University of Illinois, Urbana-Champaign

Stat 542 Final Report: Using Blog Data to Predict Comments

Albert Yu, Steven Hurwitt, Yan Liu

December 15, 2016

1 Introduction

As the traditional modes of media are on the decline, social media has risen to take its place. As the internet is unparalleled in its reach, it has given ordinary people the ability to reach global audiences. One rising vehicle of information on the internet is the weblog, or blog for short. A blog is an online journal or diary which is frequently updated.

Topics for blogs are diverse, ranging from detailing a hobby, providing technical information, or giving personal reflections on the news. Especially in the last few years, blogs are revolutionizing the way information and news is shared. As this form of communication is on the rise, it may be desirable to have a blog that generates traffic, revenue, and popularity. One metric to determine the popularity of a blog could be the number of comments it generates. Therefore, the aim of this project was to predict the number of comments that appear in a blog post - specifically, the number within the upcoming 24 hours.

2 Exploratory Data Analysis

The data used for this task was obtained from the raw HTML documents of blog posts. The original dataset contains 280 attributes. Attributes 1-62 are basic features such as the number of comments and links in certain time periods prior to the basetime.

Attributes 63-262 are 0-1 features indicating whether 200 most discriminative words appear in the blog posts. Attributes 263-276 are 0-1 features indicating the weekday. Attributes 277-280 are features about the parent pages.

We begin by focusing on features in the 1-62 range, as these are generally summary statistics of things related to comments, such as the total comments before the baseline, number of comments one day before the baseline, as well as the number of comments. One thing to note is that a lot of these variables are very "zero-heavy" meaning most blogs don't seem to get comments, which could be an issue in making predictions.

The first thing we will look at is the averages for the number of total comments before baseline, comments in day one before baseline and comments in day two before baseline.

