STAT 542   Final Project Report

Team
Corn Killer Team:
Zhoutao Pei (zpei3@illinois.edu)
Peiran Cao (pcao6@illinois.edu)
Boyang Liu (bliu52@ilinois.edu)

0. Background

The dataset originates from Budapest University of Technology and Economics' research on blog posts. Data were collected by crawling raw HTML-documents. A base time was chosen and blog posts published at most 72 hours before that were selected and parsed. The task is to predict the number of comments for a posted blog after 24 hours of the baseline.
In the train data, the base time were in year 2010 and 2011. In the test data, the base time were in February and March 2012.

1. Data Exploration and Analysis

The first 280 columns in the dataset are independent variables, including number of comments/links in different time periods in the last 72 hours, aggregation of such features by source, words contained in the blog posts, weekday information and parent page information. The last column of training data is the dependent variable – number of comments in the next 24h.
The train data has 52397 observations, and the test data has 7624 observations. The dependent variable is highly skewed. 64.05% of the dependent variables in train data are zero, and among the non-zero ones(Fig 1), about 75% of them are less than 10, however, the dependent variable can also be as high as 1424.

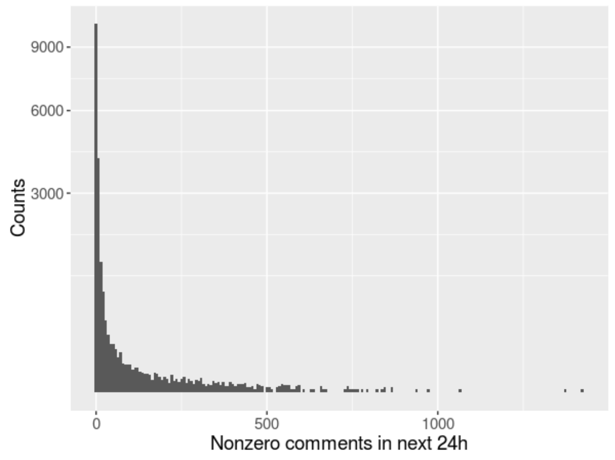

Fig 1. Histogram of Non-zero dependent variable

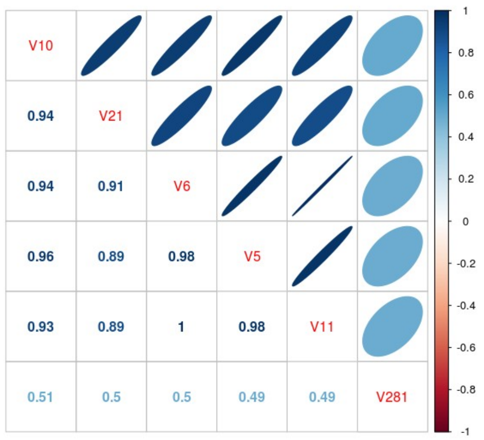

Fig 2. Correlation matrix

To see which independent variable is strongly correlated with y, we calculated the correlation between each pair of independent variable and dependent variable. As shown in Fig 2, the 10th, 21st, 6th, 5th, 11st independent variable have the highest correlation with the dependent variable. Those features are medians or averages of the comments from the source in different time periods.

Fig 2 reveals another feature of this dataset – highly correlated independent variables. Actually, many of the first 60 features are highly correlated with each other. Some of these variables are redundant, and by dropping those variables, the model performance will be improved or at least not impaired.
The dataset provides 200 independent variables for the words contained in blog posts. The left plot of Fig 3 shows the frequency that each word appears in train data. 3 words appear in more than half of the train data, they are very likely to be stop words, which do not have much information for our predictive model. 117 words appear in less than 523 observations, which is only 1% of the train data.

