Search Query Data Analysis and Relevance Prediction

Robert Yates

Abstract

In this class project, there were 80046 training records of search queries per specific urls along with the relevance of the url for the given query. The 12 attributes of the data were analyzed and the relationships between them. The data was normalized to make relevance prediction easier. Relevance prediction was based on four binary classifiers: decision tree, k nearest neighbors, SVM, and naive bayes. Lastly, a voting algorithm was implemented to produce the final relevance prediction on 30001 records of testing data.

1 Introduction

A search engine locates documents relevant to a user's query. Search engines use many attributes to determine how relevant a particular webpage is to a specific query. The URLs of resulting webpages are returned to the user in order of webpage relevance. In this class project, we make a prediction for the relevance of URLs given query and URL data.

2 Data Description

We are given 80046 records including the binary relevance attribute for training data and 30001 records without the binary relevance attribute for testing data. Our task is to predict the binary relevance attribute on the 30001 records of testing data. The attributes are as follows: query_id, url_id, query_length, is_homepage, sig1, sig2, sig3, sig4, sig5, sig6, sig7, sig8, (training only) relevance. Their properties are shown in table 1.

Analyzing table 1, we observe training mean and maximum values are often slightly larger than testing mean and maximum values. This is likely due to the training dataset being larger and having more large value outliers than the test dataset. In addition, no values are smaller than 0. Also note test query_ids are from 0 to 4630 and training query_ids are from 4631 to 17038. The differences between training and test data could affect our model when predicting relevance. Currently we do not alter our training or test data set to account for this, but it is something to keep in mind.

Note for the attribute sig6, the 3rd quartile is quite small (only 2) yet the maximum values are quite large (3645 and 2667 for training and test data). Looking at the data we see the majority of the values in the sig6 column are actually 0. This can affect model because the data is too right-skewed. Therefore, we will have to transform sig6 into another scale as described below.

3 Relationship Between Data

The relationships between the data values are shown in figures 1. It is obvious attributes are correlated with themselves, so we ignore this cases (in the diagonal). Query_id and url_id are correlated with each other which makes sense since a single query is often used on 5-10 urls. Query_id has no correlation with anything else while the url_id has barely any correlation with

Attribute	Type	Dataset	Mean	Variance	Quartiles (1st,2nd,3rd)	Range
relevance	boolean	training	0.4370862	0.2460449	0, 0, 1	0 or 1
query_id	integer	training test	10862.07 2327.048	12864903 1794313	7756, 10876, 13950 1162, 2340, 3491	[4631,17038] [0,4630]
url_id	integer	training test	64219.33 14350.04	507282379 69187749	45386.25, 64325.5, 83446 7121, 14334, 21512	[30,102816] (most values are roughly [28624,102816]) [0,86140] (most values are roughly [0,28623])
query_length	integer	training test	2.585826 2.475651	2.31677 1.764082	2, 2, 3	[1,18] [1,16]
is_homepage	boolean	training test	0.2689454 0.270191	0.1966162 0.1971944	0, 0, 1	0 or 1
sig1	real	training test	0.1832401 0.1826976	0.02171307 0.0219385	0.08, 0.15, 0.24	[0.0, 1.0]
sig2	real	training test	0.3469466 0.3466594	0.02977173 0.02957468	0.21, 0.34, 0.48	[0.0, 0.86] [0.0, 0.81]
sig3	integer	training test	4857.079 4247.314	553753763 351338344	78, 417, 2537.75 84, 418, 2568	[0, 673637] [0, 672625]
sig4	integer	training test	742.3163 679.3362	23216585 6626420	24, 220, 591 24, 234, 591	[0, 660939] [0, 237705]
sig5	integer	training test	550.5276 508.36	3564295 259748	10, 64, 336 10, 64, 334	[0, 46994] [0, 39248]
sig6	integer	training test	14.09916 12.68661	8112.321 5334.44	0, 0, 2	[0, 3645] [0, 2667]
sig7	real	training test	0.3194636 0.3207436	0.0192241 0.01879649	0.22, 0.31, 0.42	[0.0, 0.88] [0.0, 0.87]
sig8	real	training test	0.4718456 0.4695493	0.05350247 0.05280345	0.29, 0.46, 0.64	[0.0, 0.94]

Table 1: Table of Data Properties

other values. The signals in general seem to be somewhat, but not significantly correlated with each other. sig_3 and sig_5 seem more correlated with each other than the rest of the signals. is_homepage seems to have a positive correlation with sig5, a slight positive correlation with sig3, and a slight negative correlation with sig8. query_length seems to have a slight negative correlation with many of the signals, especially sig6. is_homepage and sig8 have a slight negative correlation with each other. sig1 has the least correlation to any of the other signals. Note is_homepage is either 0 or 1 only and sig1, sig2, sig7, and sig8 are real numbers from 0.0 to maximum values of at most 1.0. Relevance (the value we are trying to predict) has a slight correlation with all the signals except for sig8. Relevance has no correlation with the remaining attributes (query_id, url_id, query_length, is_homepage). However, there appears to be a somewhat stronger correlation between sig2 and relevance compared with the other signals. So this indicates it might be possible to predict relevance with only sig2.

