Final Project Report

Yanxiu Chen
Partner: Zhuoran Ying
Stanford ID # 006237072

Abstract

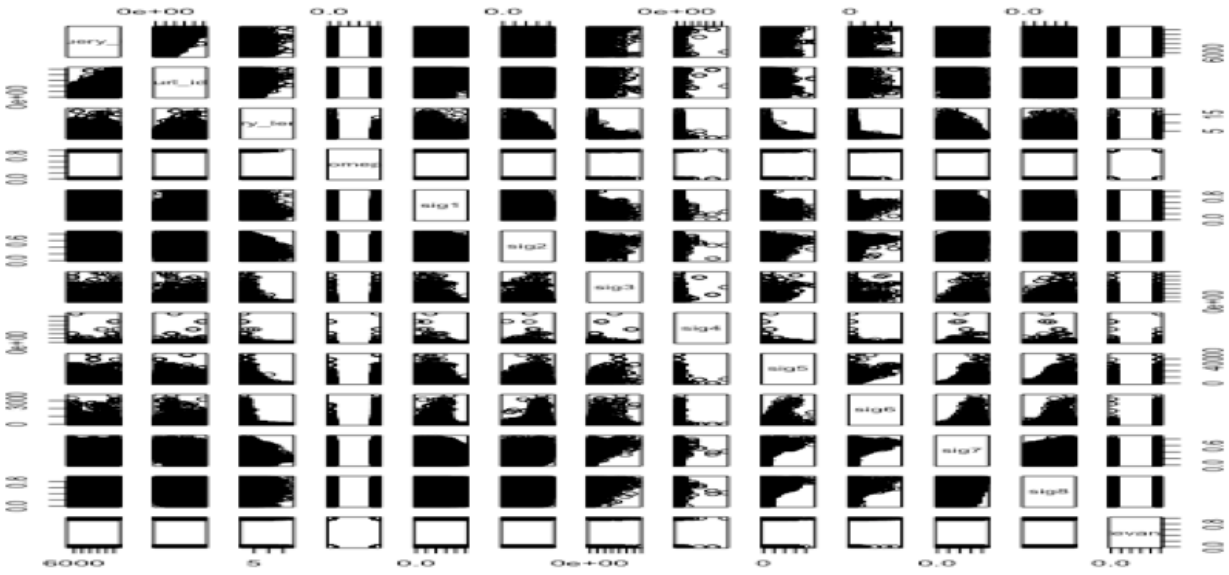
The class project provides features which determine relevant urls for search queries. The main goal of this project is to predict the relevance of urls, given specific search queries. I try five different classifiers: Random Forest, Ada Boosting, Gradient Boosting, Logistic Regression and Support Vector Machine. Gradient Boosting performs the best based on my experiments. This report presents data observations, solution evaluation, candidate solutions and corresponding tuning methods.

Introduction

A training set of query id, url id, 10 attributes, and a response variable is provided. 80,046 observations are included. Binary response variables are labeled as 1 for relevance and 0 for irrelevance. A test data with 30,001 observations is used for a final evaluation of the project.