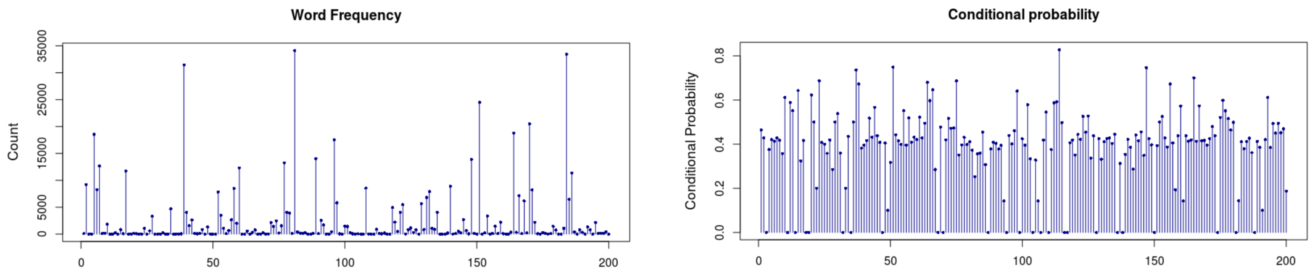

Fig 3. Frequency of words and conditional probability of y>0

In order to see which word is positively or negatively associated with the dependent variable, we calculated conditional probability P(dependent variable is greater than zero | word appears in the post) for each word, and the result is shown in the right plot of Fig 3. We can see that most of the conditional probabilities are around 0.4. Remember that in train data, 35.95% of observations have non-zero dependent variable, so the words with conditional probability around 36% will not be very indicative for the dependent variable.

Model Construction

The flowchart of the model construction is:

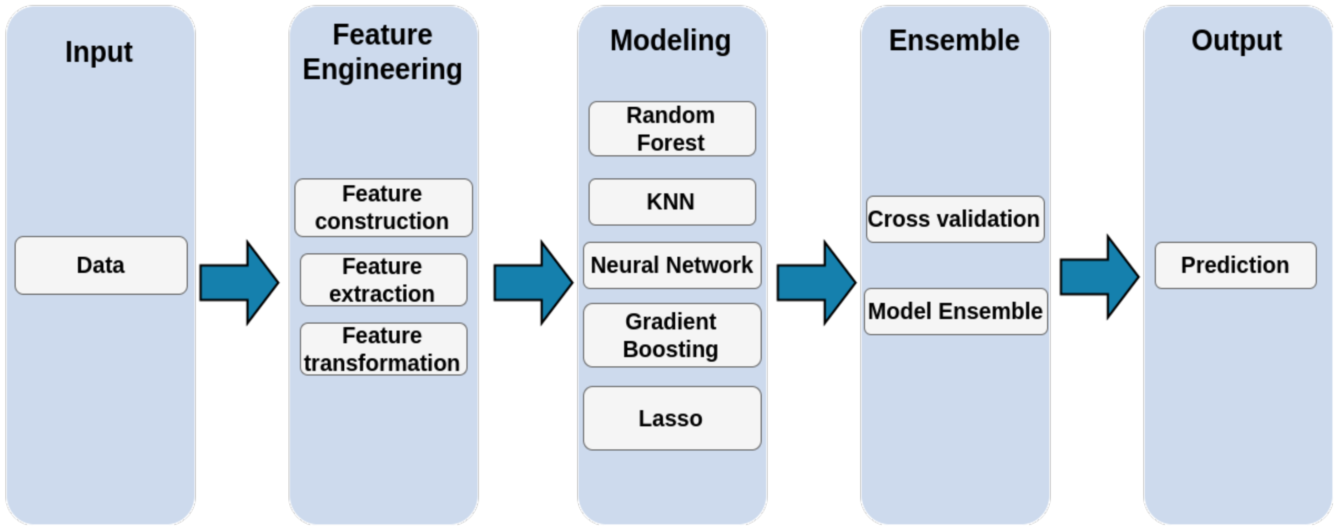

Fig 4. Flowchart

We spend most of our time on Feature Engineering which is effective in improving our final loss value. Meanwhile, The ensemble also reduce the loss value of the single Gradient Boosting Tree by around 0.015, which is also a useful technique.

2. Feature engineering

2.1 Data Transformation

Since the error is represented as:

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

we will transform the target variable y of the train and the test dataset into $\log(y+1)$ and then train and evaluate the models with the transformed data. For linear model and neural network model, we also applied log transformation on the independent variables which are related to number of comments. However, for tree model, log transformation on independent variables doesn't have significant improvement on the result, so we kept independent variables unchanged.

2.2 Feature Selection by Group Lasso

As stated above, although there are as many as 280 independent variables in the dataset, many variables are highly correlated with each other and some of them are redundant. By experimenting on some simple models, we found that many of the independent variables are not very useful. Our intuition is that the model will benefit from dropping some unimportant variables.