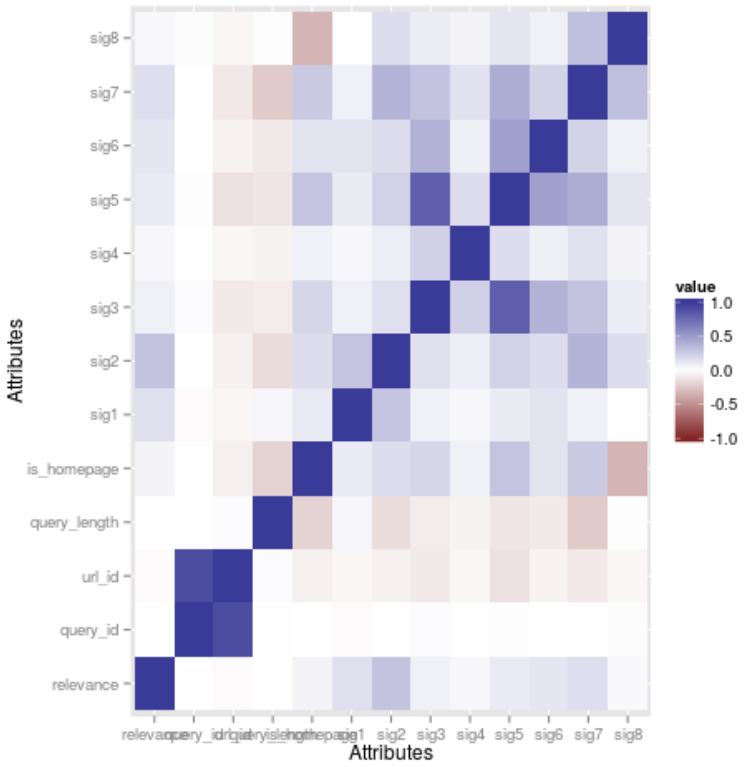

Figure 1: Training data relationships. (Testing data shows a similar correlation.)

Naive Bayes assumes all attributes are conditionally independent given data. But since there is correlation (both positive and negative) between the attributes, this breaks that assumption. However, in practice in still can be useful to run Naive Bayes since the algorithm still runs well. Note that when we added the is_sig6_on column naive bayes performed slightly worse and the reason behind it may be due to the correlation between is_sig6_on and sig6 and lack of new information conveyed in the extra attribute.

4 Normalization Procedure

Figure 2 is a box plot of the original training data. Notice how sig3, sig4, sig5, sig6 have very large ranges. These attributes were pre-processed so they have values closer to a normal distribution. (The box plots showing 1st, 2nd, and 3rd quartile would be clearly visible.) When plotting any changes to the normalized data, I created a version of the box plots that excludes the query_id, url_id and query_length in order to see the signal values more clearly. I tried a few approaches of normalizing the data for sig3, sig4, sig5, sig6 attributes. The remaining attributes I left unchanged.

The normalization procedure was performed on sig3, sig 4, sig 5 and sig6 since as shown in table 1, these are the integer values with large ranges. For these four attributes, the mean is much much greater than the median so the data is heavily right-skewed. (In a left-to-right graph, the right tail is longer and the mass of the distribution is concentrated on the left of the figure.)

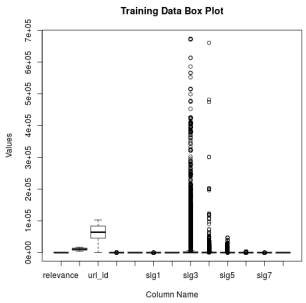

Figure 2: Training data before normalization. (Testing data shows a similar plot.)

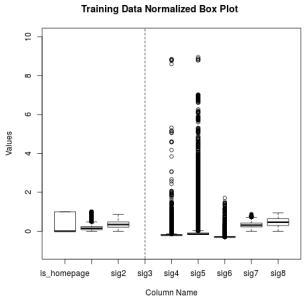

Figure 3: Training data after transforming data to z scores: $z = \frac{x-\mu}{\sigma}$.