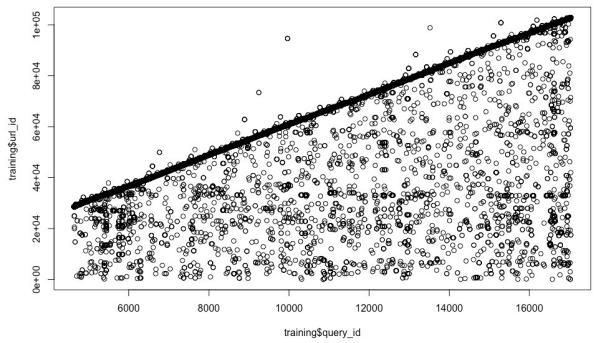
Data Observations

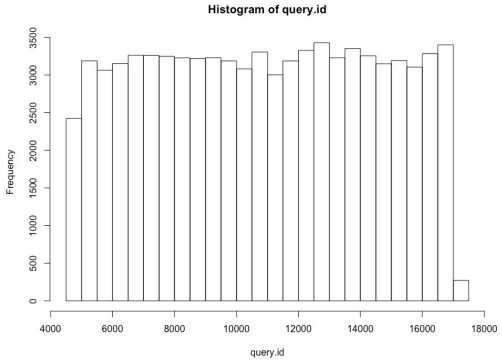
Both training and test data files have been examined and no missing data are found. There are about 43.7% relevant url for search queries in the training set. There are ten attributes, one query id and one url id for each observation. To understand the relationship between predictors, a scatterplot matrix is created.



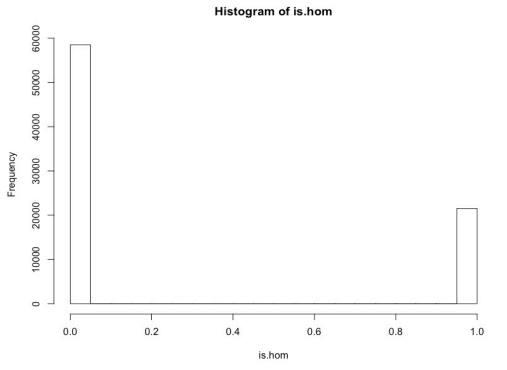
The scatterplot presents a strong linear relationship between “query_id” and “url_id.” The correlation between “query_id” and “url_id” is abnormally high, 0.9061758. However, given url IDs are randomly assigned, this strong relationship may not directly reveal the property of query. To further investigate, a histogram of query id is plotted. I observe that the same query id repeats across the data sets. Since the

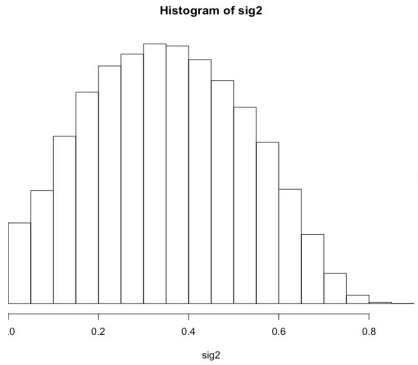
same query id has the same query length, the same “query_id” observations may refer to one query. “url_id” may refer to the url that is connected with query. The search engine may return multiple urls given a specific query. However, the number of urls per query differs, which may reveal the difficulty of finding relevant urls for query. “query_length” may refer to the amount of words for each query rather than the character a query has, since the minimal length is 1. “query_length” has 18 levels.

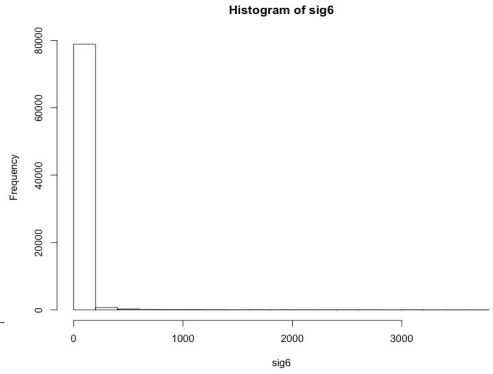




Given that response variable is binomially distributed, I am particularly interested in variables that have similar distribution. Note “is_hom” only has two values: 0 for “no” and 1 for “yes.” The histogram of “is_hom” is plotted below. The range of “sig 2” is between 0 and 1. The histogram of “sig 2” is plotted and it is normally distributed with a little right skewness. A high proportion of “sig 6” concentrates on 0; therefore, “sig 6” may be an important signal. “sig 1” has a right-skewed distribution. “sig 3,” “sig 4,” and “sig 5” are randomly distributed. “sig 7” and “sig 8” are roughly normally distributed with right skewness.







Outliers and leverage points are detected in training set. Ideally, we should have removed some extreme observations. Here given the sufficiently large normal observations, these steps could be skipped. In addition, because different tuning and feature selections methods are applied, these extreme observations should not have huge impact on the classification models and outcomes.