We used group lasso for feature selection, because the independent variables can be divided into several groups. It is more reasonable to remove a group of variables as a whole than to remove them one by one.

The definition of groups for the 280 independent variables are shown in Table 1.

Table 1. Group definition of independent variables

Group	Variable Number	Group	Variable Number
1	1 6 11 16 21	10	30 35 40 45 50
2	2 7 12 17 22	11	51 52 53 54 55
3	3 8 13 18 23	12	56 57 58 59 60
4	4 9 14 19 24	13	61 62
5	5 10 15 20 25	14	63~262
6	26 31 36 41 46	15	263 264 265 266 267 268 269
7	27 32 37 42 47	16	270 271 272 273 274 275 276
8	28 33 38 43 48	17	277 278 279 280
9	29 34 39 44 49		

Since the R package we used for group lasso is not very efficient, we randomly sampled 10000 observations to fit the model. 10 different lambda values were used. As lambda increased, group lasso model penalized more heavily on non-zero regression coefficients. Table 2 shows the sequence in which each group of variables were penalized to zero coefficients (red block means the regression coefficient is zero).

Table 2. Zero-coefficient group under different penalty

Group	lambda									
	1e-4	1e-3	1.5e-3	2e-3	2.5e-3	3e-3	5e-3	8e-3	0.01	0.1
1										🟥
2			🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
3			🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
4								🟥	🟥	🟥
5										
6	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
7		🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
8	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
9				🟥	🟥	🟥	🟥	🟥	🟥	🟥
10		🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
11										
12								🟥	🟥	🟥
13										
14					🟥	🟥	🟥	🟥	🟥	🟥
15									🟥	🟥
16						🟥	🟥	🟥	🟥	🟥
17							🟥	🟥	🟥	🟥

* red block means the regression coefficient of the group is zero

From Table 2, we can see that group 6~10, which correspond to variable 26~50 (statistics of number of links of the source website), are probably the most useless variables. Their regression coefficients are penalized to zero under relatively small lambda. Group 5(variable 5, 10, 15, 20, 25), group 11(variable 51~55) and group 13(variable 61, 62) have non-zero regression coefficients under the highest penalty (lambda = 0.1). They are the most influential covariates for the dependent variable.
To check the effect of dropping variables, we dropped each group of variables in the same sequence with group lasso, i.e.:

group 6, 8 → group 7 → group 10 → group 2, 3 → group 9 → group 14 →
group 16 → group 17 → group 12 → group 15 → group 1

And each time a group was dropped, we fitted a gradient boosting tree model on the remaining variables and calculated its test error (we could have used cross-validated group lasso to make decision about which groups of variables to drop, however, the R package "gglasso" is too slow). The results are shown in Fig 4. The most significant improvement of model accuracy comes from dropping 8 groups. The 8th group getting dropped by group lasso was group 14, which corresponds to variable 63~262 (word variables). After that, model test error will increase as more groups of variable are dropped. Thus, we decided to drop variables with group number 2, 3, 6, 7, 8, 9, 10 and 14.

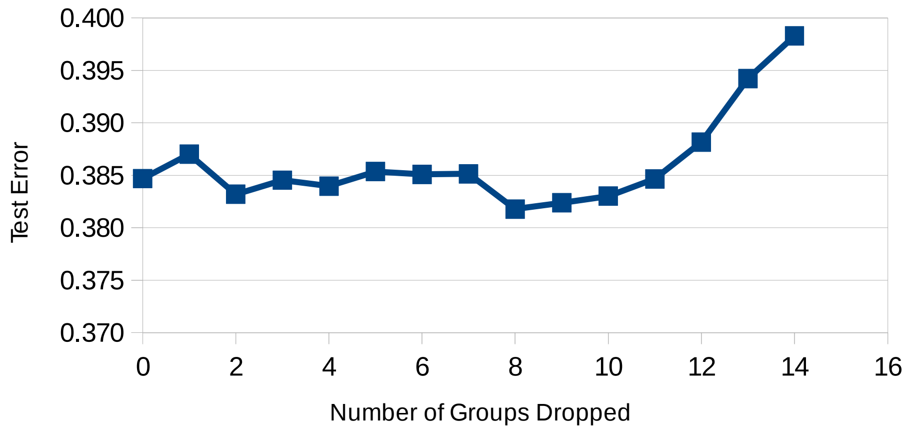

Fig 5. Test error by dropping different number of variable groups

2.3 Feature Generation

With brainstorming and background knowledge, we generated features from the existing variables. The main method of creating new variables are binning variables, aggregating variables and adding interactions among variables. The following table shows the name of the newly generated variables and their source:

