ISOM 674 - Machine Learning I Final Project: Prediction of Clicks for On-Line Advertisements

Rush Bhardwaj, Yi Cai, Meng Wang, Nicholas Yang

INTRODUCTION

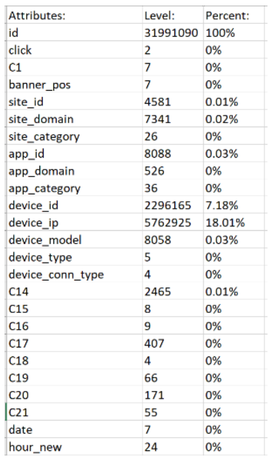
Today, data is abundant and is becoming complimentary to analytics. Companies are trying to find new ways to enhance its user's experience and build a stronger, long lasting and satisfying relationship with its users. Data science is helping define the future of analytics and simultaneously helping companies get a competitive edge to be the front runners in capturing market share and customer retention.

The goal of the project is to leverage machine learning techniques to analyze predicting clicks for on-line advertisements. We analyze data which little over 45 million observations and 24 different variables to make our predictions. Our outcome variable is a binary categorical variable, 1 representing an ad was clicked on and 0 being an ad was not clicked on. We go through a rigorous process of model building and parameter selection of the best fit model and the performance criterion used to evaluate the performance is the log loss function.

DATA PREPERATION

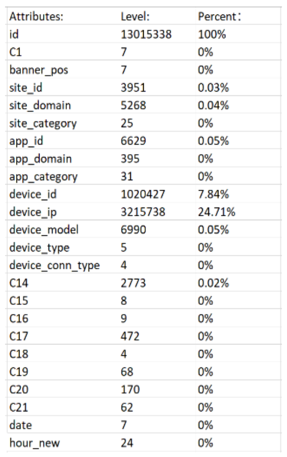
There are two sets of data provided to conduct the project - ProjectTrainingData and ProjectTestData. ProjectTrainingData contains 31991090 rows and 24 columns of data. ProjectTestData contains 13015341 rows and 23 columns of data (without target variable "click"). Including the target variable "click", all the data in both datasets are categorical data. For the convenience of the future validation process, we decided to clean the whole training data set and then subset the data for computational efficiency. Hence, with the purpose of exploring the data as well as preparing the data, we noticed the data entries for variable “hour” are arranged in the format of “YYMMDDHH”. Since we only have 9 days’ data, there is no data variation for Year and Month. Then it is intuitive to split this variable into two - “date” and “hour_new”, which represent the weekdays and hour information as we assume time will affect customers’ interests in clicking certain advertisements. For weekdays information, we created new categorical attribute levels to represent the information - Monday through Sunday. In such a way, we are able to code this information for the test data set as test data set’s date information has more attribute levels than training data set. With very limited information provided for the rest of the x variables, we decided not to create more attributes and attributes’ levels.

Thus, we decided to conduct manual feature selection based on the existing attribute level as shown below in figure 1 and 2.



Attributes:	Level:	Percent:
id	31991090	100%
click	2	0%
C1	7	0%
banner_pos	7	0%
site_id	4581	0.01%
site_domain	7341	0.02%
site_category	26	0%
app_id	8088	0.03%
app_domain	526	0%
app_category	36	0%
device_id	2296165	7.18%
device_ip	5762925	18.01%
device_model	8058	0.03%
device_type	5	0%
device_conn_type	4	0%
C14	2465	0.01%
C15	8	0%
C16	9	0%
C17	407	0%
C18	4	0%
C19	66	0%
C20	171	0%
C21	55	0%
date	7	0%
hour_new	24	0%

Figure 1: Attribute Level of Training Data



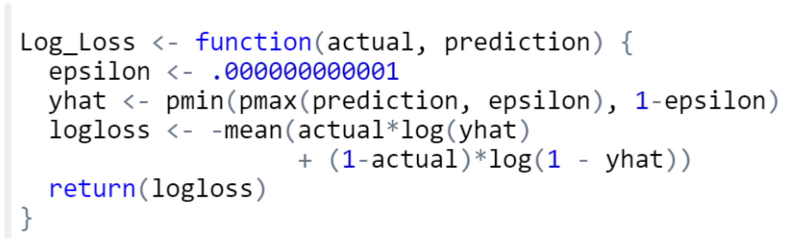
Attributes:	Level:	Percent：
id	13015338	100%
C1	7	0%
banner_pos	7	0%
site_id	3951	0.03%
site_domain	5268	0.04%
site_category	25	0%
app_id	6629	0.05%
app_domain	395	0%
app_category	31	0%
device_id	1020427	7.84%
device_ip	3215738	24.71%
device_model	6990	0.05%
device_type	5	0%
device_conn_type	4	0%
C14	2773	0.02%
C15	8	0%
C16	9	0%
C17	472	0%
C18	4	0%
C19	68	0%
C20	170	0%
C21	62	0%
date	7	0%
hour_new	24	0%

Figure 2: Attribute Level of Test Data

As we can see from the figures above, both “device_id” and “device_ip” has a rather high attribute level, taking more than 5% of each column of data. Hence, it’s reasonable to drop these two variables from our predicting attributes' list. To be noticed, "id" attribute was also excluded from the future model training process intuitively.

Next step, we created a validation data from the existing training data with a 30% ratio. However, even 30% of training data is still too large to be computed efficiently. Hence, we created subset the training data in three versions - original, one million, and 100k. Intuitively, original training data is the Training Data; One million and 100K versions of training data contains one million rows and 100,000 rows respectively of training data after sampling. Similarly, we created corresponding versions of validation data and test tata. To be noticed, different versions of test data do not differ in data length but were created to cater different attribute level trimming process, which we will introduce in the next paragraph to deal with issues caused by different attribute levels.