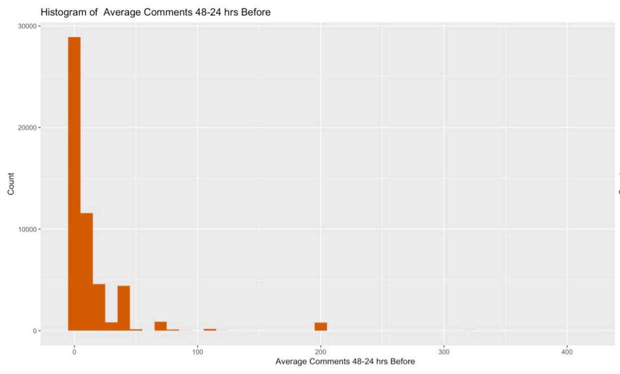

Figure 2.1: Average Comments on Day 2 Before Baseline

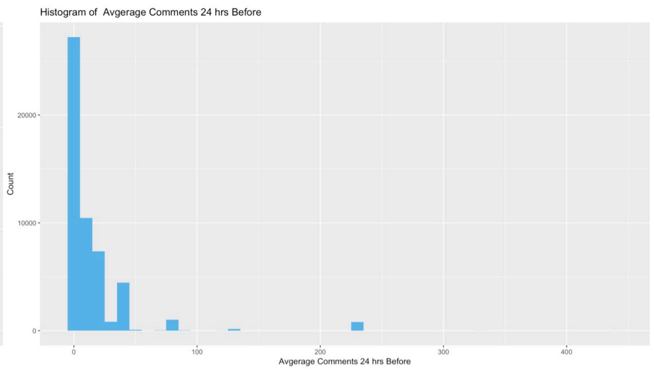

Figure 2.2: Average Comments on Day 1 Before Baseline

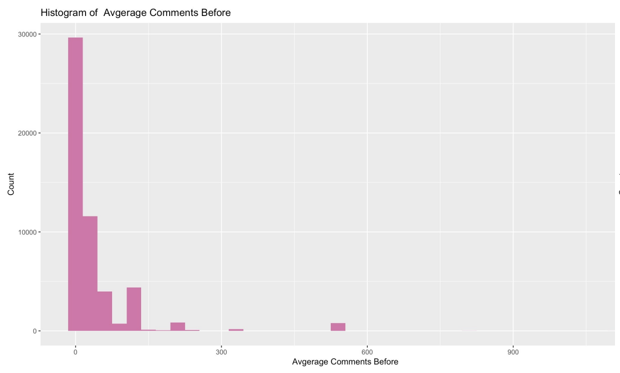

Figure 2.3: Average Comments Before Publication

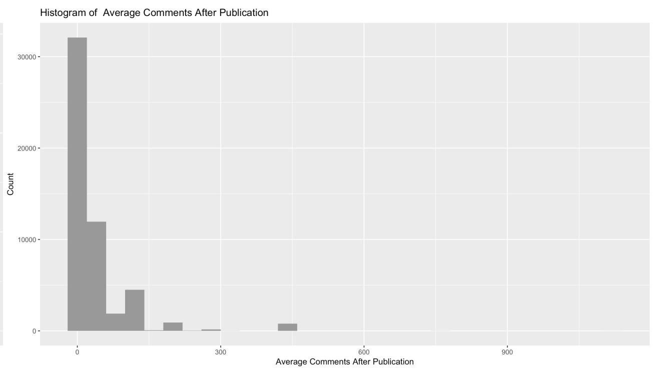

Figure 2.4: Average Comments After Publication

From these plots, we see that the number of comments beforehand all seem to follow a similar pattern in that most blogs have relatively few comments, while some blogs (maybe they're controversial or simply more popular) generate hundreds of comments. The averages are useful to look at, but looking at the medians could also help give us a better idea about how the comments are distributed before the baseline.

