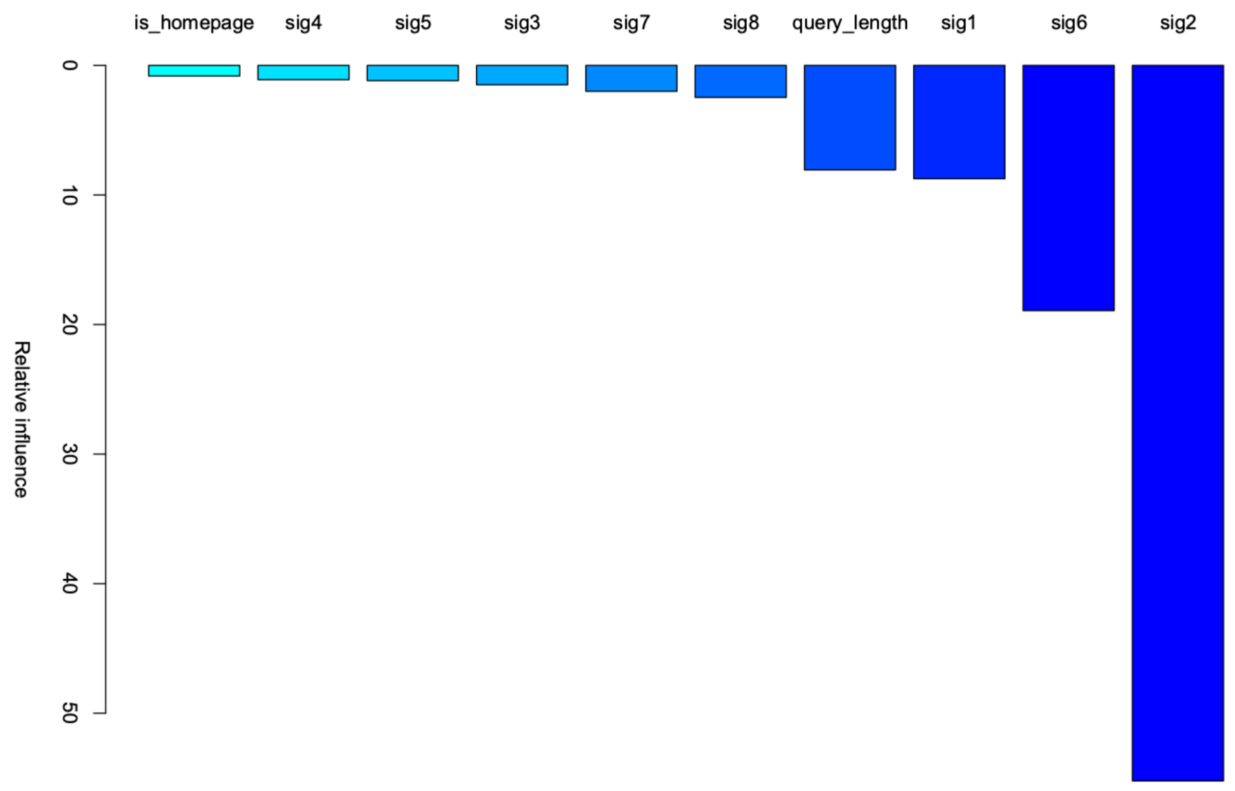


The correlation matrix heat map:



We found that `sig3` and `sig5` are highly correlated with correlation coefficient more than 0.81, and `sig2` is the most directly related predictor for `relevance`.

We then applied best subset selection method using forward/backward selection using the crirerion of Akaike Information Criterion (AIC), the results suggested that `sig2`, `sig6`, and `sig1` are the most important predictors, while `sig3` , `sig4`, `sig5`, `is_homepage` are of little importance.



The ranking of importance is also validated by `regsubsets()` in the boosting classifier (as shown in the figure above).