New Features

New features are created to understand the property of query and the relationship between query and url. 6 features are derived from url id based on the same query id except “url_appearance.” 24 variables are tailored for each signal because signals determine the relationship between query and url. Given a query

id, new variables are created to measure the ordinal ranking, mean, and max for each signal. For example, "sig1_rk," "sig1_mean," "sig1_max" are created for sig 1. The same applies to "sig 2" to "sig 8." In total, 30 new features are created. They increase the flexibility of the model. Data.table method is applied here.

"url_id"

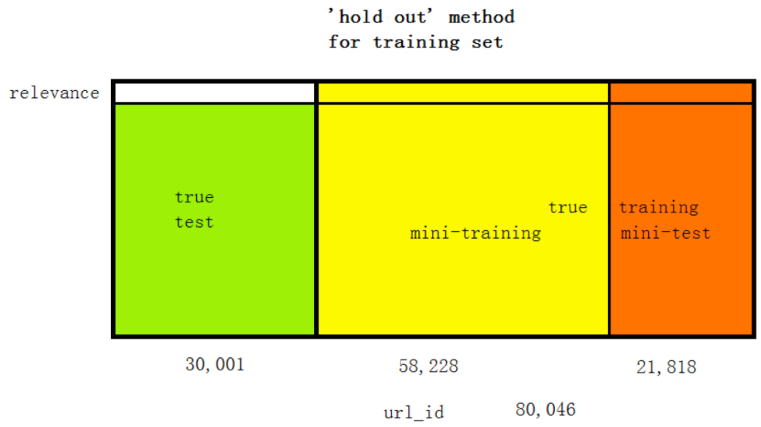
url_per_query	the number of url per query, measuring the sample size of query id group
url_median	the median of url id per query, measuring the central tendency of query id group.
url_distance	the log difference between url_median and individual url id
url_appearance	the frequency of the url id in the training data set
url_otherSuccess	the frequency of url that is relevant (relevance =1) for other query id, given a specific query id
url_otherSratio	the ratio of "url_otherSuccess" to "url_appearance"

"sig i"

sig i_rank	the ordinal ranking of each signal, given the same query id
sig i_mean	the mean of each signal, given the same query id
sig i_max	the max of each signal, given the same query id

Solution Evaluations

We apply two approaches to evaluate models. First, we use 10-fold cross validation based on randomly selected observations from the training set of 80,046 observations. Second, on the top of the cross validation result, we apply an extra out-of-sample test on a mini-train/test split with continuous query_ids for the top 3 performing models. The figure below illustrates the setup.



We split the training data into mini training (58,228) and mini test (21,818). The ratio of mini test to mini training is 37.5% (21,818/58,228), which matches that of the true test/ training ratio (30,001/80,046 = 37.5%). The extra test is based on the idea that randomly selected cross validation may not represent the true prediction of test set which has url_ids assigned in the latter stage. Both criteria will be evaluated in the following models.

Candidate Solutions and Tuning

We summarize the best misclassification rates below. **Gradient boosting, Random Forest and Support Vector Machine** are the top 3 models after we tune the methods through cross validation and out-of-sample evaluation. We are pleased to see that the error rates of two evaluations are consistent with each other, even though out-of-sample error rates exceed the cv errors slightly.

Classifier		**Test Misclassification Rate**
Random Forest	Number of Trees = 500	30.78%
Ada Boosting	With Decision Trees	32.72%
Gradient Boosting	With Decision Trees	30.20%
Logistic Regression	Lasso	33.90%
Support Vector Machine	Radial Kernel	32.08%

