BASIC WEB QUERY CLASSIFIER

Author: Christine Garver

About This Project

This project was a final project for a college course. Datasets were provided by the professor. This report was written by the author and the R code was written by the author and another contributor.

Abstract

The objective of this project is to create a model that will accurately predict whether a particular document is relevant or not based on various predictors. Therefore, this is a classification problem. While one classifier is chosen to submit, 5 classifiers in total are experimented with on the training data in order to properly pick the best model. Logistic regression, ridge regression, support vector machines (SVM), random forests, and ada-boosting are performed on the given data. Random forests produce the best results, and is therefore chosen as the classifier for submission.

1. Introduction

The training data set includes 80046 observations and 10 parameters for the output of interest, relevance. No data seems to be missing either. Having a lot more observations than parameters will be important in assessing accuracy of certain models. Relevance is recorded as 1 if relevant and a 0 if NOT relevant. In order to run experiments to determine which model to use as the final classifier, the training data is first split into a "train" subset and a "test" subset. The "train" subset includes 56032 observations (70% of the training data) and the "test" subset includes 24014 observations. Throughout the rest of this report, the "train" subset will be referred to as the training data and the "test" subset will be referred to as the testing data. All models in this experiment are trained on the training data, but then evaluate the testing data, giving us a measure of error for that model.

2. Data Observation

Before choosing the classifiers to train on the training data, the training data is observed in order to see if certain patterns or relationships exist. First, the correlation between the parameters and relevance are computed. Two relationships stand out the most - there is a very high correlation between query_id and url_id as well as between sig3 and sig5:

correlation between query_id and url_id = 0.906

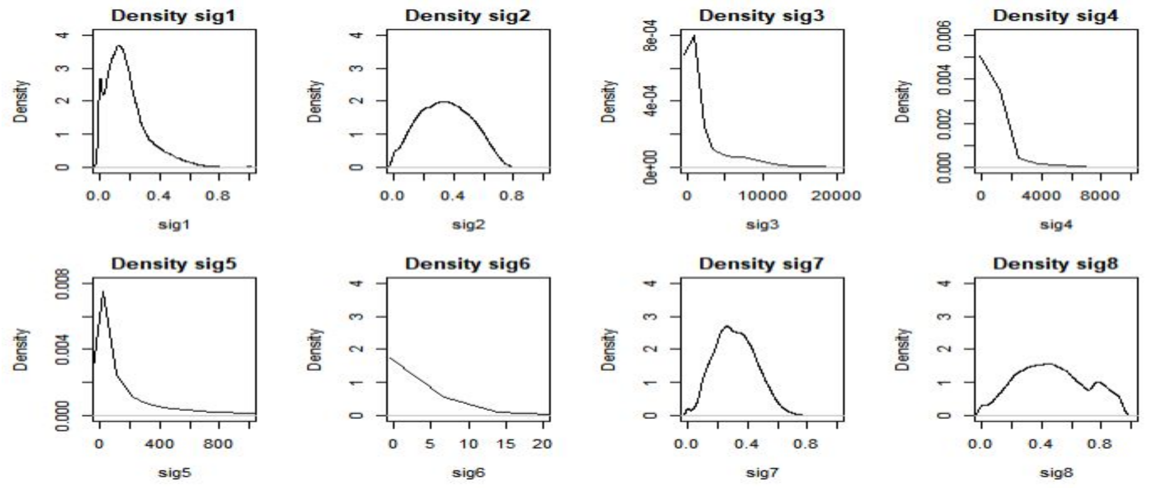
correlation between sig3 and sig5 = 0.815

The other parameter correlations do not have significant correlations. Given a high correlation, the variance inflation factor (VIF) is then calculated in order to determine whether there is a possible collinearity relationship. If given a collinearity relationship (VIF > 5), most likely not all parameters will be significant in generating the best model:

```
        res = lm(relevance~., data=query.train)
        vif(res)
                query_id        url_id query_length
                6.124051        6.263718        1.112612
         is_homepage            sig1     sig2
                1.507627        1.128366        1.306263
                sig3     sig4   sig5
                2.749719        1.080541        3.408631
                sig6     sig7   sig8
                1.323191        1.630973        1.435130
```

Both query_id and url_id generate a high VIF, possibly signaling that there is a colinearity relationship between these two parameters. Therefore, when generating a linear regression model, one would generate one model with all parameters, and then maybe another without query_id or url_id.

After evaluating correlation and VIF on the training data, distribution graphs are plotted for each sig variable in the training dataset:



Sig2, sig7, and sig8 appear to have the most normally distributed data, and all density plots are treated as continuous parameters. It is important to note that because the parameters do not all have the same distributions, some classifications generate more accurate results if parameters are scaled before training. This will be covered in more detail when each classification model is discussed.

3. Solution Evaluations

Five different models are experimented with in this project. For each model, the data is FIRST split into the training set and the test set. The particular model is trained on the training set, then allowing one to calculate test error rate. It is noted that there are 12408 unique query IDs. However, the data was just split by placing the first 56032 observations in the training set and the rest in the test set. This places 8723 unique query IDs in the training set and the remaining in the test set.

