ISOM 674 Machine Learning Final Project Report

Group 11: Kwaku Danso-Manu, Yan Li, Miller Page, Kami Wu

Abstract

For this Class Probability Estimation problem, we conducted feature engineering to reduce dimension as the first step. We fitted 5 models for 7 classifiers including decision tree, random forest, naive bayes, logistic regression (both lasso regression and ridge regression), K-nearest neighbors and neural networks respectively, then we used ensemble methods for both the same sets of models and the best models from each classifier. During model fitting process, we used decision tree and lasso regression to determine important features for different models. Surprisingly, the best model we got was a ridge regression model without feature selection.

Data Exploration

Data exploration is an important path to better understanding the data and gaining insights for the following analysis. Since almost all attributes in our data were categorical variables, we examined the number of categories of each variable in both training data and test data. The outcome can be found below.

Variables	N. of categories in training data	N. of categories in test data
C1	7	7
banner_pos	7	7
site_id	>1000	>1000
site_domain	>1000	>1000
site_category	26	25
app_id	>1000	>1000
app_domain	526	395
app_category	36	31
device_id	>1000	>1000
device_ip	>1000	>1000
device_model	>1000	>1000
device_type	5	5
device_conn_type	4	4
C14	>1000	>1000
C15	8	8
C16	9	9
C17	407	472
C18	4	4
C19	66	68
C20	171	170
C21	55	62

Figure 1: Number of categories in each variable

From the table, first, we can see that some variables had more than a thousand unique categories. Since we were dealing with categorical variables, these variables will dramatically increase the dimension of models which will make the model too

complex and potentially cause issues. This told us we should do some feature engineering to regroup these categories and decrease the dimension.

Secondly, for some variables, training data and test data didn't have the same number of categories. This will cause trouble when we predict, and we should take it into consideration when we regroup these variables. Moreover, the variable "hour" contained the date and hour information, we can extract this information from it and transform them into a more informative format.

Last but not least, the classes of the target variable "click" was uneven, the class "0" was approximately 5 times more than the class "1". Uneven target variable might confuse the model and give us a very biased predicting result. We need to take it into consideration when we fit the model.

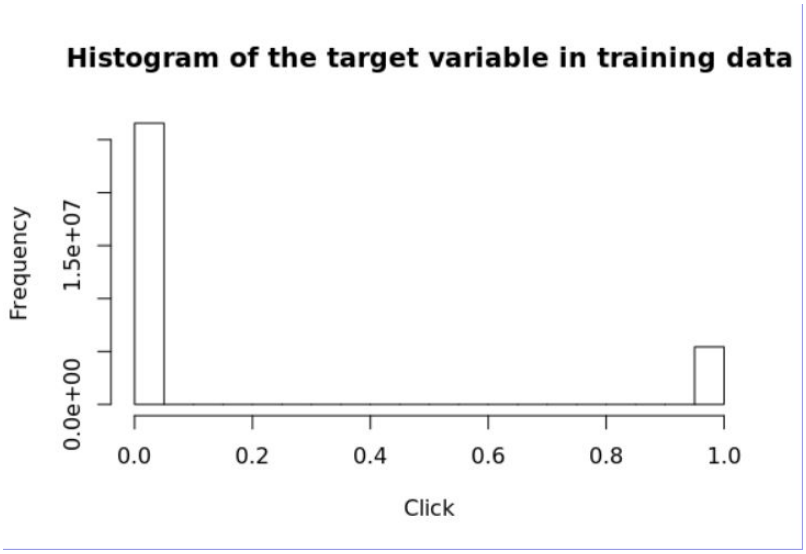

Figure 2: Histogram of the target variable in training data

Methodology

After deeply understanding the data and the problem we want to solve, we designed the methodology of the modeling process as shown below.