Single Model Classification

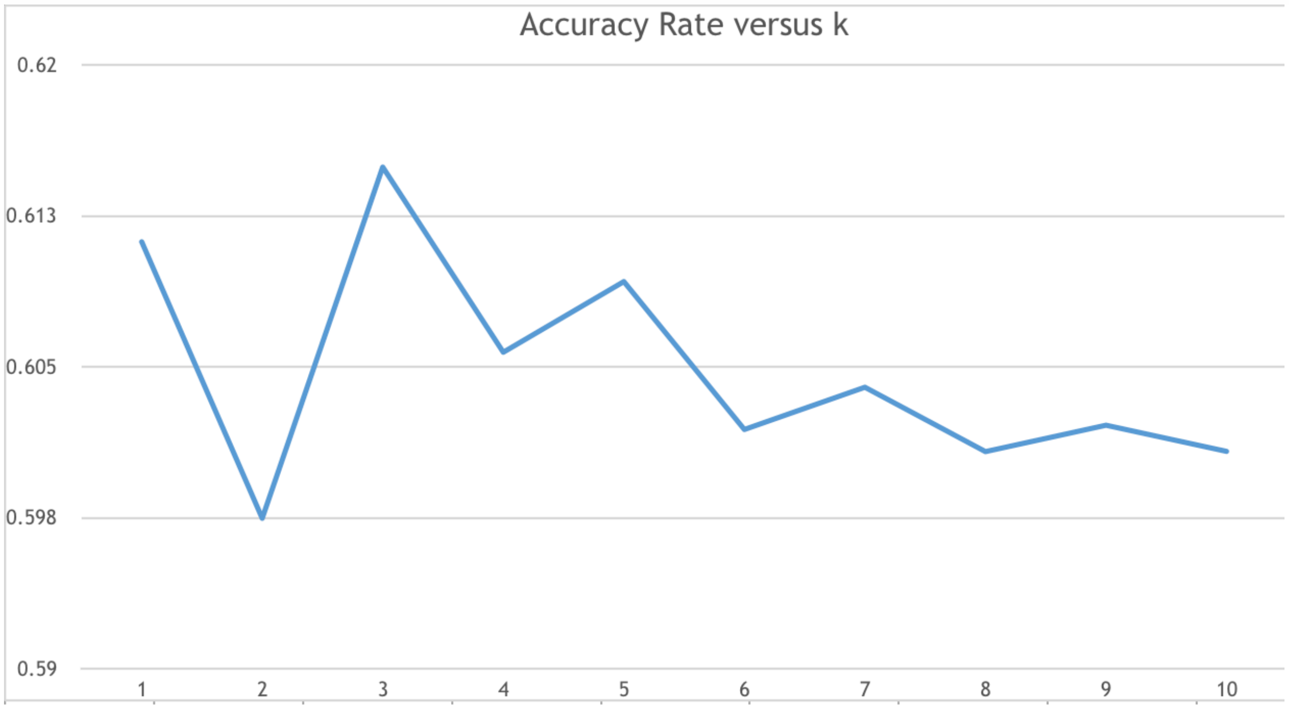
We applied 8 classification models (KNN, Naive Bayes, Decision Tree, Random Forest, Boosting, Support Vector Machine, Logistic Regression, Neural Network) to the dataset, tuned them and measured their performance by 5-fold cross validation. This step provided acceptable single classification models as well as helped feature selection and transformation.

K-Nearest Neighbor Classifier

K-Nearest Neighbor algorithm is a simple and important non-parametric method used for classification and regression. As a type of instance-based learning algorithm, it typically requires sufficient well-rounded training samples, which is satisfied by our dataset.

We used `knn()` from the package `class` to implement K-Nearest Neighbor classifiers. The most important hyperparameter `k` is selected by cross validation. Also, as K-Nearest Neighbor Classifiers can't discriminate predictors, it helps to measure the effects of feature selection and transformation. We achieved the least test error when `sig3`, `sig4`, `sig5` are removed from the vector space. Normalization of the remaining predictors didn't improve the answer, which may indicate the magnitude of the remaining predictors roughly correspond with their importance.

The cross validation result for k is plotted:



Naive Bayes Classifier

Naive Bayes is a a simple technique for constructing classifiers based on probability models. It makes strong (naive) independence assumptions between the features.

We used `naiveBayes()` from the package `e1071` to implement Naive Bayes Classifiers and the test accuracy is about 60.0% after tuning, which suggests the model may be unsuitable for this task.

