STANFORD UNIVERSITY

DATA MINING: SEARCH ENGINE QUERY RELEVANCY

MANAV AGARWAL

MANAV@ME.COM

SUID: 05252903

FINAL PROJECT

STATS 202

Table of Contents

1 Abstract

The purpose of this project is to analyze and mine a dataset pertaining to search queries. Search engines return relevant documents for search queries and the data provided includes a set of attributes and observations from search query and url data. I first analyzed the data, then selected, transformed, and engineered features, then lastly built a series of classification models finally settling on the one with the best performance.

I tried many classifiers including KNN, decision trees, random forest, various types of boosting, and SVM. I found the most success with boosting and SVM, and thus concentrated by efforts and fine tuning there. I ended up choosing the XGBoost Classifier.

2 Data Exploration & Analysis

Data analysis and processing was conducted in Python (scikit-learn/np) as well as R. I later switched to Python because it was easier to parallelize some models that were taking a long time to run.

We'll start with a histogram of all the variables which we will use later in the analysis to follow.

Figure 1: Histogram of Original Training Data

We are given a training set and a test set. The training set has 80046 observations with a total of 13 attributes, of which 10 are described as features. Each has a **query_id, url_id**, 8 numerical signals (**sig1-**

8), **is_homepage** (binary-categorical), **query_length** (numerical), and **relevance** (binary- class target). Training labels are provided for the relevance category pertaining to whether a result was relevant to the search. The 8 signals are unknown factors or “search signals”.

There is also a class imbalance with a ratio of 0.43. This will be taken into account later when splitting the data for training/validation. There are no missing, null, or negative values, however there are zeros and **sig6** has a significant amount with ~ 52k zero values. This might be useful later when transforming.

The **query_id** and **url_id** are nominal attributes, with **url_id** having 94% uniqueness, **query_id** is less so since there are multiple results for a given search, as a result it might be possible to use it later if a feature is engineered.

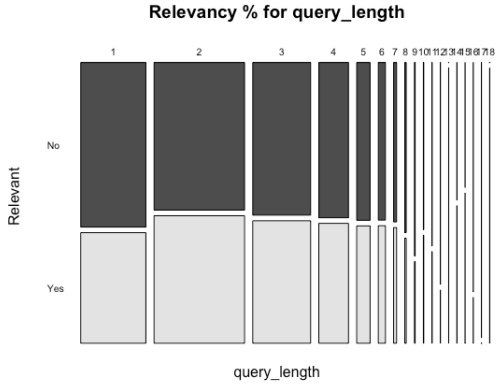


Figure 3

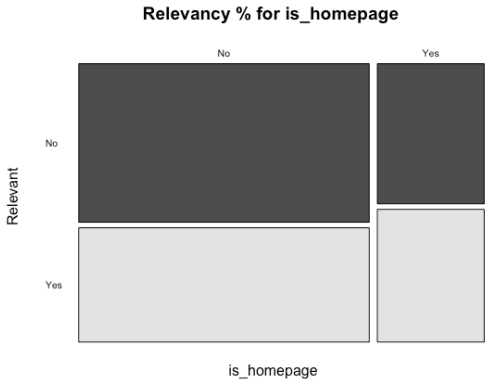


Figure 4

We can assume that **query_length** is the number of words in the query. Looking at the above figure we can see that it does impact relevancy. The factor **is_homepage** is categorical for the source of the query and we can see this has an effect on the relevancy (~7% more) as well in Figure 4.