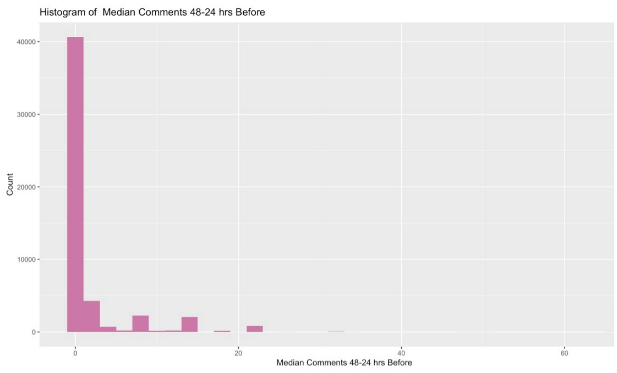

Figure 2.5: Median Comments on Day 2 Before Baseline

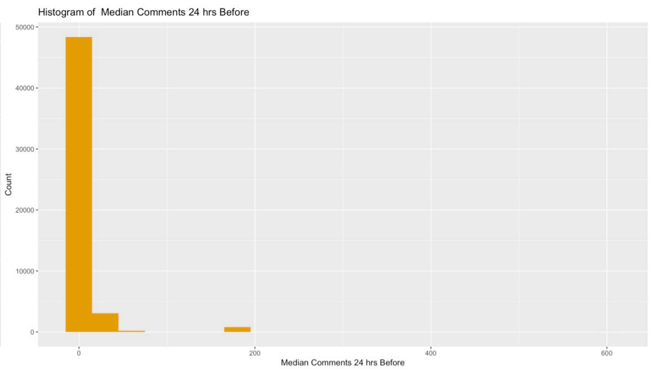

Figure 2.6: Median Comments on Day 1 Before Baseline

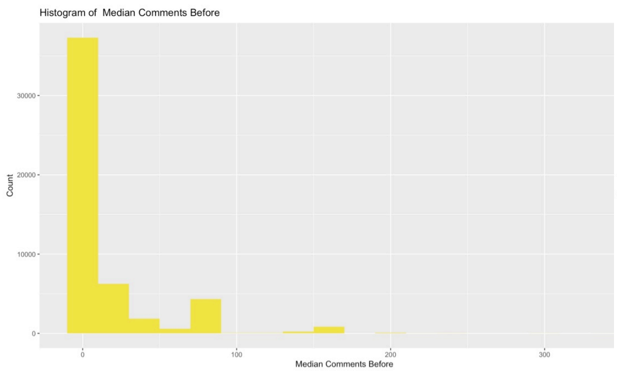

Figure 2.7: Median Comments Before Publication

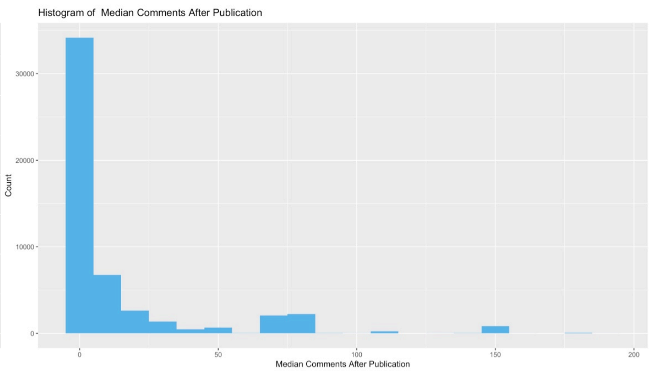

Figure 2.8: Median Comments After Publication

The medians conform to our expectations with this data, and seem to be more centered around zero. Our thinking is that because many blogs may have no comments (and a blog certainly can't have negative comments) that the distribution of comments for all blogs would be skewed to the right, in which case the mean will be greater than the median. However this means that the median will be more robust towards those blogs that have a large number of comments.

Finally, the standard deviations are another useful statistic to look at in regards to the number of comments on each blog. The graphs are shown below.

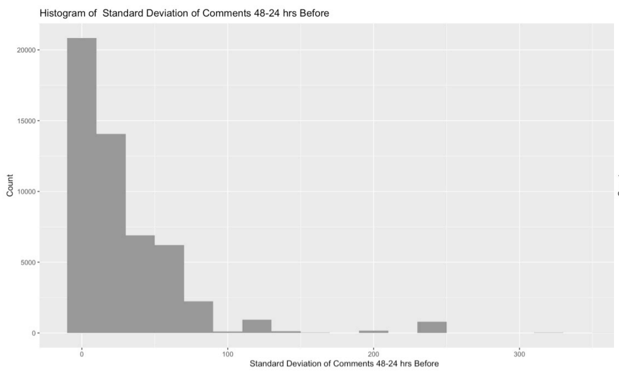

Figure 2.9: St. Dev. of Comments on Day 2 Before Baseline

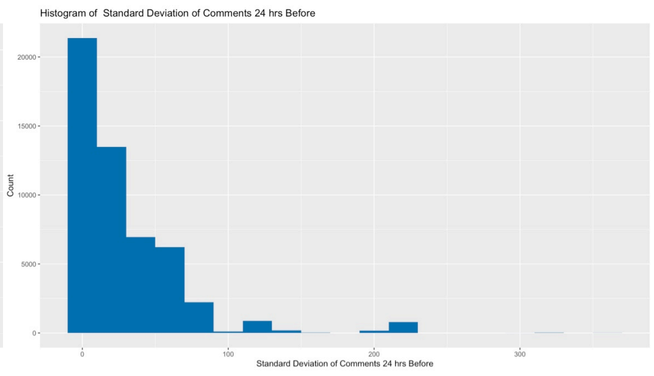

Figure 2.10: Std. Dev. of Comments on Day 1 Before Baseline

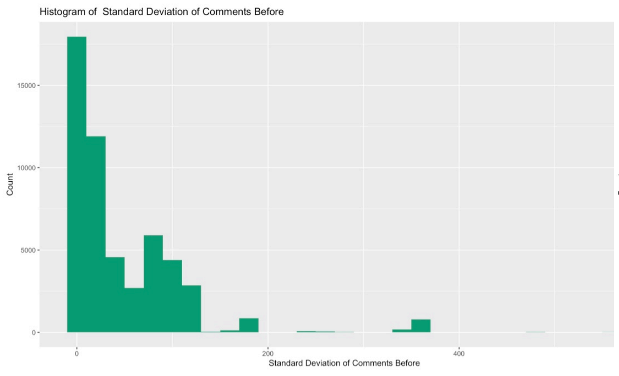

Figure 2.11: St. Dev. of Comments Before Publication

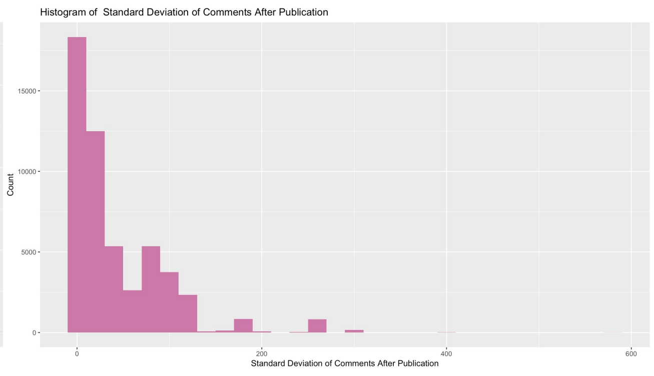

Figure 2.12: St. Dev. of Comments After Publication

The standard deviations also seem to be clumped around zero, with more of a spread than the averages. This makes sense since they describe the spread of the averages, and should only be zero when the average is constant across all blogs. This means that when they are low the averages don't fluctuate that much, which intuitively seems like it would make the number comments easier to predict if they have less variation.