Table 3. Feature Generation

Feature Name	Creating Method	Feature Name	Creating Method
N1	V52 / V51	N9	V57 / V58
N2	V53 / V51	N10	V59 / V56
N3	V54 / V51	N11	Binning V61 at 24 hour
N4	V53 / V52	N12	V62 / V61
N5	V52 / V54	N13	Interaction
N6	V57 / V56	N14	Interaction
N7	V58 / V56	N15	Interaction
N8	V59 / V58	N16	Interaction

Some of them bare realistic meaning, such as the ratio of the number of comments 24 hours before the basetime. Some others are created with guess of the nature of the data, for instance, the variables of N13 to N16 reflect whether the date of publication time and the date of Basetime together will have any influence on the number of comments after basetime.

All these 16 variables are added into the model and will be filtered out in the feature selection procedure. Most of them are kept in the final model.

2.4 Feature Extraction with Random Forest

By splitting on variables in a certain sequence, decision tree can discover the interactions between variables. If a decision tree first split on variable X_i, then split on variable X_j, we take (X_i, X_j) as a pair of variables that have potential interaction. As shown in Fig 5 is a simple decision tree with 7 nodes. From this tree we can obtain 6 pairs of variables : (1, 2), (1, 3), (2, 4), (2, 5), (3, 6), (3, 7). In principle, interaction among three variables can also be discovered through this method, but interactions among three or more variables are less likely to appear in the real world, and it's difficult to interpret high order variable interaction, so here we only consider interaction between two variables.

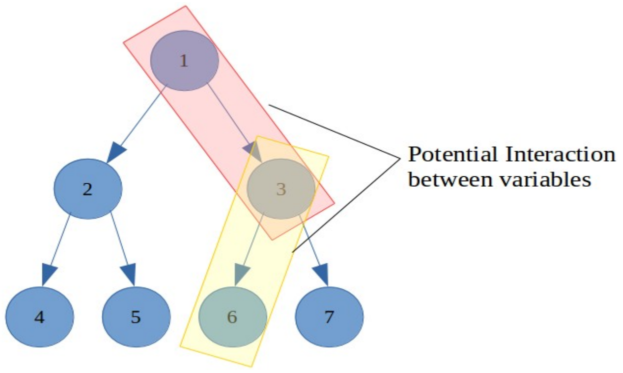

Fig 6. Discover variable interaction from decision tree

We fit a random forest model with 100 decision trees. Parameters were chosen so that the random forest are very "random". Each decision tree was parsed to obtain all potential interaction pairs. These potential interaction pairs were counted and sorted by counts. The pairs with more than 75 counts were selected.

For a selected potential interaction pair (X_i, X_j) , a new variable was create from the product of these two variables:

$$X_{new} = X_i * X_j$$

X_{new} was then added into gradient boosting tree model to test if it would improve the model. The variables that improved the model most were kept for further analysis.

2.5 Features selection

We evaluate the effectiveness of all the features by evaluating the model performance on the test dataset or cross-validation. The features that we will use in the final model building and model evaluation are:

Table 4. Results of Features Engineering

	Selection Result
Features Names	V1,V4,V5,V6,V9,V10,V11,V14,V15,V16,V19,V20,V21,V24,V25,V51,V52,V53,V54,V55,V56,V57,V58,V59,V60,V61,V62,V263,V264,V265,V266,V267,V268,V269,V270,V271,V272,V273, V274,V275,V276,V277,V278,V279,V280,N2,N3,N4,N5,N6,N7, N8,N9,N10,N11,N12,N13,N14,N15,N16,V61_V62,V54_V60,V5_V55,V6_V21,V11_V62,N2_N12,V56_V61