Decision Tree Classifier

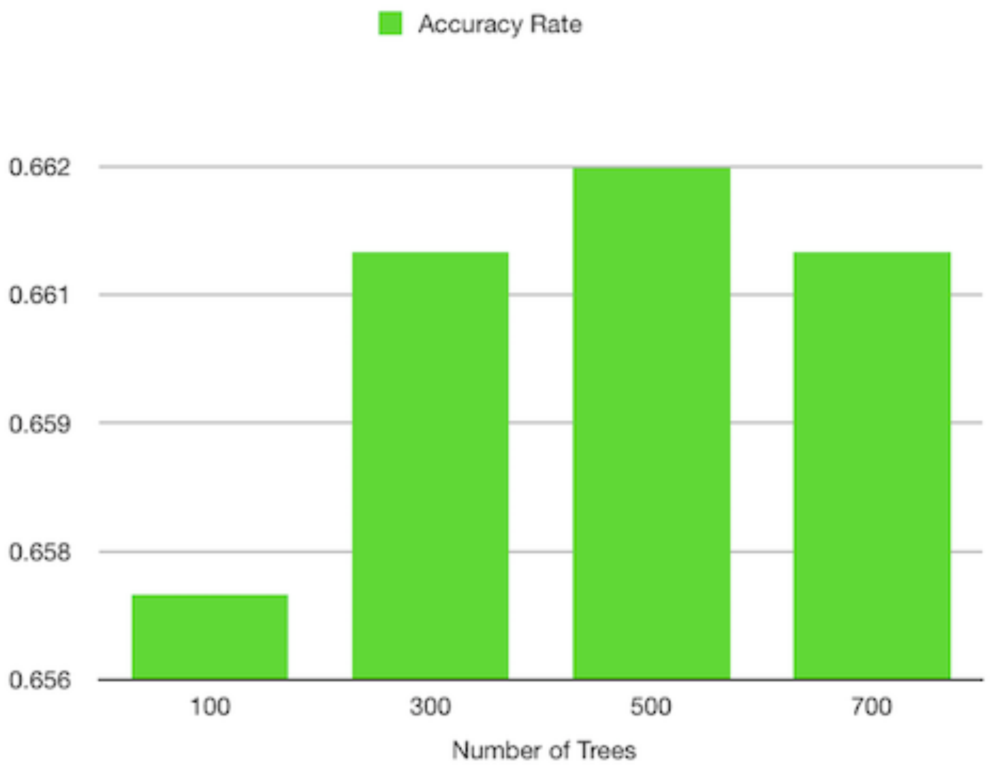
Decision Tree Classifier is a straightforward classification method. It can handle irrelevant variables and is known for great interpretability.

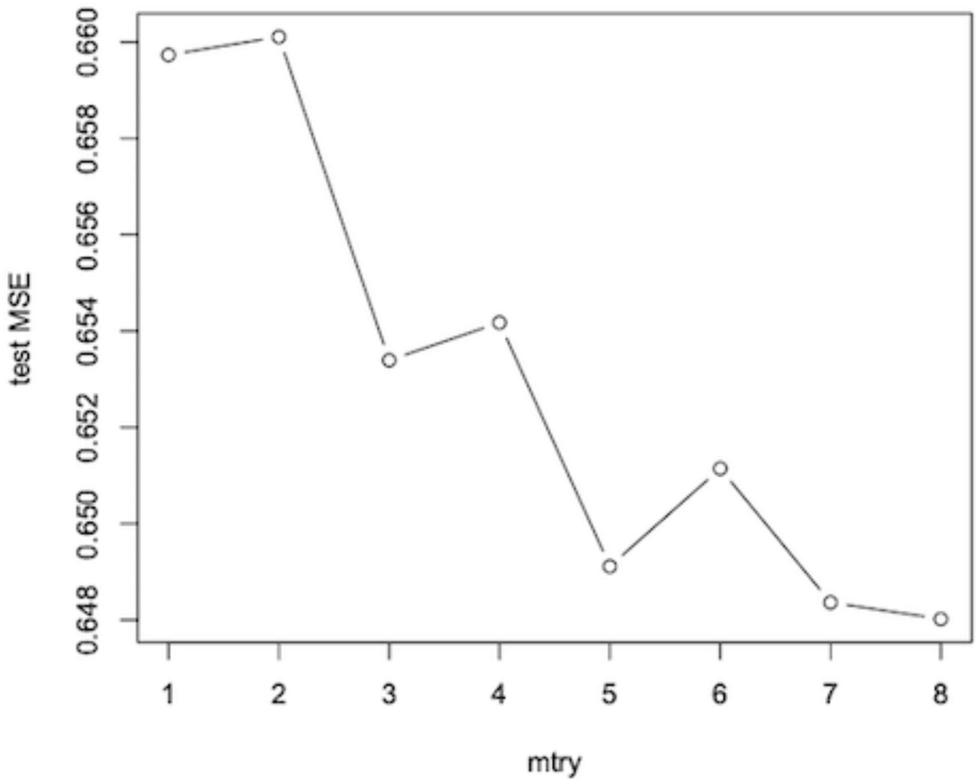
We used the function `Tree` under the package `tree`. The default parameters set gives an accuracy rate of 64.0% using only one predictor, `sig2`. We then tuned `mindev` of the model which limits the minimum deviation among samples within a leaf node. The best test accuracy is 65.0% at `mindev = 0.001`.