Finally, we noticed that new attribute categories are introduced for most x variables in test data comparing to training data. Hence, we decided to re-categorize each x variable with respect to the categorical variable attribute level limitation of 32 for Decision Tree Classification Method (attribute level trimming process). Our approach is sorting the training data based on frequency, keeping the first 31 categories unchanged, and re-category everything else as class "General". In this way, each attribute of any versions of training data would at most has an attribute level of 32. For the corresponding versions of validation and test data, categories for each attribute in validation and test data were re-categorized based on its existence in the list of 31 most frequent training attribute categories. If a validation or test attribute category exists in the list of 31, it would be kept unchanged. Vice versa, if a validation or test attribute category does not exist in the list of 31, it would be classified as class "General". Hence, after such a process, each attribute of all versions of validation data and test data would have an attribute level smaller than or equal to 32 corresponding to its version of training data. To be noticed, such process only took place on attributes with attribute level greater than 32 in its corresponding version of data - “site_id”, “site_domain”, “app_id”, “app_domain”, “app_category”, ”device_model”, “C14”, “C17”, “C19”, “C20”, and “C21”. In the 1m versions of training data, some attributes do not have an attribute level greater than 32 but do have new attribute categories introduced in 1m version of validation data and test data. We hence adjust the number of our frequent list correspondingly. For example, for “app_category”, only 27 categories were kept unchanged in the 1m training data, and 23 categories were kept unchanged in the 100k training data. 1m validation and test data were trimmed based on these two numbers respectively. A similar approach was conducted on the attribute “site_category” and “device_type”.



```
Log_Loss <- function(actual, prediction) {
  epsilon <- .000000000001
  yhat <- pmin(pmax(prediction, epsilon), 1-epsilon)
  logloss <- -mean(actual*log(yhat)
                   + (1-actual)*log(1 - yhat))
  return(logloss)
}
```

Figure 3: Log Loss Function

MODEL BUILDING

Our model building approach involved building various models on the training data sets and observe the best performance. We decided to implement naïve bayes, decision tree, logistic regression and random forest techniques for our model building approach. We then used the ensemble learning algorithm and concluded our model building with the best fit.

<u>Naïve Bayes</u>

Our first approach was to go ahead with the Naïve Bayes model building approach. This is a very simple classification approach and in a two -response classification the Bayes classifier will correspond to the predicting class (1 – click on an ad). As we had a classification problem and our outcome variable was a binary classification (there are only two categories) attribute, we decided to test our model for a benchmark result. The goal was to predict the correct class of click based on the x variables. To begin we decided to go ahead and apply the principles of Bayes rule on our first model. We trained our model on the whole training data set to begin with. The reason behind this was as Bayes classifier is based on a probability function, it will be computationally cheaper to run the model on 31 million rows.

To see if we were fitting the model correctly on the validation dataset or to see if our log-loss function was giving us the correct prediction value we downloaded the log loss function from MLmetrics library and used the LogLoss function to use as a reference. Our MLmetrics LogLoss came out to be 0.754 and the log-loss function mentioned in figure 3 gave us 0.7439. As the results showed the performance criterion was not well established, we decided to stick with our assumption for using Naïve Bayes model building as our benchmark and move ahead with other model building procedures. We decided to go ahead with decision tree model building approach as discussed in the next section.

<u>Decision Tree</u>

Decision tree or the classification trees are the most popular data mining tool. They are easy to interpret, implement and computationally cheap. As decision tree can be applied to both regression and classification models, we went ahead with this modelling approach to improve our performance criterion score.

To begin, we decided to set the control for building our tree and to do a recursive binary splitting on our training data, we went ahead and set a tree of maximum depth of 20. We left the complexity parameter at 0 as we wanted to keep every split in our initial model building phase and would prune the tree in the next steps. While building the decision tree we decided to choose the complete training data set. We pruned the decision tree model to have the least cross validation error. We obtained the optimal complexity parameter with the minimal cross validation error. After pruning the tree and then running the model on validation dataset we obtained a log loss of 0.418 and the MLmetrics LogLoss came out to be also 0.418. This was a very drastic improvement in our selection criterion. With the comparatively small log loss and the combination of decision trees being easily interpreted and we do not have to make dummy variables, we decided to move ahead to our next model building steps.

We also performed a computation on our 1 million dataset and validated our model for the purposes of ensemble procedure. We will touch on this topic in the sections ahead.

Lasso

The next approach we take is logistic regression with Lasso regulation. Considering the amount of variables we have in the training, validation and test set, we decided to use Lasso rather than Ridge for variable selection and avoid overfitting. By assigning different weights to each variable, the Lasso regression will move insignificant variables out of the model by assigning the weight as zero, hence reduce the complexity of the regression. We tried to build the model with optimal tuning parameter λ. If λ is large, the model will be too simple to capture all significant variables; if λ is small, the model will tend to overfit and have a bad generalization performance.

To improve the computing speed, we sample 1,000,000 instances from training set and 400,000 instances from the validation set.

To train the model and evaluate the result on the validation set, we first built a matrix for the x variables using model.matrix() function. As there are different levels for each categorical variable in train, validation and test set, building matrixes individually would have caused dimensionalionality issues. The solution was to combine the datasets together using rbind(), transform the whole dataset into a matrix, and then split it up back into train, validation, and test sets.

After solving the dimensional error, we dig deeper into how to find the optimal parameter. First, we incorporated function cv.glmnet() to run a cross validation that can help to select λ. The number of folders is set at the default value, 10. The output model has two values- lambda, which includes the values of lambda used in the fits, and lambda.min, which returns the value of lambda that give minimum mean cross-validation error. The lambda.min in our model is 0.0001022639. With λ = 0.0001022639, the model returns a log loss of 0.424.

Then we further explored λ by setting lambda=grid. In our case we set the grid as 10^seq(-7, -2, length=30). After plugging the grid into the glmnet() function, we made the prediction on the validation dataset and use apply() to calculate the log loss for every given lambda. To visualize the change of log loss given different lambda, we plot the line chart (figure 4) and find the best lambda that can minimize the log loss.

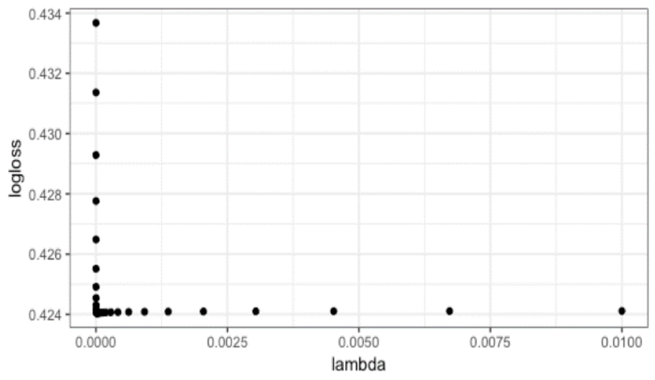


Figure 4: Lambda vs log loss to find minimized log loss

The plot is a U curve (although not visually obvious because of the scale of x and y) with the minimum point for the logloss in the lower left corner. After calling the min(loss), we saw the best performing lambda to be 1.743329e-05 with a log loss of 0.424.