4. Candidate Solutions and Data Selection

Even though query_id and url_id have a significant VIF, no parameters are eliminated before performing any classification. This is because certain models have parameter elimination properties, and the models that do not will be trained another time, eliminating some variable(s). The table below lists the classifier and its optimal calculated error on the test set. Further detail in how the classifier is implemented and why it was implemented will follow the table.

Classifier	**Error**
Logistic Regression	0.365
Ridge Regression	0.348
Random Forest	*0.338*
SVM	0.345
AdaBoosting	0.353

a. **Logistic Regression**
   Logistic regression is one of the more basic classifications, but it allows one to model a classification problem by using the likelihood function. While this is a more simple classifier to understand and implement, it can be inaccurate if there is collinearity. After splitting the data into training and test set, the "glm" function is used to train the model using the training data and all parameters, for scaling of the variables is not necessary. The family parameter of the model is "binomial," making the model a logistic regression. Then another logistic model is trained on all parameters except query_id (this is called fit) and another logistic model is trained on all parameters except url_id (fit2). An anova table with Pearson's Chi-Square test is generated in order to determine which model is a best fit.

```
Analysis of Deviance Table

Model 1: as.factor(relevance) ~ url_id + sig1 + sig1 + sig3 + sig4 + sig5 +
    sig6 + sig7 + sig8 + is_homepage + query_length
Model 2: as.factor(relevance) ~ query_id + sig1 + sig1 + sig3 + sig4 +
          sig5 + sig6 + sig7 + sig8 + is_homepage + query_length
Model 3: as.factor(relevance) ~ query_id + url_id + query_length + is_homepage
+ sig1 + sig2 + sig3 + sig4 + sig5 + sig6 + sig7 + sig8
  Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
1     56021      72939
2     56021      72939  0    -0.15
3     56019      69887  2  3052.27 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The model with all parameters appear to be most significant, so therefore an error rate is calculated for this model, which ends up being 0.365.
The confusion matrix is below:

```
          truth
predict       0      1
      0   12760   7954
      1     810   2490
```

b. Ridge Regression

Ridge regression is chosen as one model for a few reasons. One, ridge regression fits models quickly and is a shrinkage method. Therefore, no parameters need to be eliminated before training the model. It also benefits in the bias-variance trade-off, for as lambda (the tuning parameter) increases, the flexibility of the model decreases (meaning variance decreases, bias increases). Through cross-validation/tuning of lambda, an optimal model can be fit. Unlike Lasso (another shrinkage method), ridge regression, uses all parameters in the final fit. While this makes interpreting the model more challenging, the accuracy of the model tends to be greater than the Lasso (if most parameters are non-zero values).

After splitting the data into a training and test set, cross-validation is performed on the tuning parameter, lambda, using “cv.glmnet” only on the TRAINING set. It’s important that one tunes lambda only on the training data, in order to more accurately determine how well the model predicts data that it is not trained on (i.e. the test set). After selecting the best value of lambda, ridge regression is performed on the training set using “glmnet” and setting relevance to “as.factor(relevance)”. The “glmnet” default scales the parameters variances to 1 before performing the training, which is important in that is prevents lambda from penalizing some parameters more than others. Now the trained ridge regression model takes the data from the test set and predicts the relevance. The actual and predicted relevance values are compared. The test error rate is calculated to be

0.348 when implementing "ridge = ifelse(ridge.pred>.5, 1, 0)". Therefore, the ridge regression model has a test error rate of about 34.8%. The confusion matrix is shown below:

```
table(prediction=ridge, truth=test$relevance)
          truth
prediction      0       1
          0 10663  5453
          1  2907  4991
```

c. Random Forests

Random forests is a cool classifier in that is similar to bagging, but better, and it involves trees and bootstrapping in one. By generating different trees with random parameters (not all parameters are used), there is less correlation between the trees, generating more accurate results that just bagging trees. Also, contrary to ridge regression, random forests is also easy to interpret due to its tree properties. However, the model takes more time to fit.

After splitting the data into the training and testing set, the number of parameters is chosen to be 3, since this is a classification problem (3.46 = 12^(½)). The model is trained on the training set using "randomForest" R package. The output value, "relevance" is calculated with "as.factor(relevance)" in order to ensure the random forest model is a classification problem, not a regression problem. Scaling the variables is not necessary for random forests, nor is eliminating variables. After the model is finally fit, it predicts the relevance of the test data which is compared to it's true value. The OOB error is calculated to be 0.338 and the test error rate is also 0.338.