Random Forest Classifier

We used the function `randomForest` under the package `RandomForest`.

As the lecture notes suggest, the empirical predictor number used in a single tree is $m = \sqrt{p}$ which is about 3 in this case. We finetuned `m` as well as `number of trees` and the results are shown as follows:



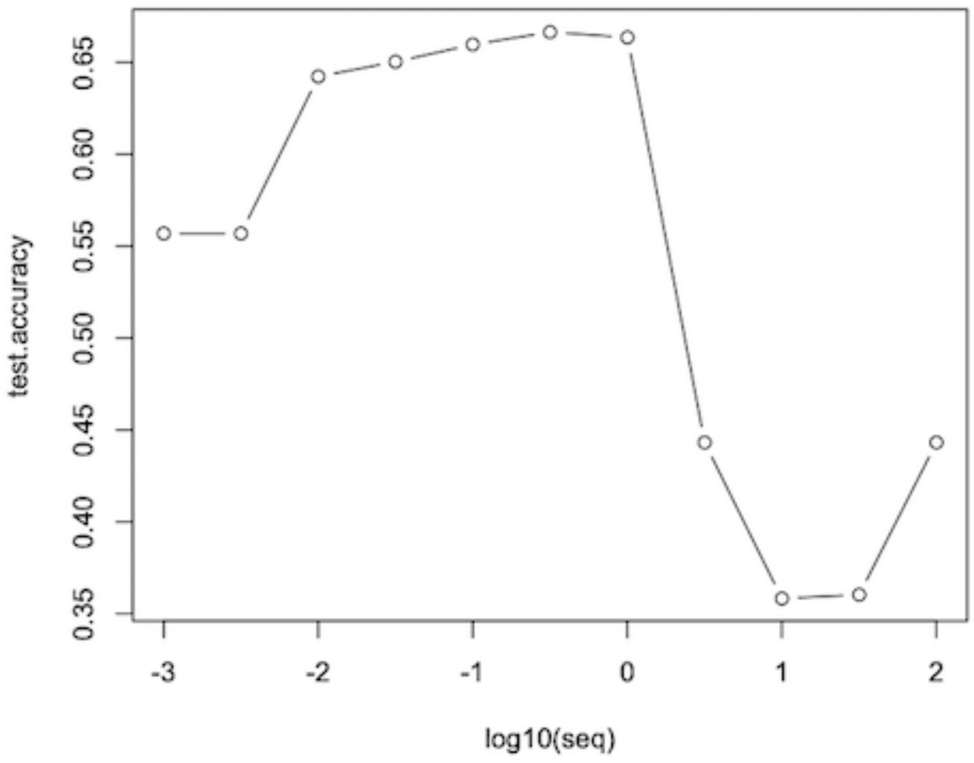


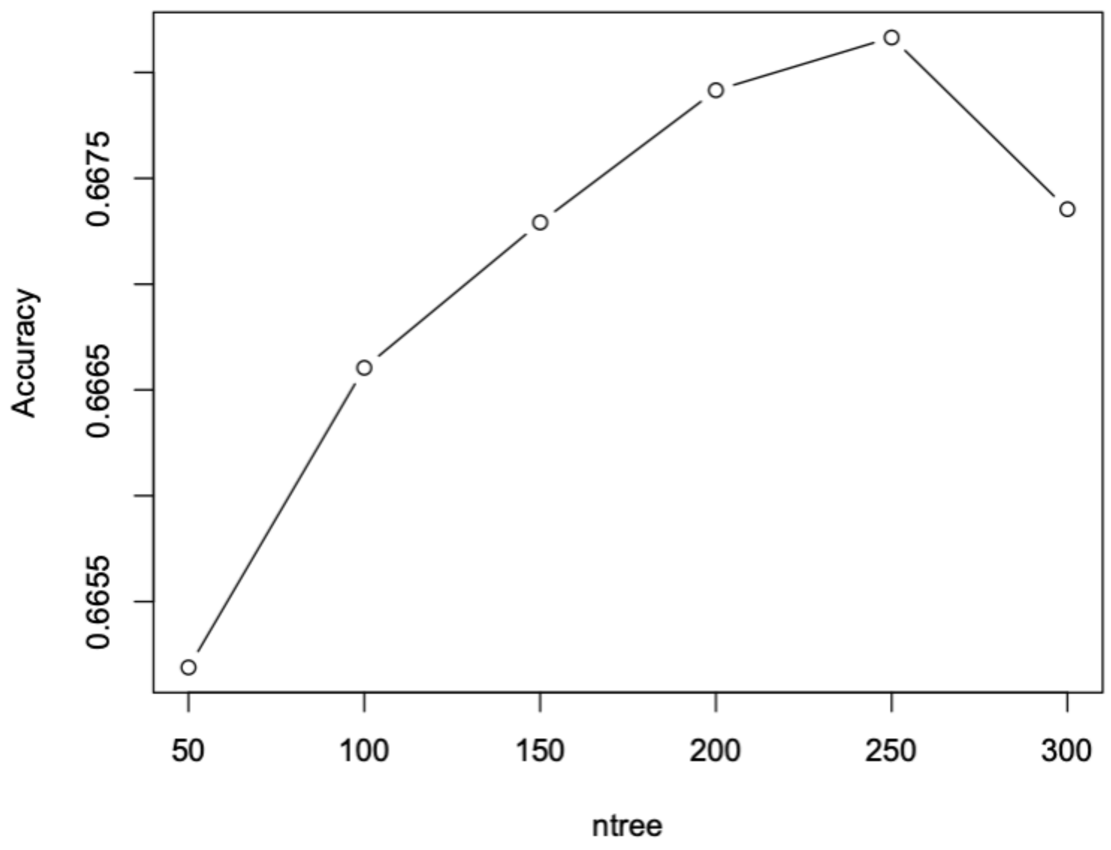
Therefore, we select `m = 2` and `numbers of trees = 500`.

Boosting Classifier

As the "best off-the-shelf classifier in the world", boosting classifiers are assembled from weak classifiers with iterations of training and self-adjusting learning weights.

We used the boosting function `gbm` under the `gbm` package, where decision tree is the base classifier by default. We further tuned `shrinkage` and `number of trees`, two most important parameters for the model.





From the grid search results, we select `shrinkage = 0.1` and `number of trees = 250`

We additionally tried to employ new features to the Boosting classifier.

Support Vector Machine Classifier

Support Vector Machine Classifier is a kind of non-probabilistic binary linear classifier. By selecting the kernel properly, SVMs can efficiently perform linear and non-linear classification.

We used `svm()` from the package `e1071` to train Support Vector Machine Classifiers. Kernel selection is vital to the performance of SVM classifiers for different kernels implicitly mapping the predictors into high-dimensional feature spaces. We tested all the built-in kernels (linear, polynomial, sigmoid, radial) and the results are shown as follows:

Kernel	Test Accuracy
linear	65.18%
polynomial	63.18%
sigmoid	54.69%
radial	66.26%