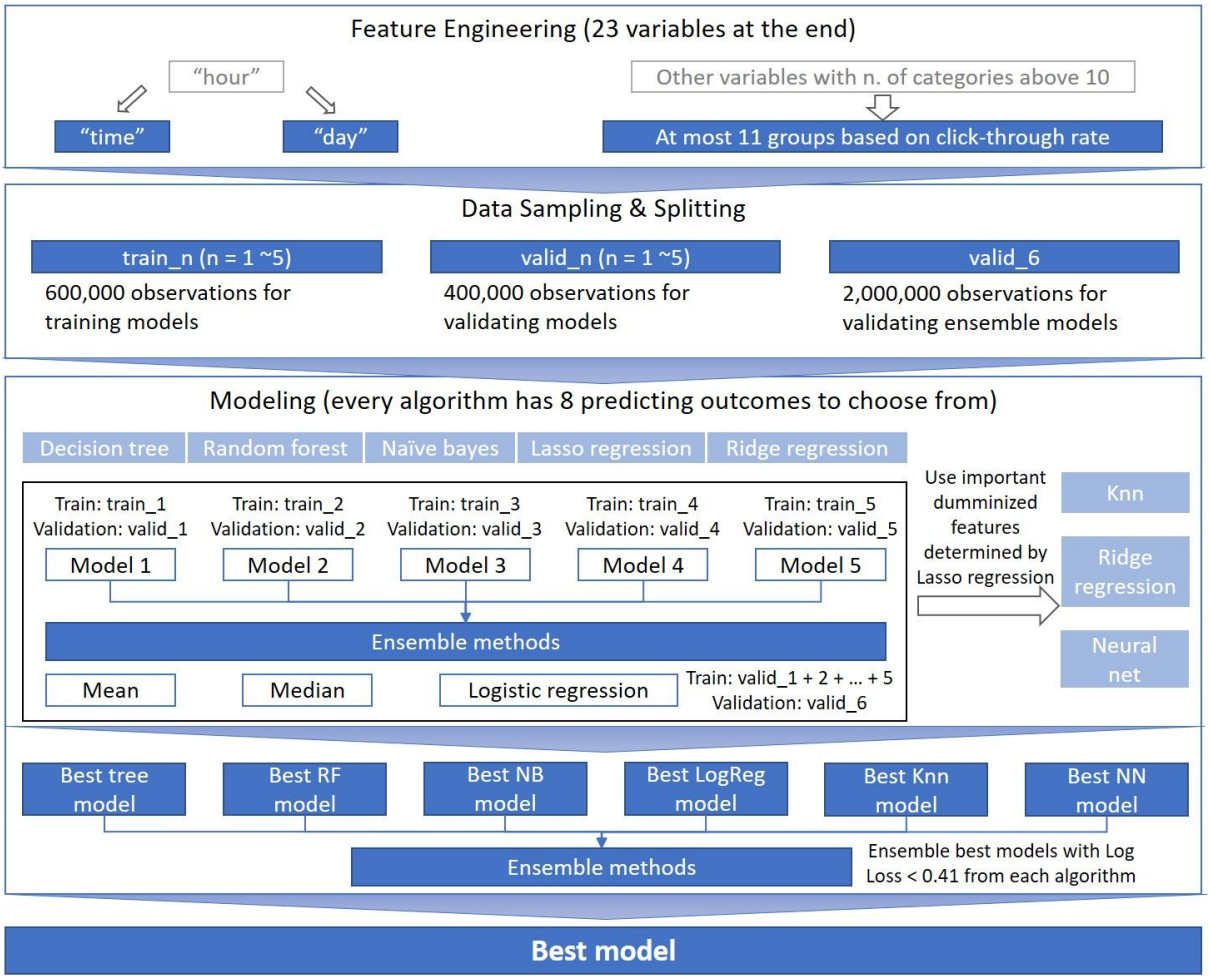

Figure 3: Methodology of the modeling process

First, we transformed the variable "hour" and other categorical variables that have more than 10 categories. For the "hour", we extracted the digits representing hours and the digits representing date out. The 24 hours were then divided into 8 time intervals/categories in a new variable called "time", where each interval had 3 hours. The date was then categorized into a new variable "day" which had 2 categories, "weekday" and "weekend" . For other categorical variables, our goal was to reduce the dimension and regroup the categories into meaningful groups. Thus, we decided to find those high-frequency categories in each variable, and group them based on the click-through rate of each category. Basically, if a category had a click-through rate from 0-10%, it will be categorized as group "1"; if a category had click-through rate from 10-20%, it will be categorized as group "2"; and so on. However, for those low-frequency categories, the click-through rate will be biased due to limited samples. And given that there might be new categories in test data, we decided to group those low-frequency categories and new categories as "other". After such feature engineering, each variable will have at most 11 categories, which achieves our goal of reducing the dimension.

After transforming both the training data and the test data, we needed to sample the data and split them into training and validation sets. Since we have a large volume of data, we decided to use 5 training and validation data sets and build 5 different models accordingly. We will then use ensemble methods to ensemble these 5 models, creating a need for another validation data set to validate the performance of the ensemble model.
The 7 classifiers we chose to use are decision tree, random forest, naive bayes, logistic regression (both lasso regression and ridge regression), K-nearest neighbors and neural networks.

Although the number of variables in the data was not large, we thought it's necessary to select features for some models to increase the model performance. But we had to carefully select variables for the different classifiers as they don't all use the same type of input variables. For example, categorical variables are required by decision tree, random forest and naive bayes while categorical variables need to be transformed into dummy variables for logistic regression, Knn and neural network. Since decision tree and lasso regression can choose important features for us, we decided to use decision tree to determine the important features for naive bayes, and use lasso regression to determine the important features for ridge regression, Knn and neural network.