** Variable_Variable represents the interaction of the two*

2.6 Feature importance

To demonstrate the effectiveness of the Feature engineering, we construct a simple Random Forest model and generate the features' importances from it. The top 30 features are:

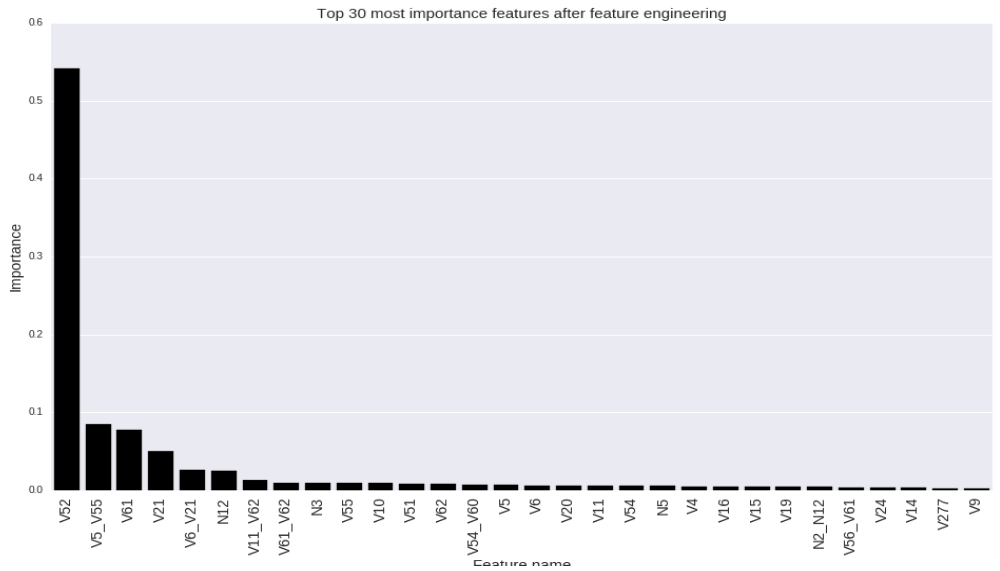

Fig 7. Feature Importance

It is apparent that feature V52 is still the dominant variable in the feature set. Besides we can see that some interaction items or newly generated variables such as V5_V55, V6_V21, N12 and N3 also achieves high level in the feature importance rank, meaning that they are good representative of the pattern of the training data and will be used in the tree splitting when building the random forest model. We will see how the features engineering process make difference in the evaluation of the final model.

3. Model construction

With new features as the input of the model, our next target is to build a model that could achieve the lowest value of the loss function on the test dataset. Thus, we experiment with different models that are popular for regression problem and then tune them to perform well on the test set. We compare their performance and select the best one or two models to construct the final ensemble model.

3.1 Model Comparison

For each model, we only conduct rough tuning on the parameters of them. The performance and computation cost of the models are:

Table 5. Model Setting and Performance

Model and package	Training Time	Parameters	Loss Value
RandomForest (python scikit-learn)	Moderate	n_estimators: 700, criterion: RMSE, max_depth: 12, max_features: Square root of number of features, min_samples_split: 2, min_impurity_split: 0.001	0.39763
KNN (python scikit-learn)	Fast	K = 20	0.63239

Lasso regression (R package:glmnet)	Fast	$\lambda=0.0174$	0.49092
Neural Network (Python Keras)	Very Slow	Layer setting: [512,256,256,50] Activation function: Prelu Optimizer: adadelta	0.42592
Gradient Boosting Tree (Python xgboost)	Moderate	num_boost_round: 600 (number of trees), min_child_weight: 1, learning rate: 0.01, colsample_bytree: 1 (all features), max_depth: 12, subsample: 0.2, gamma: 0.04 (minimum gain for each split), eval_metric: "rmse",	0.37127

We can see that the Random Forest model and the Gradient Boosting Tree model are the best two among the model library we have. The Neural Network take much longer time to train. Meanwhile, we find that the Neural Network model works better on the data set without feature selection (i.e. with all existing features and newly created features added and all the features corresponding to words removed). The main reason is that the newly created features and the word features are highly skewed, which will make the optimization of Neural Network much less efficient. However, since its performance is not the best, we will not spend time on improving the Neural Network model here.
Therefore, after considering the performance of each model and their computation cost, we finally decide to use the Gradient Boosting Tree as our final model.