I. Random Forest

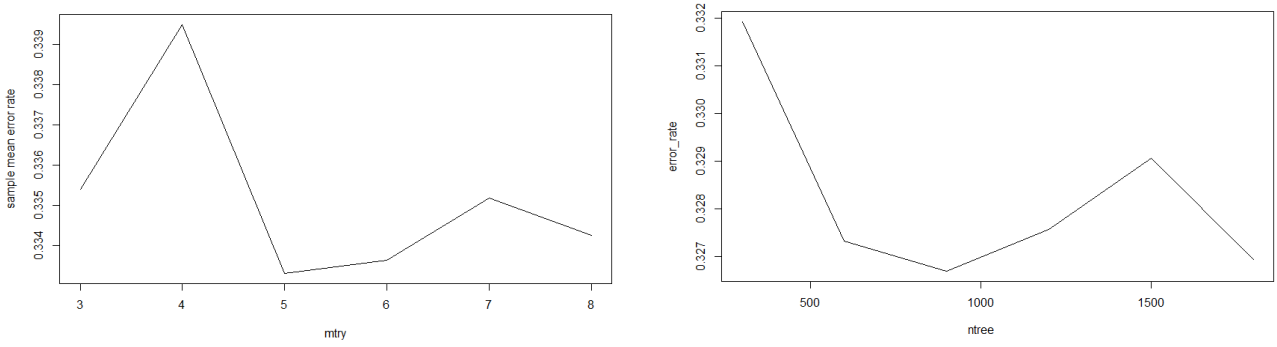
Random Forest decorrelates the trees through creating multiple independent trees, and then benefits from averaging these sub-trees. It increases variance in the training set by only considering a subset of predictors in each split. We use the randomForest() function in the package "randomForest" for the actual classifications. The output provides OOB estimate of error rate, 31.01%, which is helpful to estimate test error of this bagged model.

Tuning

"mtry" and "ntree" are tuned for this model. Since we have 42 predictors, we use the default value of mtry as 6, which is $\sqrt{number\ of\ predictors}$. The left graph below shows that the best parameters turn out to be "mtry" between 5 and 6, which can be explained by the collinearities between predictors. Then we tune the model by performing a 10-fold cross-validation and find that the optimal ntree is near 1000. Due to the limited time and computational ability of our devices, we do not build the model with ntree =1000 which will result with 0.1% difference from ntree =500. In short, we use **mtry = 6 and ntree = 500**.

mtry ntree

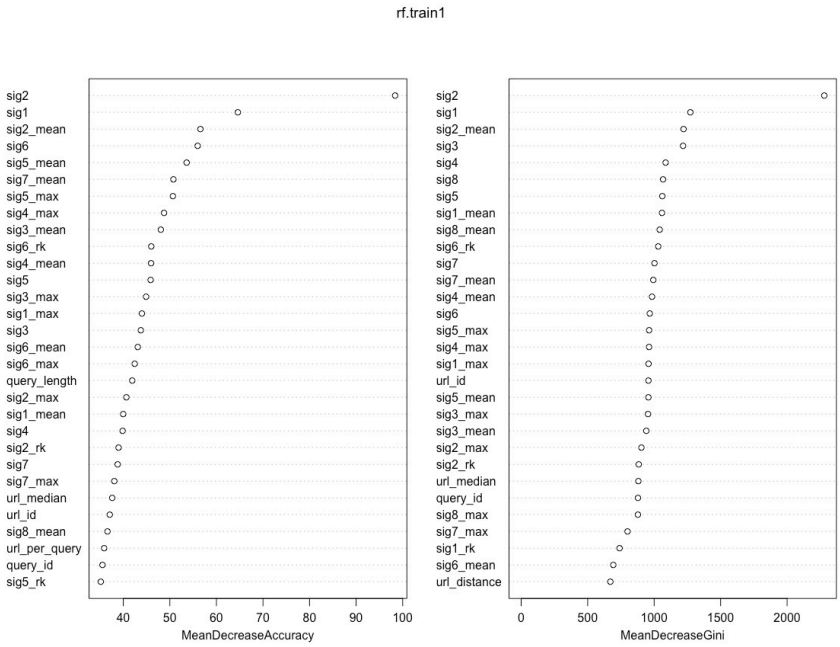
Interpretation



After tuning the model, Random Forest classifier achieves a superior performance: cv error rate of **30.78%.** It is the second lowest cv error rate in our project. The nonparametric nature of random forest contributes to its performance. We may further improve the model by omitting one variable. Because we

have limited time and random forest is not the best model, we do not proceed with further fine tuning. To further understand the model, we use the importance () function to view the importance of each variable. "**sig 2**" is the most important factor in determining relevance and new features such as "**sig 2_mean**" also demonstrate a significant effect.