After we built 5 different models for each classifier, then we will combine those models and create 3 overall prediction functions based on means, medians and logistic regression. The best models of each classifier will be chosen if it's log loss is good enough, and then will be combined and be used to create 3 ensemble models. Finally, we will choose the best model from all of these models based on the value of log loss, and we will use the best model to do prediction.

Execution Summary

1. **Feature engineering**

The transformation of variable "hour" has been stated in the previous paragraph, so we won't cover this again. For other categorical variables, we first counted the frequency of each category and examined the quartiles of the frequency because we wanted to set a reasonable threshold of low frequency and high frequency. However, the frequency of categories in some variables was extremely uneven. For example, we got 2 as the third quartile of variable "device_id", and 20720 as the first quartile of variable "C21". For these variables, we decided to find ourselves a rational threshold, and then categorized the categories that had a frequency lower than the threshold as low-frequency categories and labeled them as "other". After getting high-frequency categories, we calculated the corresponding click-through rate of each category, and categorized them into at most 10 groups based on the click-through rate. At last, we got 23 features as total with reduced dimension.

2. **Data sampling and splitting**

For data sampling, we decided to do random sampling because random samples will have similar characteristics as the test data, thus the model can capture similar patterns. Although in our trial test, tree models arbitrarily predict every observation into the majority group, we found a way to solve this by adding weights to the model. We set a different random seed when we sampled each data set so there might be overlaps in different samples. We sampled 1,000,000 observations 5 times and did a 60-40 split and then got 5 training and validation data sets. We sampled another 2,000,000 observations for validating ensemble models, which we will call valid_6.

3. **Modeling & Evaluation**

   a. **<u>Decision tree</u>**

For decision tree, we built 5 models for both weighted and unweighted target variable respectively by using package “rpart” and then pruned it based on the value of xerror. The reason why we didn’t use package “tree” was that it kept giving us error when fitting the model. The unweighted models performed generally better than the weighted ones, as the Log Loss of unweighted models were routinely around .416 while the Log Loss of weighted models were routinely around .604. However, when we built ensemble models based on both the unweighted and weighted models respectively, the best ensemble model was the one based on weighted models by using logistic regression. This model had a Log Loss of .403. This was the best Decision Tree model and it will be used to build the final ensemble models.