Logistic Regression Classifier

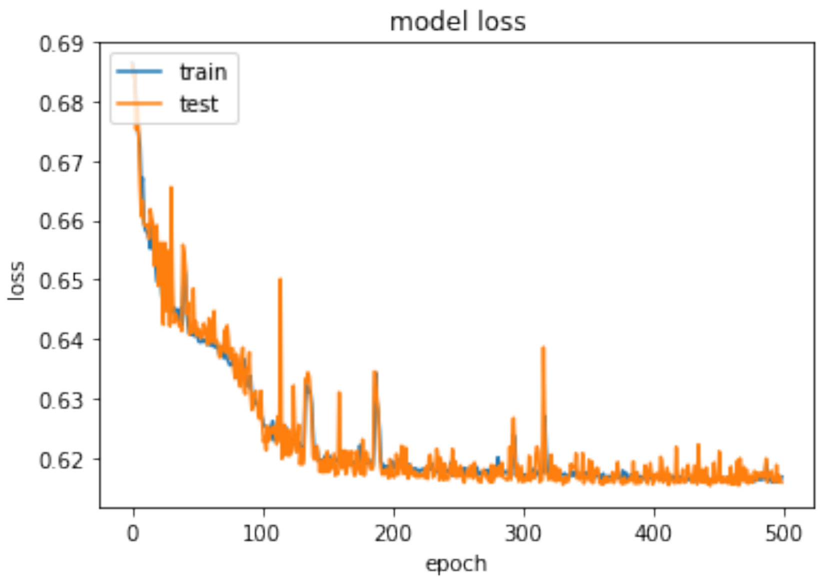
Logistic Regression Classifier is one of the most classic binary classification models. We used the built-in `glm()` to perform logistic regeression.

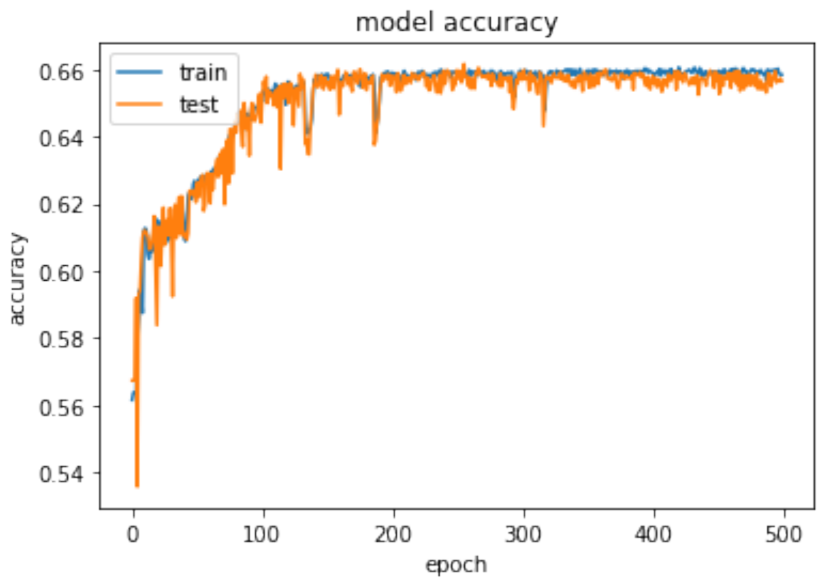
Transformation of predictors is our main approach to tune logistic regression classifiers. We achieved the best performance of when ommited `sig3`, `sig4`, `sig5` and added a cross term `sig1*sig2*sig6*query_length`.

Neural Network Classifier

Artificial neural networks can be used for classification without being programmed with any task-specific rules, which matches our task well.

First we used the built-in multilayer perceptron (MLP) model in scikit-learn to fit the training data, but the test accuracy is comparatively low (~60%). We then constructed a simple neural network with 5 densely connect layers using python package Keras. By tuning activation functions and optimizer, we gained a model with over 0.66 test accuracy. With limited input dimensions, the complexity of neural network is restricted and this may be why it doen't show superiority over the classic statistical learning methods like SVM or Boosting.