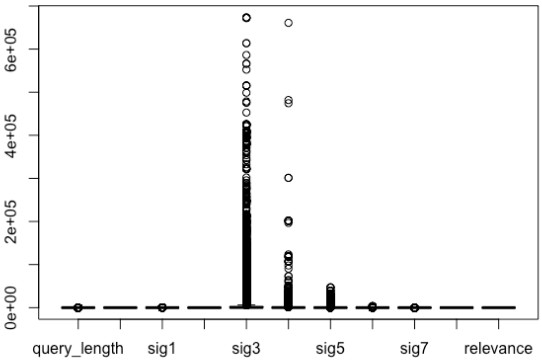


Figure 5: Boxplot of Training Data

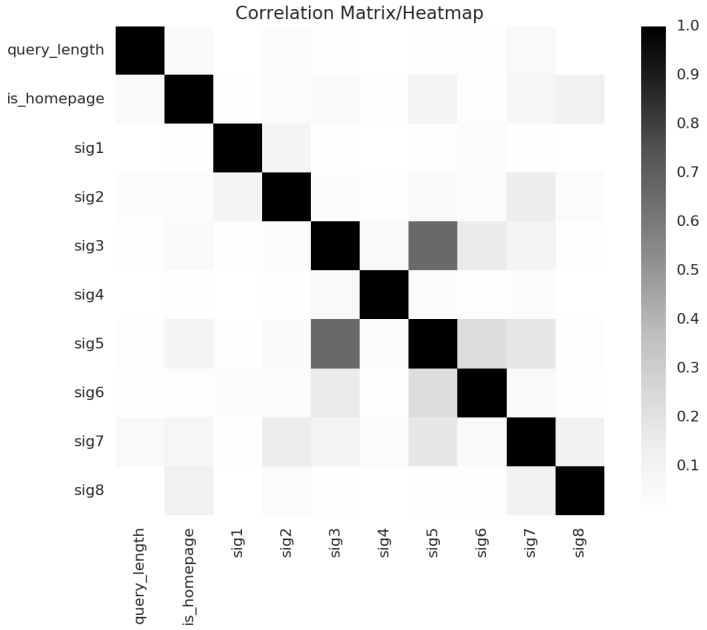


Figure 6: Correlation of Numeric Data

Looking at Figure 1 and Figure 5 we can see that **sig3**-**sig5** have many outliers and **sig3**-**sig6** are heavily skewed to the right, and **sig1 also slightly**. I will go over transformations in the following section. Looking at Figure 6 we can see that there is some correlation between sig3 and sig5 with covariance about 0.815. Since n is quite large and there is a limited amount of predictors, it probably wont help by excluding these variables. In my tests later with candidate models, I did not see any benefit and often got increased error with removing these attributes.

3 Data Transformation & Processing

3.1 Transformations

Looking back at the histogram in Figure 1, there is apparent right skew in **sig3-6** and some in **sig1**. While a normal distribution nor transformations are required of all classifiers, it may help with some of them that are based on regression. After trying a few methods, I settled on *log(x+1)* for **sig3-6** and *sqrt(x)* for **sig1**. Rather than replace the variables entirely, I created new signals so I can later test to see if the transformations had any effect.

With **sig6** it was improved with the log transform and still looks off. It may have to do with all of the zeros mentioned earlier, I will potentially look into this later if time permits.

3.2 Feature Engineering

Since the number of attributes is only 10 and **n** is relatively high the classifier could likely benefit from additional features. First as we explored in the data the **query_id** and **url_id** are not really unique, with **query_id** less so. This means that there could be more or less **url_id**'s for a given query. While I am not sure exactly why this is the case, it could just have something to do with the search and type of metric applied based on how many actual results there were, perhaps similar to *"Pagerank"*.

I first create a variable that assigns a number to each **url_id** sequentially, restarting when it reaches a new **query_id** called **url_rank**. I also then sum the total number of urls returned for a given **query_id** in a variable called **query_num**. In my testing I found that these features improved some models and sometimes had no effect or minimal negative effect.

We can see the new features and transformations in the histogram below.