   b. **<u>Random forest</u>**

For random forest, we tried different numbers of “mtry” and found that the best “mtry” with the lowest Log Loss for all models was between 11 to 14. However, the performance of random forest was terrible in this case. The Log Loss of random forest models were around 3.3. Nevertheless, again, the ensemble method using logistic regression dramatically improve the performance. We got a Log Loss as .433 for the logistic regression ensemble model.

   c. **<u>Naive bayes</u>**

For naive bayes, the correlation between features and target variable matters a lot. If we put insignificant variables into the model, such noise will harm the performance of naive bayes. Thus, we decided to use the important features determined by the best decision tree model to train the naive bayes models. We used the function “varImp” from the “caret” package which can calculate the importance scores of features. We then sorted them by importance score and got a list of important features. Another problem we encountered dealt with how many important features we should select. We decided to use a for loop, and put top n important variables into naive bayes everytime, then validate different models based on their Log Loss value. The number of variables we used would increase as the n getting larger. Surprisingly, the best model we got was always the model with only one variable which was the most important variable. We thought this was because

as more and more features were added, the noise was increased as well, thus the model got worse with more variables. The Log Loss of naive bayes including the ensemble ones were around .42 to .43.

d. Logistic regression - Lasso regression

For lasso logistic regression, we used the “glmnet” function with the family “binomial” and a grid of lambda to search the best lambda. The grid is from 0.001 to 100000, the best lasso logistic regression model out of the 5 samples is the one with a lambda of 0.001. This model gives us a Log Loss of 0.4009114. We also tried ensemble the 5 lasso regression together using logistic regression. We trained the ensemble model. The Log Loss given by the ensembled lasso regression is 0.4074958 on valid_6.

Since the best lambda seems to be the lowest value in the grid, we also tried to set the grid range as 0.001 to 0.00001. The performance did increase by a bit. However, using a lower lambda will increase the number iterations and prolong the running time so that we still decided to use the previous grid.

e. Logistic regression - Ridge regression

The ridge regression process is the same for the lasso regression. The grid to search the best lambda for ridge regression is from 0.001 to 100000, the best ridge logistic regression model out of the 5 samples is the one with a lambda of 0.001. This model gives us a Log Loss of 0.4001927 - which is the best model so far. We also tried ensemble the 5 ridge regression together using logistic regression. We trained the ensemble model, and the Log Loss given by the ensembled lasso regression is 0.4069258 on valid_6.

Considering the number of variables after transforming to dummies is as high as 139 and models such as Knn requires manual feature selection unlike ridge and lasso. We decided to take a look at the important features given by the lasso regression using the function “varImp” from the “caret” package. We then took the variables that have a variance greater than 0 as the important variables and trimmed down the feature space of the training and validation data from 139 to around 75. Using the trimmed data, we performed another 5 ridge regressions, but the results are not as good as the ridge regression with full feature space. However, we can still use the important variables given by lasso regression in the Knn modeling process to minimize unnecessary computing time.

Since the performance given by all of the logistic regression models are as close as having only a 0.01 difference in log loss, we also tried using the same validation data for all of the variables to select the model. The results remained similar, so we still argued that the ridge regression with a lambda of 0.001 is the best model so far.

f. K-nearest neighbors

For K-NN, we built the 5 models using the 25 most important dummy variables found from logistic regression on the same 5 training and validation data sets as talked about above. We decided to stick with the top 25 variables (opposed

to top 75) because we thought that the K-NN model would take way too much computation time to make it worth while. Thus, we thought the first 25 would give us a good balance of model and computation performance. The results from these five models were not as strong as other methods, as the best k gave us a Log Loss was routinely around 1.8 (compared to out better models being under .5). We found our best k for each model by using a for loop that looped over every odd value of k (since odd values work better in K-NN due to the tiebreaker) from 11 to 99 and found the k that gave us the smallest Log Loss. We found that the logistic regression ensemble method was much better, as it gave us a Log Loss of .488. However, this value was still higher than the other models.