The performance of our single model classifiers are summarized as below:

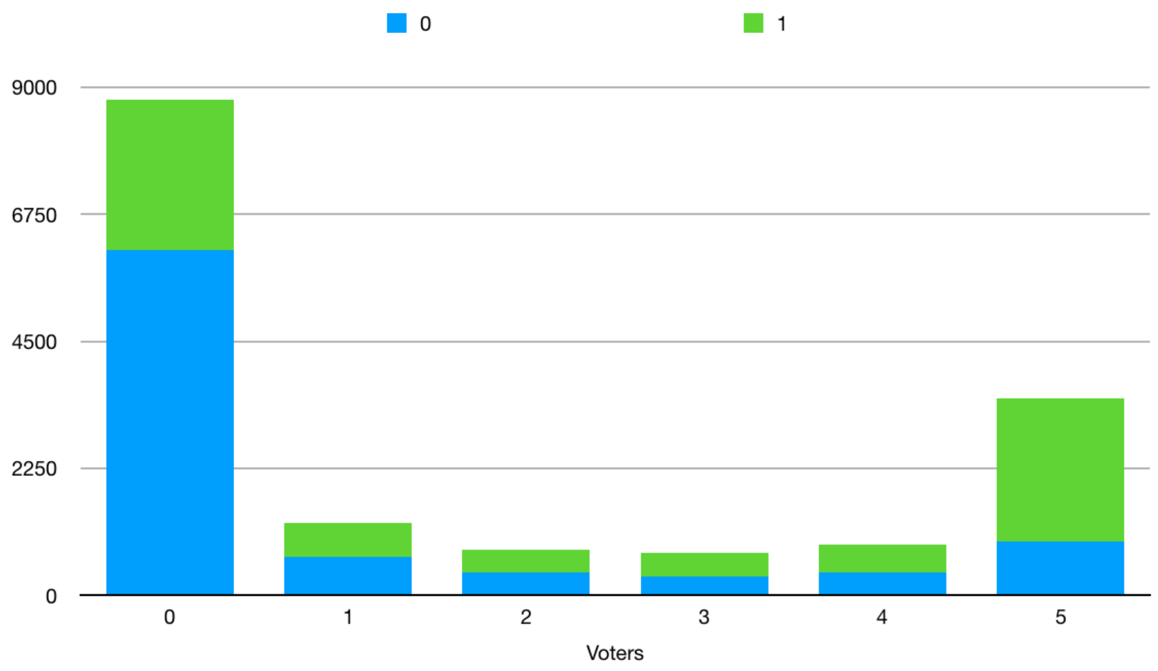
	Model	**Accuracy Rate**
1	KNN	0.615
2	Naïve Bayes	0.594
3	Logistic Regression	0.656
4	Decision Tree	0.650
5	Random Forest	0.662
6	Boosting	0.667
7	SVM	0.663
8	Neuron Network	0.658

Model Ensembling

Now that we have a variety of tuned single classifiers, one possible optimization is to ensemble them. There are several classic and intuitive model ensembling methods and the ensembling process should be cross validated to avoid overfitting. In practice, we mainly considered the single classifiers with comparatively best performance, namely Random Forest Classifier, Boosting Classifier, Logistic Regression Classifier, SVM Classifier and Neural Netwrok Classifier.

Majority Vote

Intuitively, taking a majority vote of differant model's predictions generates a fusion of individual's characteristics.



However, in practice we found that about 22.7% of samples are misjudged by all the classifiers. In other words, the classifiers are not very well-round so that the voting method only yields a medium performance improvement.

Stacking

Stacking (also called meta ensembling) is used to combine information from multiple predictive models to generate a new model. The new model takes the otput of base models as its input and generally outperform each of the individual models.

The reason is that the stacked model is able to highlight each base model where it performs best and discredit each base model where it performs poorly. For this reason, stacking is most effective when the base models are significantly different.

We used `StackingClassifier()` from python package mlxtend to apply stacking method to our single classifiers and the whole process is cross-validated. The stacking method improves three models of test accuracy 64%, 63%, 65% to more than 66%. However, when we include the fine-tuned Boosting Classifier (already 66% test accuracy), the improvement seems to be negligible, which may indicate Boosting itself is already a well-ensembled classifier.

Empirically, model ensembleing tend to yield better results when there is a significant diversity among the models. We then examined the similarity among our single classifiers and found they have strong correlation. Similarity between predictions of Neural Network Classifier and Boosting Classifier is even nearly 0.92. This may be the mainly reason for the only a slight improvement is achieved by stacking.

Conclusion

We selected important variables, trained single and ensembled classifiers and used cross validation to evaulate their performance. Variable selection and fine-tuning improves the performance of non-decision-tree-based models significantly.

However, model ensembling only improve the test accuracy slightly. Correlation among the models may be the main reason, though the structure of the classifiers are quite diverse.