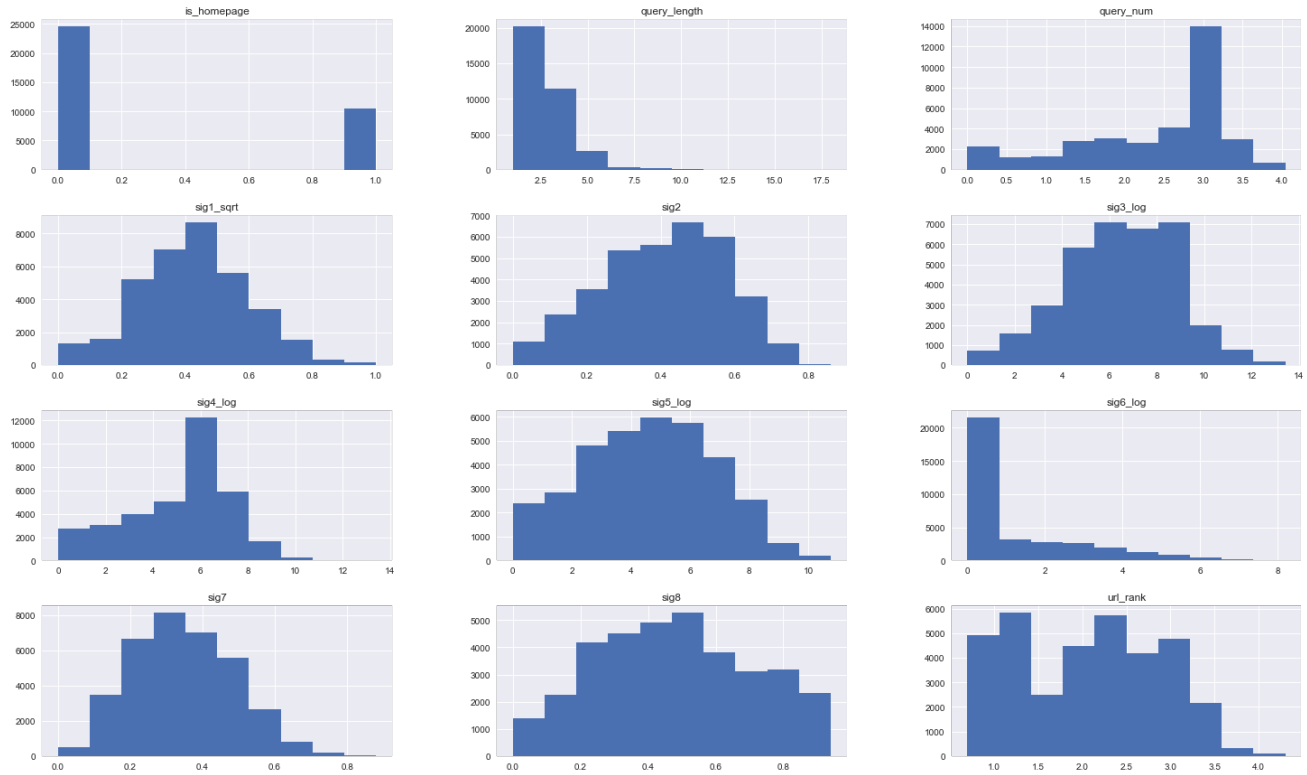


Figure 7: Transformed Predictors

3.3 Scaling and Cross Validation

While some models have scaling built in, I manually scaled the data by removing the mean and scaling to unit variance (standardization). I tested the models with scaling with and without my scaling and did not see a difference.

For the creation of training and validation sets I used an 80/20 ratio of train to test with 5 different datasets for 5-fold cross validation. As a result, the training error metric used in all of my examples will be the average of the error over those folds.

4 Data Mining Experiments

4.1 Naïve Bayes

Naïve Bayes classifier assumes conditional independence and since we have some correlation in our data I think it won't be a good classifier. I will use BernoulliNB and GaussianNB.

Average Training Error:

	ORIGINAL DATA	TRANSFORMED DATA
BERNOULLI	37.95	39.86
GAUSSIAN	39.94	38.86

Although the results aren't good for transformed data yet, its not necessarily indicative that it doesn't provide any benefit. I will move on to other methods

4.2 K-Nearest Neighbor

Our data is already scaled for KNN and I choose odd numbers of k because I won't have to have the issue of tie-breaking. I optimized for the k-value by searching for odd k from 1-201.