g. Neural network

As with our K-NN model above, we run through several iterations using the 25 most important variables as well and using the binary classification with softmax approach. Both features and target variables were on a 0-1 scale so there was no need to rescale the features for the neural network. Amongst the activation functions, the rectified linear unit generally had the best performance so we tuned and modified the network to unearth our best model. The simplest model of one layer with four neurons yielded a log loss of 0.434. We now tried adjusting the model to be deeper and wider and examined several results. Four layers of four neurons each made performance only slightly worse with a loss of 0.435. Making the network wide with one layer of 10 neurons also made performance worse with a log loss of 0.461. Attempting a wide and deep model now of four ReLu layers of ten neurons each dropped performance drastically to 1.153. This trend indicated that the more complex we made the neural net, the worse performance got, so we now tried slight modifications of the model which provided the best log loss. The final network that with the lowest log loss of 0.418 was one with four ReLu layers of 3 neurons each.

h. Ensemble methods

The first ensemble method with different algorithms we tried was an logistic regression of the predictions from the tree ensembled method and the best ridge logistic regression. We believed it was reasonable to just use the best two models that give similar log loss outcomes. However, this ensembled method didn't give us a better outcome on the validation data. The log loss is 0.4048252, which is higher than both the tree ensemble model and the best ridge regression model.

Since the performance of random forest, naive bayes, K-NN and Neural network is not as good as the ensembled tree and the ridge logistic regression, as well as the lengthy running time of the neural network algorithm, we decided not to ensemble all of the algorithm together.

4. Predictions

The following table shows all the models we have tried and their performance outcome.

Algorithm	Model	Ensemble Method	Train Sample	Valid Sample	Performance on the Validation Data
tree	tree_weighted_1		1	1	0.60256
	tree_weighted_2		2	2	0.60376
	tree_weighted_3		3	3	0.60361
	tree_weighted_4		4	4	0.60264
	tree_weighted_5		5	5	0.60423
	tree_ensemble_mean	mean	1,2,3,4,5	1,2,3,4,5	0.60144
	tree_ensemble_median	median	1,2,3,4,5	1,2,3,4,5	0.60262
	tree_ensemble_logreg	logistic regression	1,2,3,4,5	6	0.40339
random forest	rf_1		1	1	3.37425
	rf_2		2	2	3.33316
	rf_3		3	3	3.24547
	rf_4		4	4	3.31526
	rf_5		5	5	3.38231
	rf_ensemble_mean	mean	1,2,3,4,5	1,2,3,4,5	2.53849
	rf_ensemble_median	median	1,2,3,4,5	1,2,3,4,5	3.27956
	rf_ensemble_logreg	logistic regression	1,2,3,4,5	6	0.43357
naive bayes	nb_1		1	1	0.42458
	nb_2		2	2	0.42411
	nb_3		3	3	0.42390
	nb_4		4	4	0.42265
	nb_5		5	5	0.42480
	nb_ensemble_mean	mean	1,2,3,4,5	1,2,3,4,5	0.42400
	nb_ensemble_median	median	1,2,3,4,5	1,2,3,4,5	0.42400
	nb_ensemble_logreg	logistic regression	1,2,3,4,5	6	0.42483
logistic regression	lasso 1		1	1	0.40266
	lasso 2		2	2	0.40253
	lasso 3		3	3	0.40174
	lasso 4		4	4	0.40091
	lasso 5		5	5	0.40315
	ridge 1		1	1	0.40185
	ridge 2		2	2	0.40170
	ridge 3		3	3	0.40084
	ridge 4		4	4	0.40019
	ridge 5		5	5	0.40234
	ridge_trim		1	1	0.42076
	ridge_trim		2	2	0.41289
	ridge_trim		3	3	0.44999
	ridge_trim		4	4	0.42570
	ridge_trim		5	5	0.41712
	ridge_trim_2		1	1	0.43617
	ridge_trim_2		2	2	0.43824
	ridge_trim_2		3	3	0.44108
	ridge_trim_2		4	4	0.43712
	ridge_trim_2		5	5	0.43819
	lasso_ensemble_logreg	logstic regression	1,2,3,4,5	6	0.40750
	ridge_ensemble_logreg	logstic regression	1,2,3,4,5	6	0.40693
	ridge_lasso_ensemble_logreg	logstic regression	1,2,3,4,5	6	0.40688
	lasso 1		1	6	0.40273
	lasso 2		2	6	0.40277
	lasso 3		3	6	0.40283
	lasso 4		4	6	0.40272
	lasso 5		5	6	0.40276
	ridge 1		1	6	0.40191
	ridge 2		2	6	0.40192
	ridge 3		3	6	0.40192
	ridge 4		4	6	0.40194
	ridge 5		5	6	0.40192
knn	knn_trim_1		1	1	1.80220
	knn_trim_2		2	2	1.83699
	knn_trim_3		3	3	1.81388
	knn_trim_4		4	4	1.79309
	knn_trim_5		5	5	1.74827
	knn_ensemble_mean	mean	1,2,3,4,5	1,2,3,4,5	1.56806
	knn_ensemble_median	median	1,2,3,4,5	1,2,3,4,5	1.72793
	knn_ensemble_logreg	logistic regression	1,2,3,4,5	6	0.48881
neural net	one relu layer, 4 neurons		train trim 1	valid trim 1	0.43400
	four relu, 4 neurons		train trim 1	valid trim 1	0.43500
	one relu, 10 neurons				0.46100
	four relu, 10 neurons				1.15300
	4 relu, 3 neurons				0.41800
	6 relu, 3 neurons				0.41900
	4 relu 2 neurons				0.43400
ensemble (different	ensembled_tree + ridge 4	logistic regression	1,2,3,4,5	6	0.40483
	ensembled_tree + ridge 4	mean/median	1,2,3,4,5	6	0.51619