graph of the importance

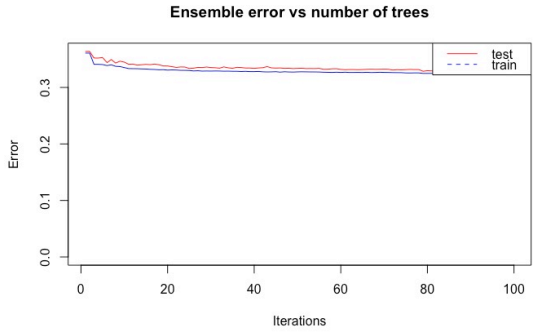


II. Boosted Decision Tree

Boosting is a popular classifier. We use two boosting algorithms for this project, because they are slightly different from each other. Ada Boosting builds the model by learning from high-weight data points while Gradient Boosting learns through the differences between predicted values and real values. Boosting upweighs points of misclassified observations at each iteration and therefore solves the existing problems in order to optimizes the model.

Ada Boosting

We use the function boosting() in the "adabag" package to build the training model. From the graph below, we witness how the model improves through benefiting from averaging each tree and taking the majority vote of it. The 10-fold cv error rate is **32.72%**.



Tuning

"mfinal," the number of iterations, is tuned in this method. Given k=3, max_depth=6, mfinal = 100 gives a better misclassified rate, 31.79%. As a result, in the function boosting(), we set coeflearn = "Breiman," mfinal = 100, max_depth = 30.

mfinal	Misclassified Rate
10	33.6%
100	31.79%

Gradient Boosting

Gradient boosting sequentially grows the trees through optimizing the loss function. Loss is the difference between predicted response and real response. We use the package "xgboost" in this project.

Tuning

We tune the model by reading instructions on [3] from the "caret" package. train() function is used here, and it provides AUC and Accuracy. We subtract Accuracy from 1 to get error rate, which is used as our evaluation criteria. The tuning procedure is presented in three parts.

To begin with, we build a base model by using a high learning rate "eta" = 0.1. It enables the model converges fast for cross-validation. After some try-and-errors, initial parameters are chosen. Our base model is shown in the table below. It is very impressive that the base model provides an error rate of 30.83%. A grid search method is used to tune the 7 parameters.

base model						
nrounds	eta	max_depth	gamma	colsample_bytree	min_child_weight	subsample
300	0.1	6	0	0.8	1	0.8
cv accuracy						
AUC	Accuracy	error rate				
0.7497	0.6917	0.3083				

Second, "max_depth" and "min_child_depth" are tuned because they are the most important parameters. A grid formed by max_depth = c(4,6,8,10,12) and min_child_weight = c(1,3,5) is searched. The optimal parameters are shown below.

xgb_train_1$bestTune						
nrounds	eta	max_depth	gamma	colsample_bytree	min_child_weight	subsample
300	0.1	8	0	0.8	1	0.8

Then we tune "subsample, "colsample_bytree," and "gamma" by using the same grid search. The results are in the table below.

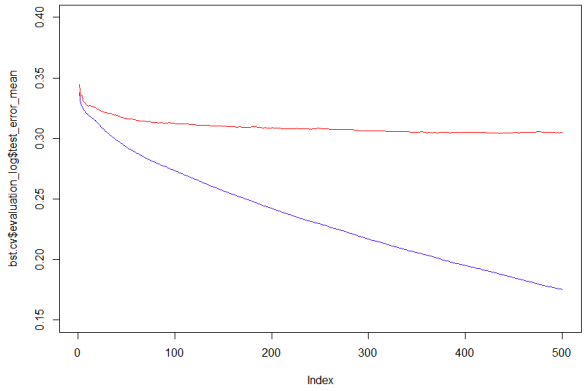
xgb_train_3$bestTune						
nrounds	eta	max_depth	gamma	colsample_bytree	min_child_weight	subsample
300	0.1	8	0	0.6	1	0.9