Using cross validation and grid to try different lambda, we ended up our best model for the Lasso regression (figure 5):

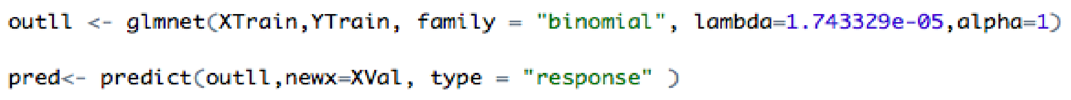
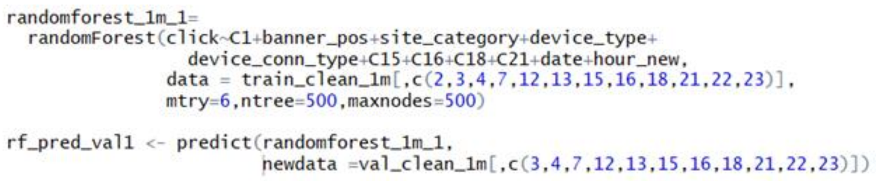

```
outll <- glmnet(XTrain,YTrain, family = "binomial", lambda=1.743329e-05,alpha=1)

pred<- predict(outll,newx=XVal, type = "response" )
```

Figure 5: Lasso regression final model

After choosing the model, we applied the model and make the prediction on the 1 million validation dataset so that we can compare different models and ensemble the models. We further make the predict on the whole test set to prepare for the next step.

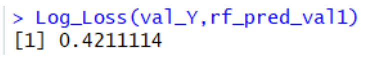
Random Forest

As we consider innumerous categorical variables in the dataset, random forest becomes our inevitable model of choice since it can deal with categorical variables by assigning observations based on different splitting criterions. Random forest is an ensemble learning methods, which is designed to use multiple learning algorithms to obtain better model performance than any used individual algorithm. Furthermore, it evolves from the decision tree algorithms, aimed to decrease the large variance of predictions. Precisely, each decision tree in random forest takes only a subset of features into considerations and has only access to a subset of training observations. For computational and cost saving purposes, we decided to use the 1m dataset for random forest. When approaching the model building process, we first inverted all variables in the dataset to factor level by using a simple "lapply" function. We then converted the "click" – Y variable into numeric level for further calculations of the log loss function. We decided to run a benchmark "randomforest_1m_1" at first by selecting several variables since random forest model takes a long time to build. Specifically, we assigned the ntree as 500 and maxnode as 500 in the model. Figure 6 displays our first random forest model with specified model parameters and its predictions on the validation dataset and figure 7 displays its result.



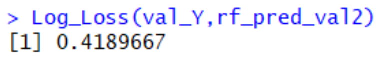
```
randomforest_1m_1=
  randomForest(click~C1+banner_pos+site_category+device_type+
                 device_conn_type+C15+C16+C18+C21+date+hour_new,
               data = train_clean_1m[,c(2,3,4,7,12,13,15,16,18,21,22,23)],
               mtry=6,ntree=500,maxnodes=500)

rf_pred_val1 <- predict(randomforest_1m_1,
                        newdata =val_clean_1m[,c(3,4,7,12,13,15,16,18,21,22,23)])
```

Figure 6: The Random Forest Benchmark model and Its Predictions on Validation



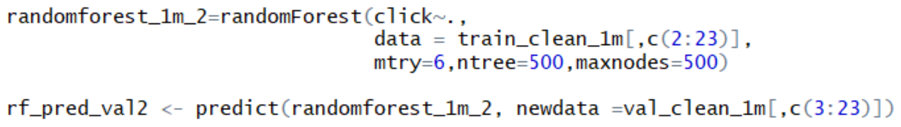
```
> Log_Loss(val_Y,rf_pred_val1)
[1] 0.4211114
```

Figure 7: The Log-Loss result of The Random Forest Benchmark model



```
> Log_Loss(val_Y,rf_pred_val2)
[1] 0.4189667
```

Figure 8: The Log-Loss result of The Random Forest Final model



```
randomforest_1m_2=randomForest(click~.,
                               data = train_clean_1m[,c(2:23)],
                               mtry=6,ntree=500,maxnodes=500)

rf_pred_val2 <- predict(randomforest_1m_2, newdata =val_clean_1m[,c(3:23)])
```

Figure 9: The Random Forest Final model and Its Predictions on Validation

After establishing our benchmark, we can finally build the model of all 21 variables. After 24 hours of running and building in R, we eventually got the random forest model with a log loss result of 0.419 (Figure 8). From the comparison of both results, we are able to conclude that feature selection in random forest would not cause a big difference on the prediction accuracy. Thus, we decided to keep the “randomforest_1m_2” (Figure 9) as our final random forest model.

The most difficulties we faced when building random forest models were dealing with factor levels. There were always cases that the factor level of a variable in the validation is bigger than the one in the training set; thus, it cannot be predicted as new levels has not been trained in the training set since the training set does not have that level at all. Another road block we faced was regarding to increase the log-loss accuracy. We later corrected this by changing Y variable into numeric level.

<u>Ensemble</u>

Ensemble learning is a powerful machine learning method, which has revealed to show many advantages in applications. We decided to add ensemble into our exploring journey as well, with the purpose of increasing model accuracy. The way we implement ensemble learning is to take the average of the predicted values from the three best-performed models, decision tree, lasso and random forest. We first checked the log loss result of the defined ensemble learning algorithm. Figure 10 displays the log-loss result of the ensemble learning is 0.4188207. From the result, we can see that it outperforms the other models and we decided to use ensemble for our final prediction on the test set.