It will also be useful to look at the correlation between all of the variables. As such, a correlation matrix is shown below.

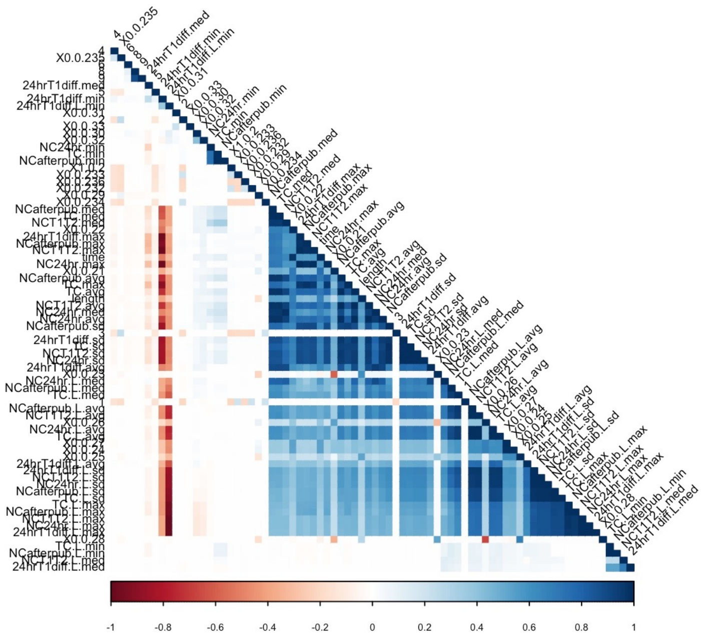

Figure 2.13: Correlation Between Explanatory Variables

This helps us easily identify which the few variables that are pretty strongly negatively correlated with one another, as well as the variables that are positively correlated.

3 Data Cleaning and Transformation

By using a for loop to go through the data, we found that there were 4 columns whose values were all 0 (including the minimum number of comments between T_1 and T_2, the minimum number of links in the last 24 hours before basetime, and the minimum number of comments that the parents received). Since these columns would not be helpful in prediction, they were removed from the dataset.

Before training the model, we do the following transformation to the response variable in the raw data.

$$\tilde{y} = \log(y + 1)$$

For all the methods considered, the prediction accuracy will be evaluated by mean squared error of the log-transformed response.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}[\log(1+y_i) - \log(\hat{f}(x_i)+1)]^2$$

4 Data Analysis

The following techniques were used for prediction: Multivariate regression, K-nearest neighbor, Ridge regression, Lasso, MCP, SCAD, Random forests, and XGBoost.

4.1 Multivariate Linear Regression

For multivariate regression, first, we used the lm function to fit a model with all of the remaining covariates. One thing to consider is that the residuals with this model are assumed to be multivariate normal. Since this is a strong assumption and the nature of the data does not conform to this assumption, it is not surprising that this model does not perform the best. Still, the MSE was calculated according to the formula provided, and we obtained a value of 0.668, which is a baseline to compare our future results with.

Since there were so many variables, we attempted to come up with a way to deal with this issue. Stepwise methods such as forwards and backwards AIC could potentially be used to eliminate non-significant variables. However, due to the large number of variables, this was not feasible so different approaches were tried.

4.2 Penalized Linear Models

As the number of predictor variables grows, the traditional stepwise selection methods mentioned above suffer from high variability and low prediction accuracy. Since the aim of the project is to minimize the prediction MSE, we turned next to penalized linear models for higher prediction accuracy as well as computational efficiency.