Figure 4: Performance summary

After choosing the ridge logistic regression model with a lambda of 0.001 as the best model.

Variable name	Coefficient
(Intercept)	-5.8796
C11002	0.0411
C11005	0.1082
C11007	-0.4118
C11008	-0.4342
C11010	0.0301
C11012	0.1401
banner_pos1	0.0868
banner_pos2	0.0683
banner_pos3	0.2285
banner_pos4	0.0213
banner_pos5	0.7941
banner_pos7	-0.0963
device_type1	-0.0393
device_type2	0.0000
device_type4	0.0571
device_type5	-0.1336
device_conn_type2	0.0631
device_conn_type3	-0.1413
device_conn_type5	-0.3504
C15216	-0.1122
C15300	0.0494
C15320	0.0345
C15480	0.0295
C15728	0.2912
C15768	0.1825
C151024	0.6577
C1636	-0.1120
C1650	0.1761
C1690	0.2910
C16250	-0.1103
C16320	0.0296
C16480	0.0211
C16768	0.6526
C161024	0.1798
C181	-0.2386
C182	0.0232
C183	0.0822
timebeforedawn	-0.0077
timedawn	-0.0028
timeforenoon	0.0102
timemidnight	-0.0920
timemorning	-0.0104
timenight	-0.1352
timenoon	0.0479
dayweekend	-0.0118
site_id_group1	0.5735
site_id_group2	0.8390
site_id_group3	1.2960
site_id_group4	1.5347
site_id_group5	1.3322
site_id_group6	1.7749
site_id_group7	2.2678
site_id_groupother	0.9021
site_domain_group1	0.4525
site_domain_group2	0.6499
site_domain_group3	0.6948
site_domain_group4	0.8882
site_domain_group5	1.2969
site_domain_group6	1.3480
site_domain_group7	1.9368
site_domain_groupother	0.4948
app_id_group1	0.8269
app_id_group2	1.4400
app_id_group3	1.7774
app_id_group4	2.4781
app_id_group5	2.4809
app_id_group6	2.8124
app_id_group7	3.6366
app_id_group8	3.8207
app_id_group9	5.0380
app_id_groupother	1.1678
app_domain_group1	-0.0782
app_domain_group2	-0.0887
app_domain_group3	5.5026
app_domain_group4	-5.4251
app_domain_group6	-0.1751
app_domain_groupother	-0.9915
device_id_group1	0.6271
device_id_group2	0.8180
device_id_group3	0.6986
device_id_group4	0.2742
device_id_group5	0.7671
device_id_group6	2.1009
device_id_group7	1.6514
device_id_group8	3.5373
device_id_group9	5.3524
device_id_groupother	0.5737
device_ip_group1	0.3462
device_ip_group2	0.5595
device_ip_group3	0.9032
device_ip_group4	1.2575
device_ip_group5	1.7034
device_ip_group6	1.8096
device_ip_group7	2.5684
device_ip_group8	2.6672
device_ip_group9	4.0569
device_ip_groupother	0.4684
device_model_group1	0.1106
device_model_group2	0.3568
device_model_group3	0.3882
device_model_group4	0.8705
device_model_group5	-0.4788
device_model_groupother	0.1767
C14_group1	0.3697
C14_group2	0.5087
C14_group3	0.6793
C14_group4	0.9506
C14_group5	1.2004
C14_group6	-0.7174
C14_group7	-0.4222
C14_groupother	0.2384
C17_group1	0.0194
C17_group2	0.0765
C17_group3	-0.0384
C17_group4	0.0121
C17_group5	0.0623
C17_group6	1.6743
C17_group7	1.2136
C17_groupother	-4.1602