```
Call:
 randomForest(formula = as.factor(relevance) ~ ., data = train,  importance =
TRUE, mtry = 3)
          Type of random forest: classification
                    Number of trees: 500
No. of variables tried at each split: 3

          OOB estimate of  error rate: 33.82%
Confusion matrix:
          0          1 class.error
0 24658  6831   0.2169329
1 12117 12426   0.4937049
```

Since random forests do not use all parameters when creating each tree, it is not necessary to eliminate parameters before training the model. Out of curiosity, after looking at the importance of each variable in the tree, “is_homepage” has the lowest Gini index.

d. SVM

Support vector machines allow one to fit a model for non-linear decision boundary lines. By tuning the cost parameter, overfitting can be controlled as well. The “e1071” package is used to implement SVM. After splitting the data into training and test sets, the cost parameter is tuned in order to determine which cost value will produce lowest error rate (“tune” is used to tune the cost parameter). Then the SVM model is trained on the training data using a linear, radial, and polynomial (degree = 2) kernel. In this experiment, the variables are not normalized, but they should have been, for SVM could perform better when variables are scaled. Training the model takes a good amount of time, but after it is trained, the model predicts the relevance for the test data. The actual and predicted values are compared and the calculated test errors are in the table below, with the best error rate of 0.345.

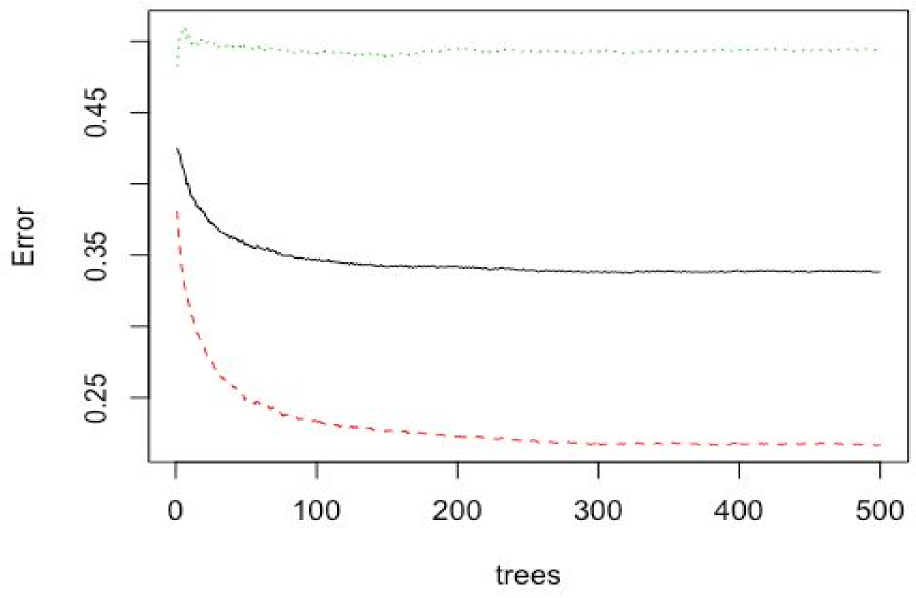
Kernel	**Error**
Radial	0.345
Linear	0.345
Polynomial (degree = 2)	0.399

e. AdaBoosting

Boosting has many advantages in that it uses averaging (like random forests) and upweights misclassified points at each iteration. An AdaBoosting algorithm is written in R (based on the lecture notes), using 400 iterations and the package “rpart”. After splitting the data into training and test data. The AdaBoosting model is trained on the training data, and since boosting accounts for parameter limitations, all parameters are used in the training of the model. Then the model predicts the values of the test set and the actual and predicted values are compared. The test error rate for AdaBoosting is 0.353.

5. Fine Tuning

Random forests generates the best results in that it has the lowest *test* error rate. The test error is more important than the training error, so the chosen classifier is based on test error. Therefore, the random forest model is used to classify the test file. While random forest does not eliminate any parameters, it only uses a proportion of the parameters when generating each tree, making the model fairly accurate. Therefore, no parameters are eliminated before training the random forest model, nor are variables scaled. The randomForest package has a default of using out-of-bag cross-validation, so no tuning is used in this random forest classifier. Increasing the number of trees could have had an affect on the error, but it most likely would have been minimal, for when looking at the graph below, the error levels off around 300 trees.



6. Summary and Improvements

Overall, the models are all trained on 70% of the observations and then classify the other 30% in order to determine the test error rate. All error rates fall below 40%, but this does not necessarily mean the classifier will perform at the test error rate. Only for logistic regression, were parameter eliminations performed. However, some classifiers, such as ridge regression and random forests have properties that limit parameters. Tuning of parameters using cross-validation, such as lambda and cost, were important in finding the optimal value of a particular model. Random forest test error rate performed the best, so the random forest model is the submitted classifier.

As in all experiments, there are inherent errors and room for improvements. First, there will be irreducible error, an inherent error in our data or classification method that we cannot change due to randomness and variability in the data/query search itself. However, through splitting the data a particular way, tuning parameters, using cross-validation, it's possible to minimize other errors.

One improvement if this experiment were to be redone, would be to use AIC or cross-validation to determine which parameters are most important before performing logistic regression. While logistic regression was attempted by just eliminating one possible insignificant parameter and then computing a chi-square value, cross-validation would provide a more robust choice of parameters, opposed to just testing out one classifier. Another improvement would involve creating a hold-out group when training the models and making sure to scale the variables before performing SVM.

Also, further observing the relationship between parameters would be interesting in trying to determine what the particular signals represent. While this is not the main objective of this report, further investigation would be interesting.

RESULTS

The model I turned in had a 36% error rate. Most people had between 30-40% error rates.