Thirdly, "eta" is lowered to 0.01 and "nrounds" is boosted to 2000. The model learns very slowly but the result is worth. The significant improvement can be found in the figure below (Improvement over iterations). As a result of tuning, the cv error rate decreases from initial 30.83% to 30.19%. This is the best model we have!

xgb_train_final$results						
nrounds	eta	max_depth	gamma	colsample_bytree	min_child_weight	subsample
2000	0.01	8	0	0.6	1	0.9
cv accuracy						
AUC	Accuracy	error rate				
0.7574	0.6981	0.3019				

The figure below shows the satisfactory improvements over iterations. We witness a significant decline of training error over iterations, which is much steeper than adaboost. The best cv error rate achieves **30.20%**.

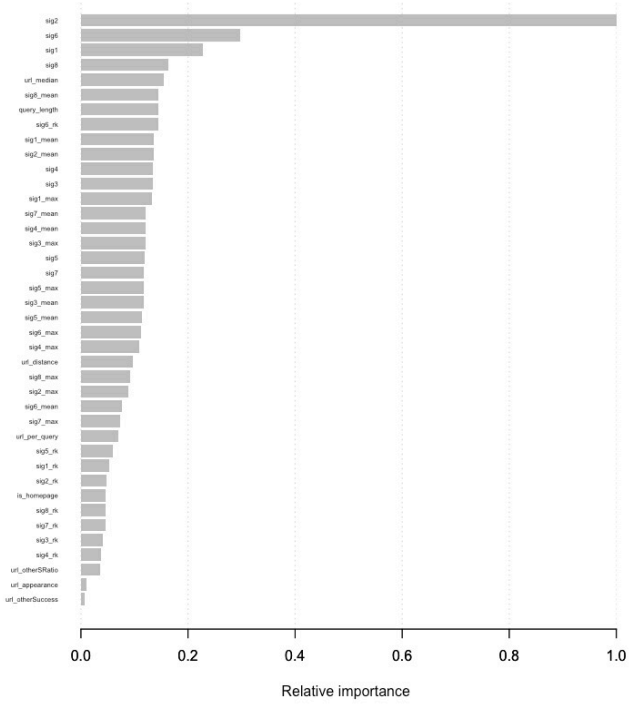
Improvement over iterations



Interpretation

We then understand the model through xgb.importance(). "**sig 2**" is shown the most important predictor again following by "sig 6" and "sig 1."

Important predictors



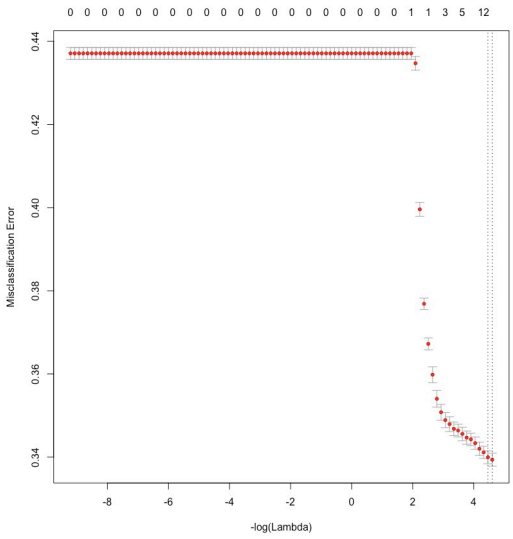
III. Logistic Regression

Logistic regression is a robust model for binary response. Parameters in the model are estimated by maximum likelihood. By using the function glmnet() and setting alpha as 1, we build the classifier incorporating lasso to select predictors. Lambda are given values from 10^-2 to 10^4.