One concern in modeling was the correlation between the predictor variables, for example, the correlation of variables 1-50 with variables 51-60. To address this problem, instead of removing variables directly, penalized linear models keep all the predictor variables in the model but constrain the regression coefficients. RSS is minimized using an additional penalty term on the coefficients, which shrinks them. When the shrinkage is large enough, the coefficients will go to zero, effectively giving us another means of variable selection.

This shrinking of the regression coefficients will introduce bias in the estimation of regression coefficients. We can see this with the first two penalized models: Lasso and Ridge Regression. On the other hand, a benefit of shrinkage is that it provides a decrease in variance. When the increase in bias is significantly less than the decrease in variance, the MSE of the resulting model will improve compared to the multivariate linear regression model.

4.2.1 Lasso

The lasso imposes an L^1 penalty on the loss function

$$\text{argmin}_\beta ||\mathbf{y} - \mathbf{X}\beta|| + \lambda|\beta|$$

where the shrinkage parameter λ is selected by cross-validation.

Over 200 variables have nonzero coefficients in this model. The MSE calculated from Lasso was slightly better than regular regression.

4.2.2 Ridge Regression

Ridge regression imposes an L^2 penalty on the loss function

$$\text{argmin}_\beta ||\mathbf{y} - \mathbf{X}\beta|| + \lambda||\beta||^2$$

The shrinkage parameter λ is still chosen by cross-validation.
Ridge regression again gave use a slight improvement on the MSE.

4.2.3 Unbiased Penalties

In addition to the biased penalties, we also considered unbiased penalties such as minimax concave penalty (MCP) and SCAD (Smoothly clipped absolute deviation) penalty. MCP selected 56 variables, while SCAD selected 60 variables. With the default parameter settings, the MSEs calculated for MCP and SCAD were 0.656 and 0.634, respectively. Of all 4 of the penalized methods, the unbiased SCAD penalty produced the lowest MSE.

4.3 k-Nearest Neighbor

K-Nearest Neighbors is a relatively simply algorithm that uses all the data to predict a numerical target using a similarity measure, such as a distance function. Using 1 nearest neighbor minimizes the bias but has a large variance. Increasing the number of nearest neighbors, k, introduces more bias, but will reduce the variance. We ran the standard unweighted knn with varying k neighbors considered. The best value we found was $k = 20$, which gave us an MSE of 0.509.

4.4 Ensemble Methods

We also looked at bootstrap aggregation (bagging) and boosting to see if they could lower the MSE further than the other models. These methods use multiple learning algorithms to provide better predictive performance compared to any of their parts alone.

4.4.1 Random Forests

Decision tree learning can use trees for either classification or regression purposes. The trees act as predictive models which map the features of an item to a conclusion about the target value such as our number of blog posts generated within 24 hours. When trees are grown very deep, the models often have low bias but high variance, as they overfit the training set. Random forest applies the technique of bagging to tree learners to correct the overfitting issue.

4.5 Random Forests

For random forests, we set number of trees $ntree = 500$, $nodesize = 25$, and $mtry$ as the default value. MSE = 0.542. When we lower the $nodesize$, we can get better MSE, but it takes longer time for computation.

4.6 XGBoost

Finally we decided to use XGBoost, which is a computationally more efficient version of the gradient boosting techniques covered in class. The basic idea of XGBoost is same as boosting. We train a weak learner (tree-based models) to minimize the loss function at each iteration, and add a fraction (η) of this learner to the model.

Parameter η controls the fraction of the new learner added to the model in each iteration. After parameter tuning, this method resulted in the lowest MSE of 0.390. This is what we chose as the main part of our final model.

4.7 Comparison

Table 4.1: Comparison of Model Performance

Method	MSE	Method	MSE
Linear regression	0.668	MCP	0.656
k-nearest neighbor	0.513	SCAD	0.634
Ridge regression	0.643	Random Forests	0.542
Lasso	0.656	XGBoost	0.390

According to the results above, almost all the linear methods and penalized linear methods gave MSE larger than 0.6, and tuning the parameters cannot improve the results a lot. Also, penalized linear models cannot effectively select variables. Therefore, linear model is not a good choice for this dataset.

K-nearest neighbor performed better than linear models. However, it takes too much computational time to tune the parameter k. Also, a cross-validation on the training set easily leads to overfit. Another disadvantage is that the algorithm itself cannot help us to rule out the irrelevant predictors.