C19_group1	-0.0807
C19_group2	-0.0341
C19_group3	0.0627
C19_group5	-0.2621
C19_groupother	0.0000
C20_group1	0.0445
C20_group2	0.1217
C20_group3	-0.4521
C20_groupother	0.0261
C21_group1	0.0613
C21_group2	0.0792
C21_group3	0.0706
C21_groupother	-3.8892
site_category_group1	0.3053
site_category_group2	0.1438
site_category_group4	0.0360
site_category_groupother	-3.7217
app_category_group1	0.0466
app_category_group2	-0.1334
app_category_groupother	0.0000

Figure 5: Coefficients of the best model - ridge logistic regression(lambda = 0.001)

Since all the categories in the test data have already been regrouped and the “glmnet” function will transform the categorical variables into dummies itself, we then used the “predict” function to predict the probability of click for each observation in the test data. The predictions are included in the “ProjectSubmission-Team11.csv”. It is also accessible through this link in case the file does not work:

Conclusion

After trying so many models, we found that the logistic regression ensemble method worked extremely well for those bad-performance models, while it didn’t improve the performance for those models that were good enough.

In the end, we decided to go with the model that gave us the lowest Log Loss on our validation data. The best model is the ridge regression model with a lambda of 0.001 and 139 dummy variables. The process of finding the best model has been a tradeoff between better outcomes and the longer running time. However, we believe that this best model should be considered a relatively decent model.

Appendix

Code file 1 - Project-Code1-Team11.R:

This R file contains all of our code except the neural network part. This file has 10 parts including:

Part 1: Feature engineering - transform variables for both the training data and the test data

Part 2: Data sampling and splitting - sample 5 sets of training data and 6 sets of validation data

Part 3: Modeling - Decision Tree - modeling process of decision tree

Part 4: Modeling - Random Forest - modeling process of random forest

Part 5: Modeling - Naive Bayes - modeling process of naive bayes

Part 6: Modeling - Logistic regression - modeling process of logistic regression including lasso regression and ridge regression

Part 7: Modeling - K-nearest neighbors - modeling process of Knn

Part 8: Modeling - Neural network - write out data for neural network

Part 9: Modeling - Ensemble methods - combine the best models with Log Loss lower than 0.41

Part 10: Prediction - use the best model to predict test data

Code file 2 - Project-Code2-Team11.py:

This Python file contains the modeling process of neural network part.