First attempt: k=100

Original data: 34.42

Transformed data: 33.59

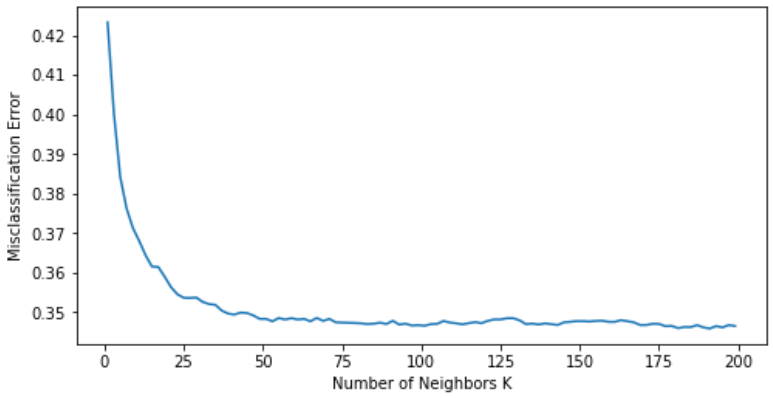


Figure 8: *k* vs Classification Error

The optimal k was 191, however, it seems to really level out after 75.

Optimal *k* = 191.

Transformed Data: 33.45

4.3 Random Forest

Randomforest is another ensemble classifier. It uses fully grown decision trees and thus has high variance and low bias. Each iteration is trained independently. To get even close to what I got with boosting methods I had to increase the number of estimators to 500. I think that with additional tuning I could probably get a better random forest model.

Average Error:

Original Data: 34.75

Transformed Data: 33.37

4.4 SVM

I expected Support Vector Machines to have pretty good performance as they work will even in relatively high dimensions. SVM since it is a regression model is sensitive to scaling and thus it is good that the data is already scaled. I tried both radial and linear kernels using python's sci-kit learn. I found tuning extremely computationally expensive and I had trouble doing much further parameter adjustment because of the time it took. I did an initial experiment with the linear kernel which gave me

over 35% and it seems like it does not change much based on parameters, so I did not explore it further.

To do some basic tuning, I adjusted the "C" value for both the radial and linear kernels. The C parameter tells the SVM optimization how much one wants to avoid misclassifying each training example. Gamma is the other parameter that can be optimized, but so far the model takes too long to run for tuning to be useful.

Linear Kernel: Best: -0.616784 using {'C': 8}

Radial Kernel: Best: -0.617891 using {'C': 3}

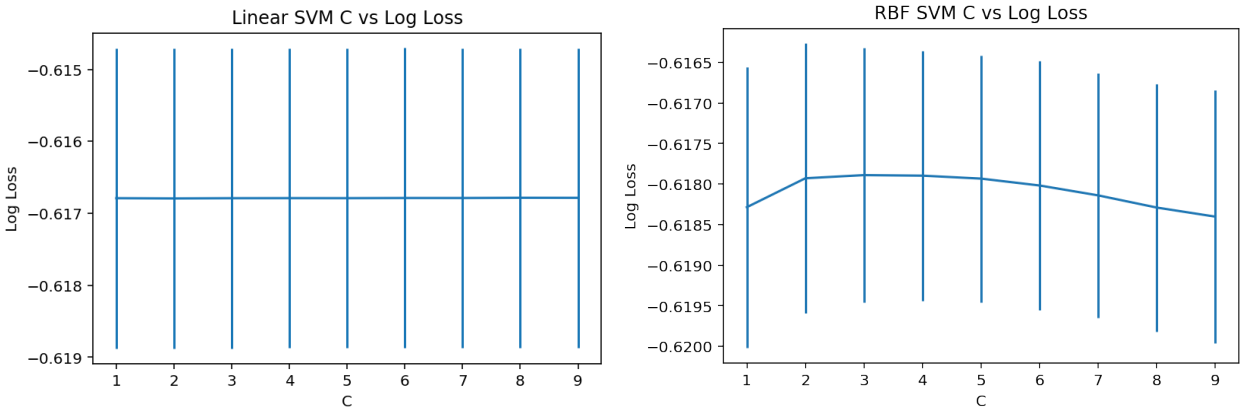


Figure 9,10: SVM Tuning

SVM Radial Kernel, C=3

Original Data: 33.87

Transformed Data: 33.87

5 Boosting Experiments

I decided to do the boosting experiments by trying an experiment on each of the three classifiers I was going to try then choosing one that I would be able to tune. The experiments are structured the same as above in terms of validation and setup. I ran all of the boosting experiments with n=100 boosted trees initially.