3.2 Evaluation of Gradient Boosting Tree and Effectiveness of Feature Engineering

We then compare the performance of Gradient Boosting Tree on train and test data with feature engineering and test data without feature engineering. The graph is as follows:

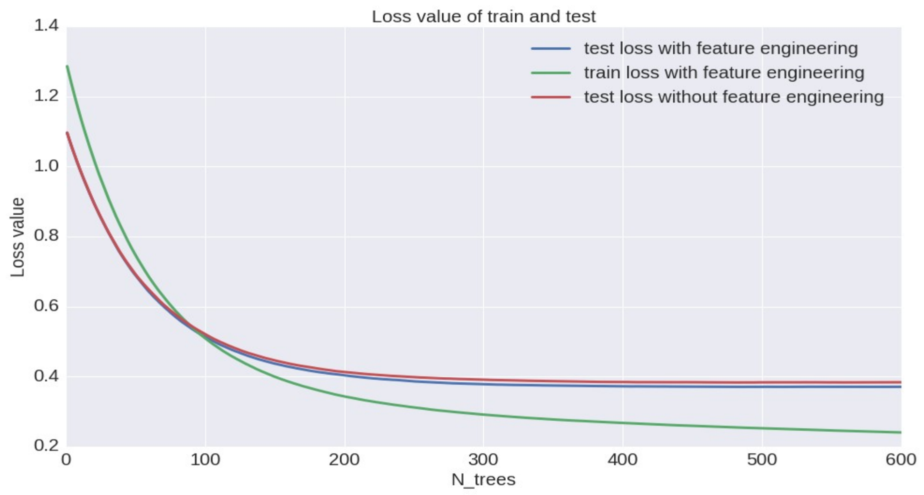

Fig 8. Loss curve

We can see that the loss of training data will always decrease while the loss of test data start to be constant. Meanwhile, we did not see serious over-fitting problem when N_trees become large. Furthermore, We can see the red and blue curve have constant gap between them, meaning the model performs better on data with feature engineering than that without feature engineering, which demonstrates the effectiveness of feature engineering.

3.3 Over-fitting Problem on Test Dataset

Our target is to fit a model on the train data and then evaluate its performance on the test data. We would like to achieve the lowest loss value on the test data. However, if we fit the model on the train and keep tuning the model on the test set, we will fall into the pitfall of over-fitting on the test data. Therefore, we will use the 10 fold cross validation to tune the model on the train dataset and then evaluate its performance on the test dataset.

3.4 Cross Validation and Model Ensemble

We first use the 10-fold cross validation to tune the model on the train data. We train the model on the 9 sections of the train and then tune this model on the other section. However, the results show that the number of trees for each of the 10 models vary greatly, which is hard to just use average to get the tree numbers for the final single model. Thus, we think of the model construction procedure as follows:

Table 6. Modeling Procedure

1. Split the data into K sections (K could be tuned) *2. Train a Gradient Boosting Tree model on (K-1) sections and then tune the model on the other section. Thus, we get K separate models.* *3. Apply the K models on the test data and get K set of predictions* *4. Average the K set of predictions as the final prediction of the test data*

The advantage of this method are:

1. This method is similar to bagging method, which will prevent over-fitting on the train dataset.
2. We does not use much information of the test data, which alleviate the over-fitting of test data.
3. Ensemble of multiple models may improve the performance of a single model.

After tuning the *K*, we find that ***K* = *8*** performs the best on the test data

Table 7. Comparison between Single Model and Ensemble Model

	Loss value
Single model	0.37128
Ensemble model	0.36985

The ensemble model is useful in improving the model preformace on the test dataset. We can find that the single Gradient Boosting Tree with 450 trees could only achieve the loss of 0.37128. After the ensemble, however, the loss value is reduced to 0.36985, improved by 0.015.

4. Summary of model

Thus, Ensemble of Gradient Boosting Tree will be our final model for this project, and our best loss value for this project is **0.36985**.

5. Discussion

After playing around with the data set for several weeks, we have detected something that may be useful experience.