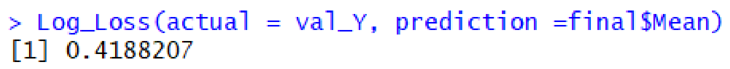

```
> Log_Loss(actual = val_Y, prediction =final$Mean)
[1] 0.4188207
```

Figure 10: The Random Forest Final model and Its Predictions on Validation

CONCLUSION

We divided the process of model building in a very methodical way where we followed the process of improving the log loss for each model built. Our main task involved analyzing 9 days of user advertisement click data and apply machine learning models to analyze predicting clicks for on-line advertisements. Our initial prognosis to this problem was to begin with a simplistic approach of naïve Bayes and decision/classification tree to set a baseline and see if we could improve on the model performance choosing techniques like ridge regression, the lasso technique, random forest and ensemble learning. As mentioned in the sections above, naïve Bayes did not perform well, and we moved to decision tree approach. This approach was taken on a full dataset and we saw a big improvement on the performance compared to Naïve Bayes.   We then decided to move the lasso technique for our next model building step and dropped the idea of doing a ridge regression as we had 23 attributes and 1 outcome variable. Lasso helps in doing a variable selection and this is what was needed for such a big dataset. We observed a log loss of 0.424 which was higher by 0.006 when compared to decision tree. We then chose random forest as we had multiple categorical variables and each time a split is made, it is based on a different splitting criterion. The random forest model gave us a log loss of 0.419. To produce an optimal predictive model, we chose the ensemble method giving a log loss of 0.418. The goal was to find a single model with a low log loss and we went ahead with the ensemble learning technique.