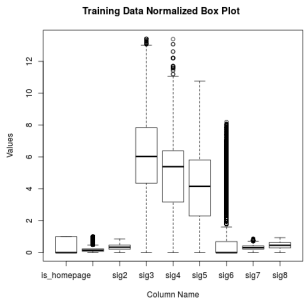

Figure 4: Training data after taking the log of values where log(0)=0

Figure 3 shows the first approach to normalizing the four attributes (sig3-6). Here, the attribute mean is subtracted from the given value, then divided by the standard deviation. This effectively computes the standard score or z-score of the values on a unit normal distribution. However, the data was still distributed too heavily to large values and still very right-skewed. The outliers above the 3rd quantile were very large and sig3 had an extremely wide range. Performing additional modifications to the z-scores yielded no better distributions.

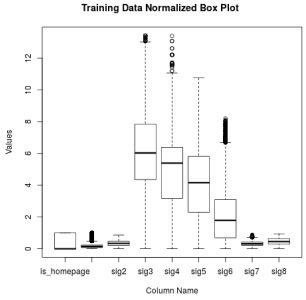

Figure 5: Training data after taking the log of values where log(0)=0, except for sig6 where log(0) is excluded from plot

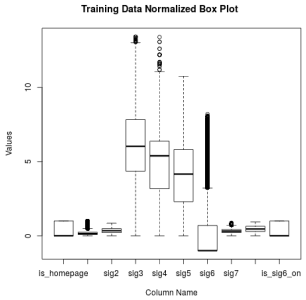

Figure 6: Training data after taking the log of values where log(0)=0, except for sig6 where log(0)=-1

Figure 7: Training data after normalization and divide by max value, where log(0) in sig6 is excluded from plot

In Figure 4, I took a log of the four attributes and set log(0)=0 when a zero was encountered and the distributions were much improved. However, sig6 still looked right-skewed. If we exclude the case where a value is log(0) from sig6 in the plot, sig6 now appears as a normal distribution as shown in figure 5. This is likely because the at least half of the sig6 values are zeros. Therefore, we will set log(0)=-1 in sig6 and create a new field called is_sig6_on which is 0 when sig6 was originally 0 and 1 when sig6 was originally a positive integer. A plot of these values is shown in figure 6.

Finally, for query_length, sig3, sig4, sig5 and sig6 attributes, we divide each value by the maximum value of that attribute across both training and test data. Again, ignoring the case where log(0) occurs in sig6, we plot the final result of normalization in figure 7.

5 Dataset Error Rates

Classifier	Slow1 (Slow K-Fold Original)	Slow2 (Slow K-Fold Ignoring Query_Id, Url_Id)	Slow3 (Slow K-Fold Normalized With Is_Sig6_On)	Slow4 (Slow K-Fold Normalized Ignoring Query_Id, Url_Id)	Fast (Fast K-Fold Normalized Ignoring Query_Id, Url_Id)
DT	0.3608122	0.3590793	0.3608119	0.3590789	0.35655
KNN	0.4902074	0.4701945	0.504735	0.4261577	0.4262
SVM	0.4510945	0.3537199	0.4420749	0.3501688	0.342825
NB	0.4641125	0.3972801	0.47615	0.3819898	0.3851

Table 2: Table of Error Rates

Figure 8: K-fold test error rates of classifiers running on different datasets.

Four binary classifiers were used to predict relevance: decision tree (DT), k nearest neighbor (KNN), support vector machines (SVM), and naive bayes (NB). I used two versions of k-fold cross validation (slow and fast) on the training data.

The "slow" version of k-fold cross validation is the initial version. Because training on a large number of instances would take a long amount of time, I decided to train on 2000 instances and test on the remaining 78046 in the training data. I repeated this process 40 times (40 folds were used). For example, when k=1 the classier trained from 1 to 2000, and tested from 2001 to 80046. When k=2 the classier trained from 2001 to 4000, and tested from 0 to 2000 and 4001 to 80046. When k=3 the classier trained from 4001 to 6000, and tested from 0 to 4000 and 6001 to 80046. And so on. When k=40 the classier trained from 78001 to 80000, and tested from 0 to 78000 and

80001 to 80046. For this "slow" version of k-fold, the misclassification error rate is an average across 40 folds.
In the "fast" version of k-fold, initially all training data is randomly shuffled by rows. A classifier is trained on the 5000 instances at the top on the shuffled ordering without replacement. The classifier is then tested on the next 5000 instances without replacement. The classifier is then trained on next the 5000 instances, and tested on the 5000 instances both without replacement. The process is then repeated 8 times over 8 training/testing pairs and we take an average of the test misclassification error for that classifier.