Tree-based models such as random forests and boosting performed better on this dataset. Overfitting is relatively easy to control, and parameter tuning is efficient for XGBoost.

5 Final Model

5.1 Importance of Predictors

Penalized linear methods such as Lasso cannot give a satisfactory dimension reduction. In order to reduce dimension, we checked the importance of each predictor given by random forests result.

Here in order to reduce computational cost, we ramdomly sampled 5000 data points from the original training set and ran a random forests model with $ntree = 2000$ and $mtry = 15$. The importance of each variable is shown in the graph below.

The predictors with the highest importance are attribute 51-62, which are basically information on the number of comments and links received before the basetime. Most of attributes 1-25 also have high importance, which are the average, standard deviation, maximum and median values of attributes 51-55. However, the minimum values of attributes 51-55 only have weak importance. Also the average and standard deviation of the number of links cannot be neglected. However, parent blog information does not seem to contribute a lot to the prediction.

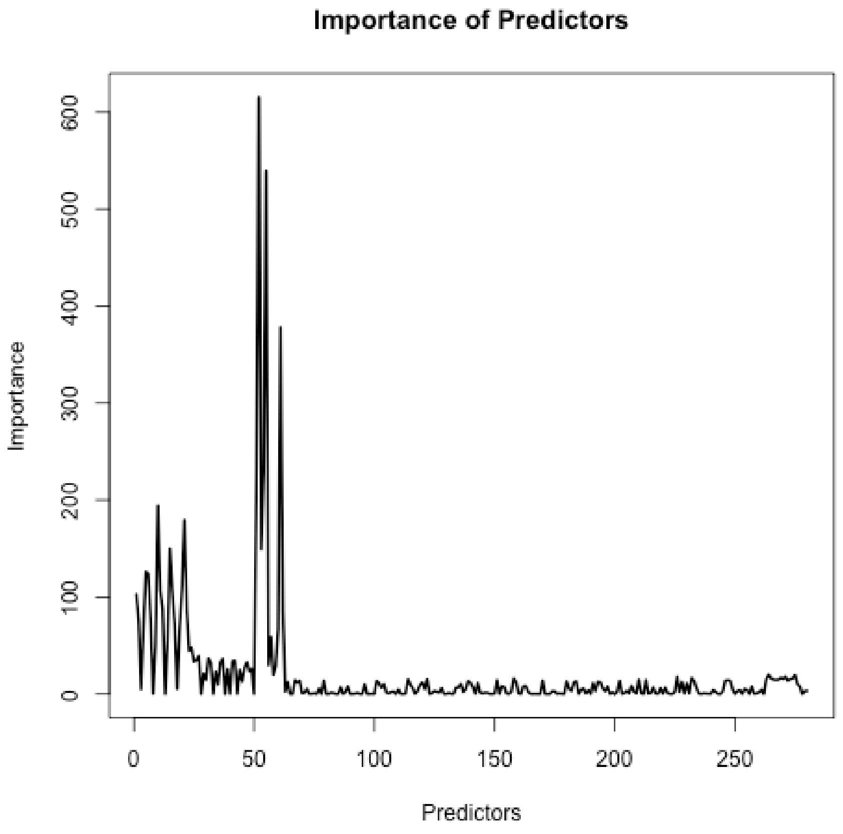

Figure 5.1: Importance of the Predictors

Only a small number of predictors are impactful, while most binary predictors for the bag of 200 words do not have strong importance. Therefore, we selected the variables with importance more than 20 into our final model.

5.2 PCA on the Binary Predictors

The binary predictors for the bag of 200 words should provide crucial information in the text. In order to capture these features, we performed PCA on these 200 predictors. The percentage of total variance explained by each principal component is shown in the following graph.

The first 55 principal components explained more than 90% of the total variance. Thus, we take the first 55 principal components as refined textual features.