APPENDIX

Beginning next page is an attached copy of the code extracted from R studio.

```r
# Data_Cleaning ----------------------------------------------------------


#setwd("C:/Users/Meng Wang/Desktop/Machine Learning/Project/File")
library(data.table)
library(randomforest)


submit <- fread("https://s3.amazonaws.com/isom674ctr/ProjectSubmission-TeamX.csv")
test <- fread("https://s3.amazonaws.com/isom674ctr/ProjectTestData.csv")
train <- fread("https://s3.amazonaws.com/isom674ctr/ProjectTrainingData.csv")

#explore the data
head(train,20)
head(test,20)

#create new variable date and hour_new to represent weekdays and hour information
train$date <- substr(train$hour,5,6)
test$date <- substr(test$hour,5,6)
train$hour_new <- substr(train$hour,7,8)
test$hour_new <- substr(test$hour,7,8)

#recategorize weekdays for train
train[grep("21",train$date),25] <- "Tuesday"
train[grep("22",train$date),25] <- "Wednesday"
train[grep("23",train$date),25] <- "Thursday"
train[grep("24",train$date),25] <- "Friday"
train[grep("25",train$date),25] <- "Saturday"
train[grep("26",train$date),25] <- "Sunday"
train[grep("27",train$date),25] <- "Monday"
train[grep("28",train$date),25] <- "Tuesday"
train[grep("29",train$date),25] <- "Wednesday"


#recategorize weekdays for test
test[grep("21",test$date),24] <- "Tuesday"
test[grep("22",test$date),24] <- "Wednesday"
test[grep("23",test$date),24] <- "Thursday"
test[grep("24",test$date),24] <- "Friday"
test[grep("25",test$date),24] <- "Saturday"
test[grep("26",test$date),24] <- "Sunday"
test[grep("27",test$date),24] <- "Monday"
test[grep("28",test$date),24] <- "Tuesday"
test[grep("29",test$date),24] <- "Wednesday"
test[grep("30",test$date),24] <- "Thursday"
test[grep("31",test$date),24] <- "Friday"


unique(train$date)


head(train,n=10)
head(test,n=10)

#check the levels of each attribute

for (i in 1:26){
  a <- length(unique(train[[i]]))
  b <-  paste(names(train)[[i]],"    Attribute Level:",as.character(a),"    Percent:",round(100*a/nrow(train),2),"%")
  print(b)
}

for (i in 1:25){
  a <- length(unique(test[[i]]))
  b <-  paste(names(test)[[i]],"     Attribute Level:",as.character(a),"     Percent:",round(100*a/nrow(test),2),"%")
  print(b)
}


#explore attribute levels
sort(unique(train$C1))
sort(unique(test$C1))

sort(unique(train$banner_pos))
sort(unique(test$banner_pos))

sort(unique(train$device_type))
sort(unique(test$device_type))

sort(unique(train$device_conn_type))
sort(unique(test$device_conn_type))

sort(unique(train$C15))
sort(unique(test$C15))

sort(unique(train$C16))
sort(unique(test$C16))

sort(unique(train$C18))
sort(unique(test$C18))





#Comparing both sets of results, device_id and device_ip are too personalized/customized, hence should not be included in the prediction model


### Data Split - train,validation, test

train_new <- train[,c(1,2,4,5,6,7,8,9,10,11,14,15,16,17,18,19,20,21,22,23,24,25,26)]
test_clean <- test[,c(1,3,4,5,6,7,8,9,10,13,14,15,16,17,18,19,20,21,22,23,24,25)]

set.seed(511)
bound <- floor(nrow(train_new)*0.7)

train_new <- train_new[sample(nrow(train_new)), ]
train_clean <- train_new[1:bound, ]
val_clean <- train_new[(bound+1):nrow(train_new), ]


#transform data all to character
train_clean[] <- lapply(train_clean,as.character)
val_clean[] <- lapply(val_clean,as.character)
test_clean[] <- lapply(test_clean,as.character)

#create three versions of data, 1m, 100k and original (length)
train_clean_1m <- train_clean[sample(nrow(train_clean)),]
train_clean_1m <- train_clean_1m[1:1000000,]
train_clean_100k <- train_clean[sample(nrow(train_clean)),]
train_clean_100k <- train_clean_100k[1:100000,]
val_clean_1m <- val_clean[1:1000000,]
val_clean_100k <- val_clean

test_clean_1m <-test_clean
test_clean_100k <- test_clean

train_clean_1m[] <- lapply(train_clean_1m,as.character)
train_clean_100k[] <- lapply(train_clean_100k,as.character)


# for (i in 1:23){
#   a <- length(unique(train_clean[[i]]))
#   b <-  paste(names(train_clean)[[i]],"    Attribute Level:",as.character(a))
#   print(b)
# }
#
#
# for (i in 1:23){
#   a <- length(unique(train_clean_1m[[i]]))
#   b <-  paste(names(train_clean_1m)[[i]],"    Attribute Level:",as.character(a))
#   print(b)
# }
#
# for (i in 1:23){
#   a <- length(unique(train_clean_100k[[i]]))
#   b <-  paste(names(train_clean_100k)[[i]],"    Attribute Level:",as.character(a))
#   print(b)
# }









#transform variables with more than 31 varaibles to 31 top variables + 1 "General" category for Decision Tree, Random Forest and Lasso


### site_id

#Train: site_id
a1 <- unique(sort(train_clean$site_id,decreasing = TRUE))[1:31]
a2 <- unique(sort(train_clean_1m$site_id,decreasing = TRUE))[1:31]
a3 <- unique(sort(train_clean_100k$site_id,decreasing = TRUE))[1:31]
train_clean[!(train_clean$site_id %in% a1),5] <- "General"
train_clean_1m[!(train_clean_1m$site_id %in% a2),5] <- "General"
train_clean_100k[!(train_clean_100k$site_id %in% a3),5] <- "General"
```

```r
#Val: site_id:
val_clean[!(val_clean$site_id %in% a1),5] <- "General"
val_clean_1m[!(val_clean_1m$site_id %in% a2),5] <- "General"
val_clean_100k[!(val_clean_100k$site_id %in% a3),5] <- "General"
#Test: site_id:
test_clean[!(test_clean$site_id %in% a1),4] <- "General"
test_clean_1m[!(test_clean_1m$site_id %in% a2),4] <- "General"
test_clean_100k[!(test_clean_100k$site_id %in% a3),4] <- "General"


### site_domain

#Train: site_domain
b1 <- unique(sort(train_clean$site_domain,decreasing = TRUE))[1:31]
b2 <- unique(sort(train_clean_1m$site_domain,decreasing = TRUE))[1:31]
b3 <- unique(sort(train_clean_100k$site_domain,decreasing = TRUE))[1:31]
train_clean[!(train_clean$site_domain %in% b1),6] <- "General"
train_clean_1m[!(train_clean_1m$site_domain %in% b2),6] <- "General"
train_clean_100k[!(train_clean_100k$site_domain %in% b3),6] <- "General"
#Val: site_domain:
val_clean[!(val_clean$site_domain %in% b1),6] <- "General"
val_clean_1m[!(val_clean_1m$site_domain %in% b2),6] <- "General"
val_clean_100k[!(val_clean_100k$site_domain %in% b3),6] <- "General"