Table 2 and figure 8 display the test misclassification error rates on various classifiers with different datasets. (Note default parameters are used on all classifiers.)
Slow1 (Slow K-Fold Original) in the table and plot respectively, represents running the "slow" k-fold cross validation on the original training data.
Slow2 (Slow K-Fold Ignore Query_Id, Url_Id) is running slow k-fold cross validation on the original training data but ignoring the query_id and url_id column. Ignoring query_id and url_id allows us to get much better test misclassification error rates for decision tree, SVM, and naive bayes classifiers. However, k nearest neighbor is only slightly better because the data isn't normalized.
Slow3 (Slow K-Fold Normalized With Is_Sig6_On) is running slow k-fold cross validation on the normalized the data. Sig3,4,5,6 are now logs where log(0)=0 except in the case of sig6 where log(0)=-1. Query_length,sig3,4,5,6 are also divided by the max value from training and testing data. Finally, an additional column is_sig6_on attribute is added to indicate whether sig6 was originally 0 or not. However, after normalizing and adding the new field we find the test error rates are much worse.
Slow4 (Slow K-Fold Normalized Ignore Query_Id, Url_Id) is running slow k-fold cross validation on the normalized data but with query_id, url_id, and is_sig6_on columns removed. Fortunately removing query_id, url_id, and is_sig6_on from the classifier training data improves the test misclassification error rate. is_sig6_on most likely made the error rate worse due to it's correlation with sig6 and the fact that no new information is added to the new in that attribute.
Finally, Fast (Fast K-Fold Normalized Ignore Query_Id, Url_Id) in the table represents the same data as Slow4 (normalized data but with query_id, url_id, and is_sig6_on columns removed), but with the "fast" version of k-fold. The error rates are roughly the same as the slow version indicating the faster algorithm can be substituted in place of the slower one.

6 Modifying Parameters

- Decision Tree

Max Depth	(Default)	1	2	3	4
Test Error	0.353175	0.3667	0.3667	0.3585	0.35775

Max Depth	5	6	7	8
Test Error	0.35715	0.3626	0.371425	0.371675

- KNN

k	1	2	3	4	5	6
Test Error	0.422125	0.425525	0.398425	0.399325	0.388775	0.387

k	7	8	9	10	11	12
Test Error	0.380175	0.382425	0.373875	0.377225	0.369675	0.36985

- SVM

Cost	0.0001	0.001	0.01	0.1
Test Error	0.437575	0.352475	0.348025	0.34645

Cost	1	10	100
Test Error	0.3462	0.34625	0.34615

- Naive Bayes

  Test Misclassification Error: 0.3851

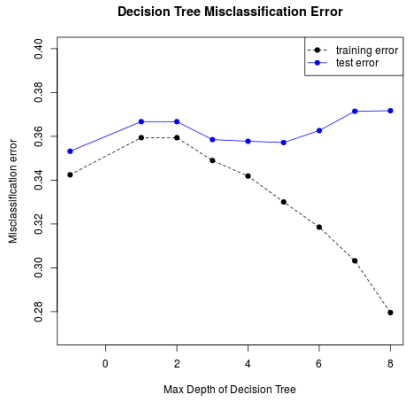

Figure 9: Decision Tree training and test misclassification error with different max depth values. -1 value represents no arguments to decision tree (rpart) function.

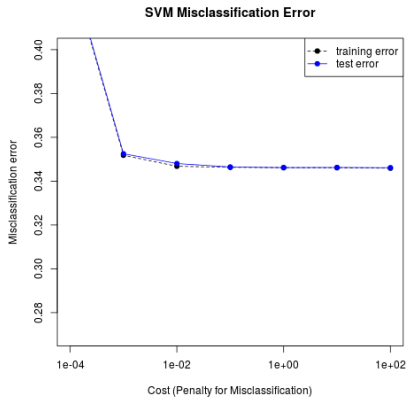

Figure 10: SVM training and test misclassification error using linear kernel with different costs (x-axis is log scale).

The list above and figures 9, 10, 11 and 12 show how varying the parameters for decision tree, SVM and k nearest neighbors affects their classification error scores. The optimal max depth value for decision tree with regards to test error is 5. The optimal k for k nearest neighbor with regards to test error is obtain with k as large as possible; as k gets incrementally larger, KNN

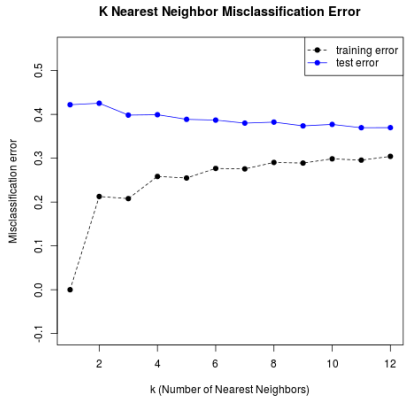

Figure 11: K Nearest Neighbor training and test misclassification error with different number of neighbors.