1. In the model construction process, we split the data into 10 folds for model training and cross validation. However, this behavior is against the setting of the data, which may cause the data leakage when training the model. But if we did not do so, we have to tune the model on the test data and may face the problem of over-fitting on the test data, which is not desirable. Thus, it is better to obtain a validation dataset, which is just for the tuning of the model. After that, we can test the performance of our model on the test dataset without any concerns.

2. When constructing the models, we tried to take logarithm of the features we used. Taking logarithm will make the features less skewed and more close to a normal distribution. The log transformation will not have much influence on the tree model. However, for the Neural Network model, it will be a good way to improve. The reason is that after log transformation, the shape of space of the loss function will get close to a circle or sphere, making the decent of the loss function fast and stable.

6. Future work

1. We could consider more complicated ensemble method for the improvement of the loss value.
2. Text mining methods could be used to extract useful information from the word features.
3. In experiments, we try to use Bagging instead of using the modeling step we have right now. However, due to the huge computational cost of Bagging and the limitation of time, we finally choose give it up. Thus, we could try whether Bagging will perform better than our current method in the future.

Reference

[1] Kaggle_CrowdFlower: URL: *https://github.com/ChenglongChen/Kaggle_CrowdFlower*
[2] Kaggle Otto Challenge: How we achieved 85th out of 3,514 and what we learnt: URL: *http://www.slideshare.net/eugeneyan/kaggle-otto-challenge-how-we-achieved-85th-out-of-3845-and-what-we*
[3] Kaggle Ensemnle Guide, 2015: URL: http://mlwave.com/kaggle-ensembling-guide/
[4] Caruana R, Niculescu-Mizil A, Crew G, et al. Ensemble selection from libraries of models[C]//Proceedings of the twenty-first international conference on Machine learning. ACM, 2004: 18.
[5] Friedman J H. Greedy function approximation: a gradient boosting machine[J]. Annals of statistics, 2001: 1189-1232.
[6] Singh K, Kaur R, Kumar D. Comment Volume Prediction Using Neural Networks and Decision Trees[C]//Proceedings of the 2015 17th UKSIM-AMSS International Conference on Modelling and Simulation. IEEE Computer Society, 2015: 15-20.
[7] Yu H F, Lo H Y, Hsieh H P, et al. Feature engineering and classifier ensemble for KDD cup 2010[C]//Proceedings of the KDD Cup 2010 Workshop. 2010: 1-16.
[8] Domingos P. A few useful things to know about machine learning[J]. Communications of the ACM, 2012, 55(10): 78-87.
[9] Sun Q, Pfahringer B. Bagging ensemble selection[C]//Australasian Joint Conference on Artificial Intelligence. Springer Berlin Heidelberg, 2011: 251-260.
[10] GUYON, I. AND ELISSEEFF, A. 2003. An introduction to variable and feature selection. J. Mach. Learn. Res. 3, 1157–1182.

Appendix – Failed but Promising Experiments

a.1 Frequent pattern mining on word features

Initially, we think there might be different topics in the blog posts, and words belong to one topic will tend to appear in the same blog posts, so by finding words frequently appear together, we can discover these latent topics.

We used Apriori algorithm – a frequent pattern mining algorithm to obtain those groups of words and added corresponding new variables into our model. However, this method proved to be not useful for this dataset.

a.2 SIR on word features

We didn't give up exploring word features. The next method we tried was SIR. Our idea was that there might be some "directions" in the space of word features that had more influence on y. By discovering these "directions", we can reduce the dimension of word features. We found 4 directions from SIR analysis. But projecting word features onto these four directions didn't improve our model.

a.3 PCA

We also tried to use PCA to reduce the dimension of the word features, however, after adding the several important extracted principle components back to the dataset, the performance of the original model is jeopardized. Thus, if we would like to use the word data further, more complicated model for text mining should be applied.

a.4 Ensemble of different models

We try to take a weighted average of the predictions from the Random Forest and Gradient Boosting Tree. However, since the performance of the two models differ much, any combination will make the loss value worse. Furthermore, we did not consider any other models because they perform relatively bad compared to Gradient Boosting Tree.

a.5 Clustering before modeling

We try to cluster the data based on their average values the V51 to V60 variables and build the models separately. However, we find that the number of clusters are extremely imbalanced and the performance of modeling precess on the clusters is bad.