#Test: site_domain:
test_clean[!(test_clean$site_domain %in% b1),5] <- "General"
test_clean_1m[!(test_clean_1m$site_domain %in% b2),5] <- "General"
test_clean_100k[!(test_clean_100k$site_domain %in% b3),5] <- "General"


### app_id

#Train: app_id
c1 <- unique(sort(train_clean$app_id,decreasing = TRUE))[1:31]
c2 <- unique(sort(train_clean_1m$app_id,decreasing = TRUE))[1:31]
c3 <- unique(sort(train_clean_100k$app_id,decreasing = TRUE))[1:31]
train_clean[!(train_clean$app_id %in% c1),8] <- "General"
#train_clean_1m[!(train_clean_1m$app_id %in% c2),8] <- "General"
train_clean_100k[!(train_clean_100k$app_id %in% c3),8] <- "General"
#Val: app_id
val_clean[!(val_clean$app_id %in% c1),8] <- "General"
#val_clean_1m[!(val_clean_1m$app_id %in% c2),8] <- "General"
val_clean_100k[!(val_clean_100k$app_id %in% c3),8] <- "General"
#Test: app_id
test_clean[!(test_clean$app_id %in% c1),7] <- "General"
#test_clean_1m[!(test_clean_1m$app_id %in% c2),7] <- "General"
test_clean_100k[!(test_clean_100k$app_id %in% c3),7] <- "General"


### app_domain

#Train: app_domain
d1 <- unique(sort(train_clean$app_domain,decreasing = TRUE))[1:31]
d2 <- unique(sort(train_clean_1m$app_domain,decreasing = TRUE))[1:31]
d3 <- unique(sort(train_clean_100k$app_domain,decreasing = TRUE))[1:31]
train_clean[!(train_clean$app_domain %in% d1),9] <- "General"
train_clean_1m[!(train_clean_1m$app_domain %in% d2),9] <- "General"
train_clean_100k[!(train_clean_100k$app_domain %in% d3),9] <- "General"
#Val: app_domain
val_clean[!(val_clean$app_domain %in% d1),9] <- "General"
val_clean_1m[!(val_clean_1m$app_domain %in% d2),9] <- "General"
val_clean_100k[!(val_clean_100k$app_domain %in% d3),9] <- "General"
#Test: app_domain:
test_clean[!(test_clean$app_domain %in% d1),8] <- "General"
test_clean_1m[!(test_clean_1m$app_domain %in% d2),8] <- "General"
test_clean_100k[!(test_clean_100k$app_domain %in% d3),8] <- "General"


### app_category

#Train: app_category
e1 <- unique(sort(train_clean$app_category,decreasing = TRUE))[1:31]
e2 <- unique(sort(train_clean_1m$app_category,decreasing = TRUE))[1:27]
e3 <- unique(sort(train_clean_100k$app_category,decreasing = TRUE))[1:23]
train_clean[!(train_clean$app_category %in% e1),10] <- "General"
train_clean_1m[!(train_clean_1m$app_category %in% e2),10] <- "General"
train_clean_100k[!(train_clean_100k$app_category %in% e3),10] <- "General"
#Val: app_category
val_clean[!(val_clean$app_category %in% e1),10] <- "General"
val_clean_1m[!(val_clean_1m$app_category %in% e2),10] <- "General"
val_clean_100k[!(val_clean_100k$app_category %in% e3),10] <- "General"
#Test: app_category
test_clean[!(test_clean$app_category %in% e1),9] <- "General"
test_clean_1m[!(test_clean_1m$app_category %in% e2),9] <- "General"
test_clean_100k[!(test_clean_100k$app_category %in% e3),9] <- "General"


### device_model

#Train: device_model
f1 <- unique(sort(train_clean$device_model,decreasing = TRUE))[1:31]
f2 <- unique(sort(train_clean_1m$device_model,decreasing = TRUE))[1:31]
f3 <- unique(sort(train_clean_100k$device_model,decreasing = TRUE))[1:31]
train_clean[!(train_clean$device_model %in% f1),11] <- "General"
train_clean_1m[!(train_clean_1m$device_model %in% f2),11] <- "General"
train_clean_100k[!(train_clean_100k$device_model %in% f3),11] <- "General"
#Val: device_model
val_clean[!(val_clean$device_model %in% f1),11] <- "General"
val_clean_1m[!(val_clean_1m$device_model %in% f2),11] <- "General"
val_clean_100k[!(val_clean_100k$device_model %in% f3),11] <- "General"
#Test: device_model
test_clean[!(test_clean$device_model %in% f1),10] <- "General"
test_clean_1m[!(test_clean_1m$device_model %in% f2),10] <- "General"
test_clean_100k[!(test_clean_100k$device_model %in% f3),10] <- "General"


### C14

#Train: C14
g1 <- unique(sort(train_clean$C14,decreasing = TRUE))[1:31]
g2 <- unique(sort(train_clean_1m$C14,decreasing = TRUE))[1:31]
g3 <- unique(sort(train_clean_100k$C14,decreasing = TRUE))[1:31]
train_clean[!(train_clean$C14 %in% g1),14] <- "General"
train_clean_1m[!(train_clean_1m$C14 %in% g2),14] <- "General"
train_clean_100k[!(train_clean_100k$C14 %in% g3),14] <- "General"
#Val: C14
val_clean[!(val_clean$C14 %in% g1),14] <- "General"
val_clean_1m[!(val_clean_1m$C14 %in% g2),14] <- "General"
val_clean_100k[!(val_clean_100k$C14 %in% g3),14] <- "General"
#Test: C14
test_clean[!(test_clean$C14 %in% g1),13] <- "General"
test_clean_1m[!(test_clean_1m$C14 %in% g2),13] <- "General"
test_clean_100k[!(test_clean_100k$C14 %in% g3),13] <- "General"


##3 C17

#Train: C17
h1 <- unique(sort(train_clean$C17,decreasing = TRUE))[1:31]
h2 <- unique(sort(train_clean_1m$C17,decreasing = TRUE))[1:31]
h3 <- unique(sort(train_clean_100k$C17,decreasing = TRUE))[1:31]
train_clean[!(train_clean$C17 %in% h1),17] <- "General"
train_clean_1m[!(train_clean_1m$C17 %in% h2),17] <- "General"
train_clean_100k[!(train_clean_100k$C17 %in% h3),17] <- "General"
#Val: C17
val_clean[!(val_clean$C17 %in% h1),17] <- "General"
val_clean_1m[!(val_clean_1m$C17 %in% h2),17] <- "General"
val_clean_100k[!(val_clean_100k$C17 %in% h3),17] <- "General"
#Test: C17
test_clean[!(test_clean$C17 %in% h1),16] <- "General"
test_clean_1m[!(test_clean_1m$C17 %in% h2),16] <- "General"
test_clean_100k[!(test_clean_100k$C17 %in% h3),16] <- "General"


### C19

colnames(train_clean)
colnames(test_clean)
#Train: C19
i1 <- unique(sort(train_clean$C19,decreasing = TRUE))[1:31]
i2 <- unique(sort(train_clean_1m$C19,decreasing = TRUE))[1:31]
i3 <- unique(sort(train_clean_100k$C19,decreasing = TRUE))[1:31]
train_clean[!(train_clean$C19 %in% i1),19] <- "General"
train_clean_1m[!(train_clean_1m$C19 %in% i2),19] <- "General"
train_clean_100k[!(train_clean_100k$C19 %in% i3),19] <- "General"
#Val: C19
val_clean[!(val_clean$C19 %in% i1),19] <- "General"
val_clean_1m[!(val_clean_1m$C19 %in% i2),19] <- "General"
val_clean_100k[!(val_clean_100k$C19 %in% i3),19] <- "General"
#Test: C19
test_clean[!(test_clean$C19 %in% i1),18] <- "General"
test_clean_1m[!(test_clean_1m$C19 %in% i2),18] <- "General"