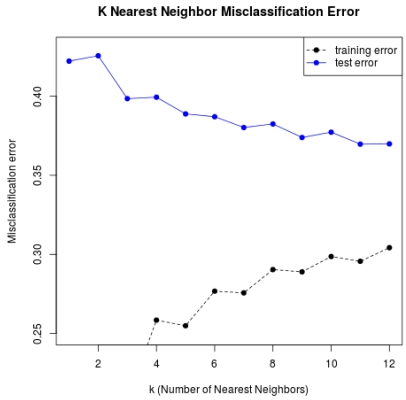

Figure 12: Zoomed in version of the KNN plot in Figure 11.

gets a smaller and smaller improvement in accuracy. In practice, setting k to 12 is makes KNN accurate enough. SVM classification gets worse as cost decreases exponentially and gets better as cost increases exponentially. However, after cost is 1, increasing the cost doesn't really make SVM any more accurate.

7 Conclusion

For our final prediction, a voting scheme was implemented that combined all our best classifiers: decision tree with max depth 5, k nearest neighbor with k as 12, SVM with cost 1, and naive bayes. In one approach ("V-Equal" in figures 13 and 14), each classifier was given an equal vote for 0 or 1 relevance with ties randomly broken. The test misclassification error was 0.347025, which is slightly higher than SVM alone at 0.3462. However, we then implemented another method ("V-Multiply" in figures 13 and 14) where we multiply the classifier accuracy times their vote (-1 for 0 relevance, +1 for 1 relevance), then added all the votes. In this situation, we achieved the best test misclassification error at 0.343625. A comparison of the best rates is found in figures 13 and 14.

Let's estimate how accurate our prediction is. Suppose we have four independent classifiers each with 35% misclassification error. Taking a majority vote, gives us an accuracy as follows: $\binom{4}{3} * .65^3 * .35^1 + \binom{4}{4} * .65^4 * .35^0 = \sum_{i=3}^{4} \text{binom}(i, 4, .65)$ = 1-pbinom(4-3, 4, .65)=0.8735188
Accuracy: 0.8735188, Misclassification Error: 0.1264812
The majority vote is correct 87.4% of the time and assuming the classifiers are completely independent. However, this is an overestimate since the classifiers are not completely independent. In the actual voting, the real voting classifier was only 65.3% accurate.

Possible future work for this project would be to use other classifiers: ensemble methods such as boosting (AdaBoost), random forest, and bagging. AdaBoost with a decision tree as the weak learner (base classifier) and random forest are known to be quite a bit more effective than a single decision tree. Though it wasn't covered in class, logistic regression might be useful as well. A majority vote of six or seven classifiers would likely perform better than four.
Another possible way to improve classifier accuracy would be to make use of what I termed "query

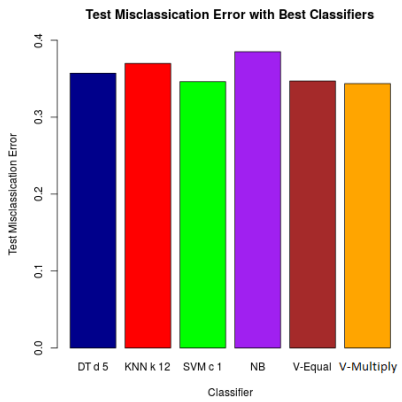

Figure 13: Comparison of test misclassification error from best classifiers.

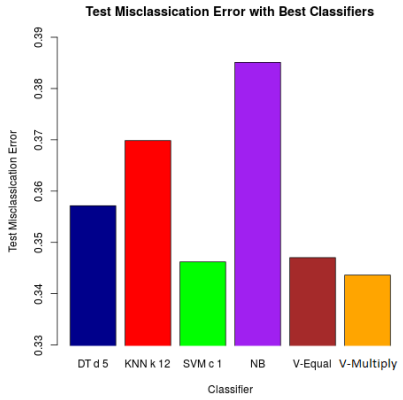

Figure 14: Zoomed in version of the plot in Figure 13 to observe differences.

groups". In one "query group", a single query_id is used multiple times with several different url_ids. For each existing attribute, a new attribute could be added that represents: the average value of each attribute within one's query group or the difference of each attribute's value from its group average. So the number of attributes would increase, but the classifier might be more accurate given the additional data.