5.1 Adaboost

Adaboost is an ensemble classifier that uses weak learners to build a classifier. It is often used to boost decision trees and it's often considered the first option for many because of its performance and accuracy. Adaboost gives more weight to samples that are weighted poorly, it tries to minimize exponential loss. Adaboost is generally also supposed to be good for classification problems. I used the AdaBoost-SAMME implementation in sci-kit (discrete). I also stopped testing the original set as it stopped giving me gains with any of the tree based methods.

Average Error: 33.73

5.2 GBM

GBM is actually similar to Adaboost in many ways except for how it weights weak classifiers based on gradients instead of higher weighting. GBM could be potentially prone to overfitting compared to random forests, but it appears to give better performance.

Average Error: 33.30

5.3 Xgboost

XGBoost is practically GBM but it has several enhancements that make it much faster and parallelizable. As a result it's a lot easier to tune models and parameters. It's very popular with competitive data scientists. XGBoost was implemented with the xgboost package from python and in sci-kit. It is also available in R. XGBoost uses gradient boosting like GBM however, the model formalization is more regularized so it might be better with overfitting.

Average Error: 33.26

So far this is the best result that I have gotten for validation error.

5.3.1 Xgboost Tuning

First I looked at all the features and charted their relative importance.

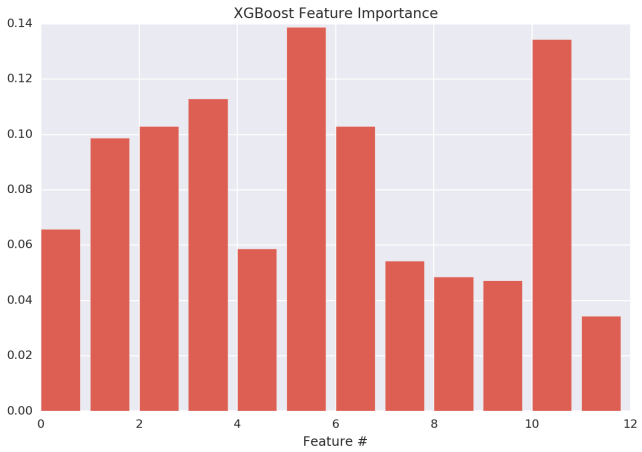
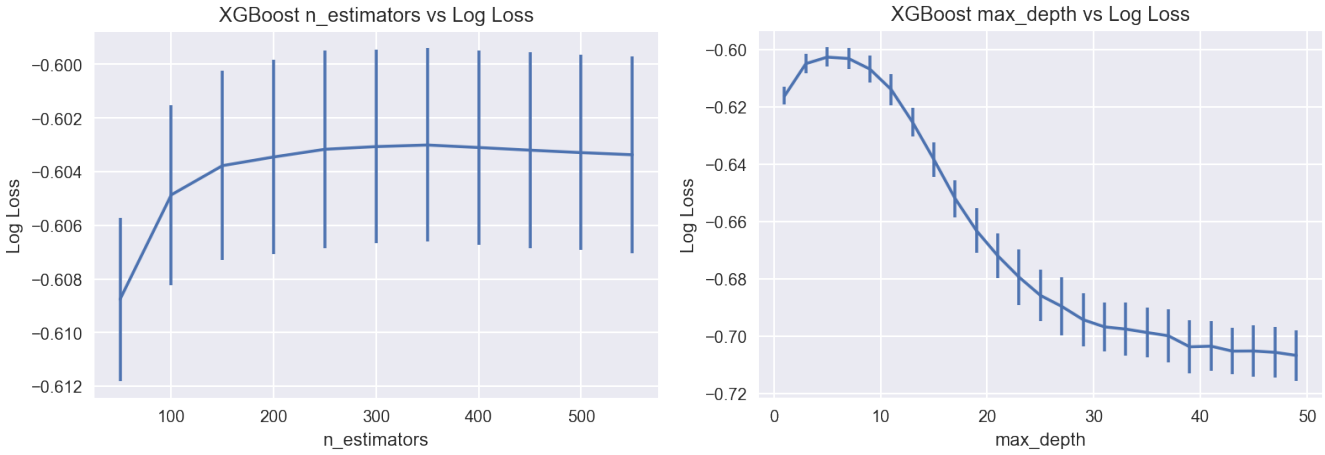