test_clean_100k[!(test_clean_100k$C19 %in% i3),18] <- "General"


### C20

#Train: C20
j1 <- unique(sort(train_clean$C20,decreasing = TRUE))[1:31]
j2 <- unique(sort(train_clean_1m$C20,decreasing = TRUE))[1:31]
j3 <- unique(sort(train_clean_100k$C20,decreasing = TRUE))[1:31]
train_clean[!(train_clean$C20 %in% j1),20] <- "General"
```

```r
train_clean_1m[!(train_clean_1m$C20 %in% j2),20] <- "General"
train_clean_100k[!(train_clean_100k$C20 %in% j3),20] <- "General"
#Val: C20
val_clean[!(val_clean$C20 %in% j1),20] <- "General"
val_clean_1m[!(val_clean_1m$C20 %in% j2),20] <- "General"
val_clean_100k[!(val_clean_100k$C20 %in% j3),20] <- "General"
#Test: C20
test_clean[!(test_clean$C20 %in% j1),19] <- "General"
test_clean_1m[!(test_clean_1m$C20 %in% j2),19] <- "General"
test_clean_100k[!(test_clean_100k$C20 %in% j3),19] <- "General"


### C21

#Train: C21
k1 <- unique(sort(train_clean$C21,decreasing = TRUE))[1:31]
k2 <- unique(sort(train_clean_1m$C21,decreasing = TRUE))[1:31]
k3 <- unique(sort(train_clean_100k$C21,decreasing = TRUE))[1:31]
train_clean[!(train_clean$C21 %in% k1),21] <- "General"
train_clean_1m[!(train_clean_1m$C21 %in% k2),21] <- "General"
train_clean_100k[!(train_clean_100k$C21 %in% k3),21] <- "General"
#Val: C21
val_clean[!(val_clean$C21 %in% k1),21] <- "General"
val_clean_1m[!(val_clean_1m$C21 %in% k2),21] <- "General"
val_clean_100k[!(val_clean_100k$C21 %in% k3),21] <- "General"
#Test: C21
test_clean[!(test_clean$C21 %in% k1),20] <- "General"
test_clean_1m[!(test_clean_1m$C21 %in% k2),20] <- "General"
test_clean_100k[!(test_clean_100k$C21 %in% k3),20] <- "General"


### site_category

#Train: site_category
l1 <- unique(sort(train_clean$site_category,decreasing = TRUE))[1:25]
l2 <- unique(sort(train_clean_1m$site_category,decreasing = TRUE))[1:20]
l3 <- unique(sort(train_clean_100k$site_category,decreasing = TRUE))[1:19]
train_clean[!(train_clean$site_category %in% l1),7] <- "General"
train_clean_1m[!(train_clean_1m$site_category %in% l2),7] <- "General"
train_clean_100k[!(train_clean_100k$site_category %in% l3),7] <- "General"
#Val: site_category:
val_clean[!(val_clean$site_category %in% l1),7] <- "General"
val_clean_1m[!(val_clean_1m$site_category %in% l2),7] <- "General"
val_clean_100k[!(val_clean_100k$site_category %in% l3),7] <- "General"
#Test: site_category:
test_clean[!(test_clean$site_category %in% l1),6] <- "General"
test_clean_1m[!(test_clean_1m$site_category %in% l2),6] <- "General"
test_clean_100k[!(test_clean_100k$site_category %in% l3),6] <- "General"


### device_type
m3 <- unique(sort(train_clean_100k$device_type,decreasing = TRUE))[1:3]
train_clean_100k[!(train_clean_100k$device_type %in% m3),12] <- "General"
val_clean_100k[!(val_clean_100k$device_type %in% m3),12] <- "General"
test_clean_100k[!(test_clean_100k$device_type %in% m3),11] <- "General"


### FinalPrepare - change data to factor
train_clean_100k[] <- lapply(train_clean_100k,as.factor)
val_clean_100k[] <- lapply(val_clean_100k,as.factor)
train_clean_100k <- train_clean_100k[complete.cases(train_clean_100k),]
val_clean_100k <- val_clean_100k[complete.cases(val_clean_100k),]
sapply(train_clean_100k, class)
sapply(val_clean_100k, class)
val_clean_100k <- rbind(train_clean_100k[1, ] , val_clean_100k)
val_clean_100k <- val_clean_100k[-1,]

#fwrite(train_clean_100k, file = "train_100k", row.names = FALSE, col.names = TRUE)
#fwrite(val_clean_100k, file = "val_100k", row.names = FALSE, col.names = TRUE)
#fwrite(test_clean_100k, file = "test_100k", row.names = FALSE, col.names = TRUE)


### Log Loss

Log_Loss <- function(actual, prediction) {
  epsilon <- .000000000001
  yhat <- pmin(pmax(prediction, epsilon), 1-epsilon)
  logloss <- -mean(actual*log(yhat)
                   + (1-actual)*log(1 - yhat))
  return(logloss)
}


fwrite(train_clean,"train_clean.csv")
fwrite(train_clean_1m,"train_clean_1m.csv")
fwrite(train_clean_100k,"train_clean_100k.csv")

fwrite(test_clean,"test_clean.csv")
fwrite(test_clean_1m,"test_clean_1m.csv")
fwrite(test_clean_100k,"test_clean_100k.csv")

fwrite(val_clean,"val_clean.csv")
fwrite(val_clean_1m,"val_clean_1m.csv")
fwrite(val_clean_100k,"val_clean_100k.csv")








# Naive Bayes ------------------------------------------------------------



if (!require('naivebayes')) {install.packages('naivebayes'); require('naivebayes')}

train_clean <- subset(train_new, select = c(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23))

val_clean <- subset(val_new, select = c(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23))

naivebayes <-naive_bayes(x=train_clean[,c(2:22)],y=train_clean$click)

naivebayesprediction <-predict(naivebayes,newdata = val_clean,type = 'prob')

predict_validation_naivebayes <- naivebayesprediction[,2]

val_Y <- as.numeric(val_clean$click)

Log_Loss(val_Y,predict_validation_naivebayes)

# We did a correlation to see if our results are working.
cor(val_Y,predict_validation_naivebayes)

install.packages("MLmetrics")

library(MLmetrics)

# This was used to verify all our results and see if Log loss formula we constreucted is correct or not.
LogLoss(predict_validation_naivebayes,val_Y)

# Decision Tree ------------------------------------------------------------

install.packages("rpart")

library(rpart)

train_clean$click <- as.factor(train_clean$click)

# CP was chosen 0 as were prunning later
decisiontreecontrol <- rpart.control(minsplit = 20, maxdepth = 20, cp = 0)

decisiontree <- rpart(click ~ ., data = train_clean, control = decisiontreecontrol)

# optimal complexity parameter with the minimal cross validation error
decisiontreeCP <- decisiontree$cptable[which.min(decisiontree$cptable[, "xerror"]), "CP"]

# prunning the decision tree model to have the least cross validation error
decisiontreePrune <- prune(decisiontree, cp = decisiontreeCP)

predict_validation_decisiontree <- predict(decisiontreePrune, newdata = val_clean, type = "prob")
predict_validation_decisiontree <- predict_validation_decisiontree[,2]