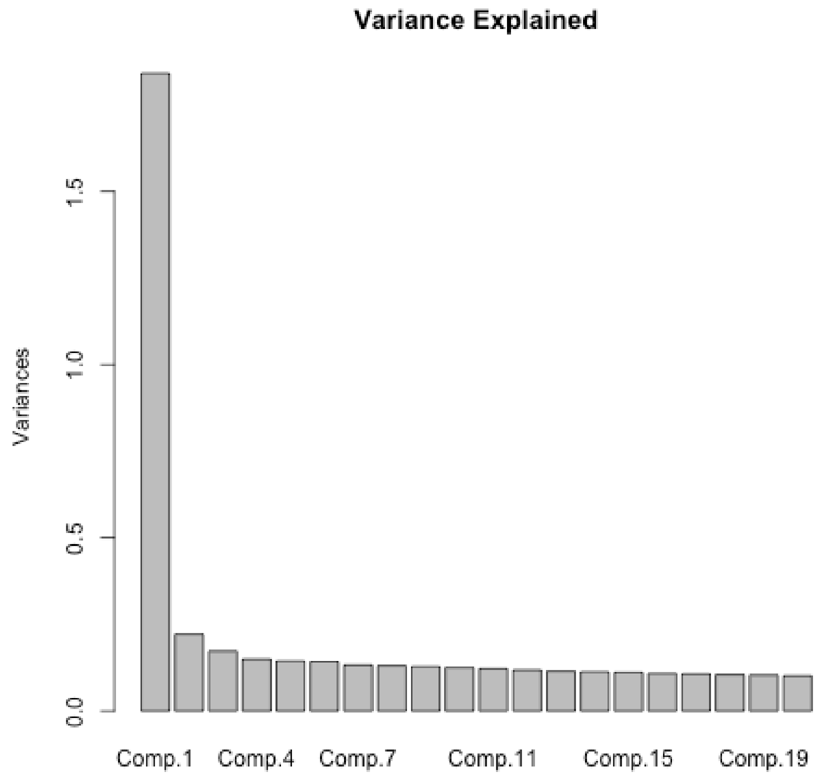

Figure 5.2: Variance Explained

5.3 XGBoost on the Refined Data Matrix

XGBoost is short for "Extreme Gradient Boosting". It is based on a tree ensemble model, which is a set of classification and regression trees (CART). In CART, each of the leaves is associated with a real value. Since a single tree is not strong enough alone, a tree ensemble model sums the prediction of multiple trees together. Given K trees, the tree ensemble uses K additive functions to predict the output as follows:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}, \text{ where } \mathcal{F} \text{ is the set of all possible CARTs.}$$

To learn the trees, we will maximize the objective function

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2.$$

Since this model includes functions as parameters, it cannot be optimized using methods in Euclidean space. Instead, with gradient tree boosting, the model is trained in an additive matter. At the t-th iteration of the i-th instance, this method adds the ft that minimizes the function:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

Since it is not practical to enumerate all the possible trees, only one level of the trees is optimized at once. This is done by looking at the loss reduction after any split of a leaf. XGBoost makes splits up to the maximum depth specified and then prunes the tree backwards to remove splits beyond which there is no positive gain. We implemented XGboost from the package *xgboost*. The final model we settled on used a maximum depth of 4.

We combined the 48 predictors with high importance and the first 55 principal components of the word matrix, and obtained the refined data matrix. We performed XG-Boosting on this refined data matrix and the MSE is 0.40.

6 Discussion and Conclusions

Based on these results, we can begin to rank the algorithms in terms of how well they predicted the number of blog comments for our test data. Linear regression performed the worst, followed by the standard regression penalties, LASSO and ridge regression. The minimax concave penalty (MCP) and the smoothly clipped absolute deviation penalty (SAD) were next and also gave MSE's in above 0.6. k-nearest neighbor and random forest were better, as they broke 0.6 and got the MSE down to the 0.5 range. However XGBoost ended up being the best model we used according to the given metric, and predicted the test data with an MSE in the 0.3 range. This ended up being our final model for the project.

The main sources of error in learning come from noise, bias, and variance. While the training stage for bagging is parallel, the learner is built sequentially in boosting. Since these methods combine several estimates from different models, they both decrease the variance compared to a single estimate. When the single model runs into issues of overfitting, bagging methods will usually be a better solution. However, in our case, it seems that the issue was low performance of our single models. Thus, as seen in the MSEs above, boosting methods are superior here since they also try to reduce the bias.

The MSE of the final model is basically the same as the MSE given by XGBoosting with all the predictors. More parameter tuning cannot improve the MSE very much. Therefore, our variable selection and dimension reduction procedure ruled out the irrelevant predictors and kept most signals in the data.

References

[1] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." arXiv preprint arXiv:1603.02754 (2016).