Tuning

We find the best lambda, **10^-2**, through cv.glmnet(); it has the minimum cv error. The left graph shows the misclassification error and –log (Lambda). The best lambda is then applied in predict() for feature selection and 15 variables are selected. The results are satisfactory because coefficient estimates are sparse; the lasso shrink relatively unimportant variables to 0.

Graph of Error Rate



Graph of Important Variables

```
44 x 1 sparse Matrix of class "dgCMatrix"
                             1
(Intercept)      -1.866464e+00
(Intercept)       .
query_id          .
url_id            .
query_length      3.684095e-02
is_homepage       .
sig1              9.632590e-01
sig2              2.977980e+00
sig3              .
sig4              .
sig5              .
sig6              1.393871e-03
sig7              7.821605e-01
sig8             -4.585982e-02
url_per_query    -3.869642e-02
url_median        .
url_distance      .
url_appearance    .
url_otherSuccess  .
url_otherSRatio   1.401310e-01
sig1_rk           .
sig2_rk           .
sig3_rk           .
sig4_rk           .
sig5_rk           .
sig6_rk           1.444953e-01
sig7_rk           .
sig8_rk          -7.838953e-03
sig1_mean         .
sig2_mean         .
sig3_mean         .
sig4_mean         .
sig5_mean         .
sig6_mean         .
sig7_mean         .
sig8_mean         .
sig1_max         -3.109097e-01
sig2_max          .
sig3_max          .
sig4_max          .
sig5_max         -6.424667e-06
sig6_max         -9.485281e-05
sig7_max         -1.872159e-01
sig8_max         -2.504240e-02
```

Interpretation

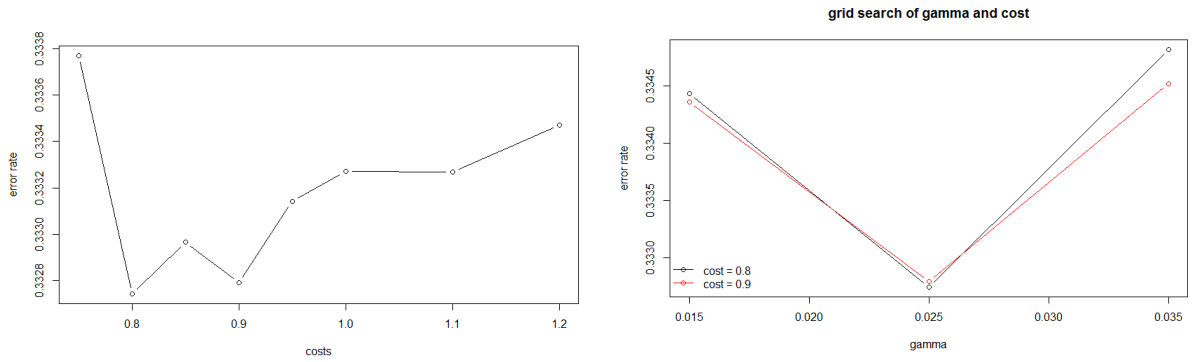
One drawback of this model is the lack of flexibility, which makes the decision boundary hard to fit into our data. Therefore, the model has a relatively high average cv error rate **33.90%.** Another disadvantage is the inconvenience of modifying interactions between variables. The coefficients become unstable when there is collinearity.

IV. Support Vector Machine (SVM)

SVM has a modest performance in this data set. It has advantage over logistic regression because the classes are separated in this data set. We do not perform feature selection for SVM because it is an approximate implementation of a bound on the generalization error. It is independent of the dimensionality of the feature space. Package “e1071” is used in this project. Due to the limited time and high dimensional feature space, radial kernel is used in the classifications. CV error rate is **32.08%**.

Tuning

tune() in the package "e1071" is used to find the best value of "cost" and "gamma." "cost" = 0.8 and "gamma" = 0.025 provide the best parameters.



Conclusion

Among the five classifiers, gradient boosting and random forest are chosen to be the best models. We use these models to predict relevance for test data. The results are in the txt files attached. rf.txt is the result of randomForest and xgb.txt is the result of gradient boosting.