Figure 11: XGBoost Feature Importance

I tried various subsets but my error always increased. So I decided to move on to tune the parameters.

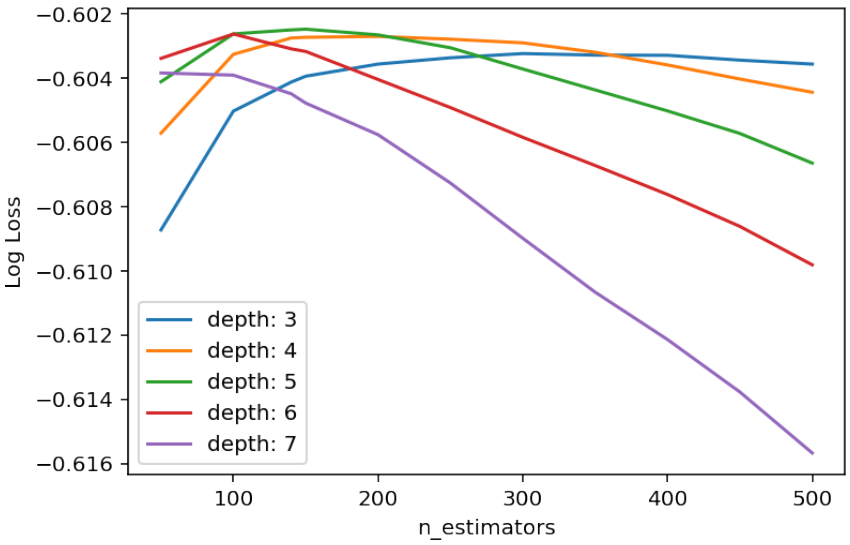
To tune Xgboost I decided to find the optimal number of trees (estimators) and depth. I used Grid-Search cross validation (GridSearchCV) from the sci-kit library which implements a function that maximizes the score of a function, so I had to use the negative log loss for the XGBoost Classifier.



Figures 12, 13: XGBoost Tuning

Best: -0.603 using {'n_estimators': 350}, Best: -0.6026 using {'max_depth': 5}

Although this did give me better performance, often when the depth increases, fewer trees are required. Therefore I did a search combining both number of estimators and depth and found that there is a better solution with fewer estimators.



Best: -0.602486 using {'n_estimators': 150, 'max_depth': 5}

Figure 14: XGBoost Depth&Estimators vs. –Log Loss

XGBoost Summary

n_estimators	*Depth=3*	*Depth=5*
100	33.26	
150		**33.01**
350		33.24

At n=150 estimators and depth=5 our error is 33.01. Hopefully the fewer estimators will be better with overfitting.

Next I will try to test this model against random forests by adjusting the subsample and colsample_bytree parameters. These essentially add a percent of random trees which could also help with potential overfitting.

Since there is much more fine tuning that can be done with things with this model I would like to use the SVM, random forest, and XGB models for my final submissions.

6 Conclusion

For the amount of parameter tuning required for boosting I think that random forests provide a great amount of performance while still giving good accuracy and somewhat preventing from overfitting, but it obviously depends on the matter at hand. If I had more time, I would do more feature engineering, explore subsampling and probably tune some of the other classifiers more.