LogLoss(predict_validation_decisiontree,val_Y)
Log_Loss(val_Y,predict_validation_decisiontree)

# Line 500 to lilne 511 were coded to be used later in the ensembling process.

temp <- as.numeric(as.character(val_clean$click))
temp <- temp[1:1000000]
identical(as.integer(temp),val_clean_1m$click)

val_clean_1M <- val_clean[1:1000000,]
predict_validation_decisiontree_temp <- predict(decisiontreePrune, newdata = val_clean_1M, type = "prob")
predict_validation_decisiontree_1M <- predict_validation_decisiontree_temp[,2]

val_Y_1M <- as.numeric(val_clean_1M$click)

LogLoss(predict_validation_decisiontree_1M,val_Y_1M)
Log_Loss(val_Y_1M,predict_validation_decisiontree_1M)
```

```r
# LASSO -------------------------------------------------------------------

install.packages("glmnet")
library(glmnet)

#factor all variables
train_clean_1m[] <- lapply(train_clean_1m,as.factor)
val_clean_1m[] <- lapply(val_clean_1m,as.factor)
test_clean_1m[] <- lapply(test_clean_1m,as.factor)

train_clean[]<-lapply(train_clean,as.factor)
val_clean[]<-lapply(val_clean,as.factor)
test_clean[]<- lapply(test_clean,as.factor)

#Train model and choose parameter lambda

#choose 400000 instances from validation
val_clean_40<- val_clean_1m[c(1:400000),]

#prepare the XTrain, XVal, YTrian, and YVal for building the model
nTrain <- nrow(train_clean_1m)
nVal <- nrow(val_clean_40)
AllData <- rbind(train_clean_1m,val_clean_40)

XAll <- model.matrix(click ~ .,AllData[,-1])
XTrain <- XAll[1:nTrain,][,-1]
XVal <- XAll[-(1:nTrain),][,-1]

YTrain <- train_clean_1m$click
YVal <- val_clean_40$click

#try to find the optimal lambda using 10-fold cross validation (cv.glmnet)
cv.glmmod <- cv.glmnet(XTrain,YTrain, family = "binomial",alpha=1)
plot(cv.glmmod)

#find the lambda with lowest mean error
lam<- cv.glmmod$lambda.min
#cv.glmmod$lambda.min= 0.0001022639

##build the model
out <- glmnet(XTrain,YTrain, family = "binomial", lambda=lam,alpha=1)
pred<- predict(out,newx=XVal, type = "response" )

y<- as.numeric(as.character(val_clean_40$click))

##calculate log loss
loss<-Log_Loss1(y,pred)

#try to find the optimal lambda using a sequense of numbers
grid <- 10^seq(-7,-2,length=30)

##build the model
out2 <- glmnet(XTrain,YTrain, family = "binomial", lambda=grid,alpha=1)
pred2 <- predict(out2,newx=XVal, type = "response" )

##calculate log loss for each lambda and compare
fn <- function(prediction, actual) {
  epsilon <- .000000000001
  yhat <- pmin(pmax(prediction, epsilon), 1-epsilon)
  logloss <- -mean(actual*log(yhat)
                   + (1-actual)*log(1 - yhat))
  return(logloss)
}

loss2<-apply(pred2,2,FUN=Log_Loss,y)

#plot lambda versus logloss
table<- do.call(rbind.data.frame, Map('c', grid, loss2))

ggplot(data=table, aes(x=lambda, y=logloss, group=1)) +
  geom_point()+
  theme_bw()

min(loss2)

#Final Lasso model after comparing:
outll <- glmnet(XTrain,YTrain, family = "binomial", lambda=1.743329e-05,alpha=1)

### pred on 1m val

#build the matrix
nTrain <- nrow(train_clean_1m)
nVal <- nrow(val_clean_1m)
AllData <- rbind(train_clean_1m,val_clean_1m)

XAll <- model.matrix(click ~ .,AllData[,-1])
XTrain <- XAll[1:nTrain,][,-1]
XVal <- XAll[-(1:nTrain),][,-1]

YTrain <- train_clean_1m$click
YVal <- val_clean_1m$click

#fit the model and make prediction
outll <- glmnet(XTrain,YTrain, family = "binomial", lambda=1.743329e-05,alpha=1)
predict_validation_lasso<- predict(outll,newx=XVal, type = "response" )

#write out the data
predict_validation_lasso<-as.data.frame(predict_validation_lasso)
fwrite(predict_validation_lasso,"pred_val.csv")






# Random_Forest -----------------------------------------------------------


install.packages("randomForest")
library(randomForest)

# convert all variables as a factor level for both train and validation sets.
train_clean_1m[] <- lapply(train_clean_1m,as.factor)
val_clean_1m[] <- lapply(val_clean_1m,as.factor)

# convert the y variable as numeric level for both train and validation sets.
train_clean_1m$click <- as.numeric(as.character(train_clean_1m$click))
Val_Y_1m <- as.numeric(as.character(val_clean_1m$click))

## Run the random forest model
# the first model we trained with variable selection
randomforest_1m_1=randomForest(click~C1+banner_pos+site_category+device_type+device_conn_type+C15+C16+C18+C21+date+hour_new,
                               data = train_clean_1m[,c(2,3,4,7,12,13,15,16,18,21,22,23)],mtry=6,ntree=500,maxnodes=500)
# second model we trained with all the x variables
randomforest_1m_2=randomForest(click~., data = train_clean_1m[,c(2:23)],mtry=6,ntree=500,maxnodes=500)

# predict both models on validation set.
predict_validation_randomforest1 <- predict(randomforest_1m_1, newdata =val_clean_1m[,c(3,4,7,12,13,15,16,18,21,22,23)])
predict_validation_randomforest2 <- predict(randomforest_1m_2, newdata =val_clean_1m[,c(3:23)])


# run two log loss function based on the prediction adn actual values on validations's click.
Log_Loss(Val_Y_1m,predict_validation_randomforest2)
LogLoss(predict_validation_randomforest2,Val_Y_1m)



# Ensemble ----------------------------------------------------------------

# combine all three predictions on validation set together, of three models (Decision tree, LASSO, Random forest)
final <- data.frame(las = predict_validation_lasso,
                    dt = predict_validation_decisiontree_1M,
                    rf = predict_validation_randomforest2)

# take the mean for each row (observation)
final$Mean <- rowMeans(final[,1:3])

# calculate the log loss function
Log_Loss(actual = Val_Y_1m, prediction =final$Mean)
LogLoss(final$Mean, Val_Y_1m)


# Predict test with chosen model ------------------------------------------

### Lasso

#build tge matrix
nTrain <- nrow(train_clean_1m)
nVal <- nrow(val_clean_40)
nTest<- nrow(test_clean)

train_X<- train_clean_1m[,-2]
val_X<- val_clean_40[,-2]
All <- rbind(train_X,val_X,test_clean)

XAll_t <- model.matrix(~ .,All[,-1])
XTrain_t <- XAll_t[1:nTrain,][,-1]
XRemain_t <- XAll_t[-(1:nTrain),][,-1]
XVal_t <- XRemain_t[1:nVal,]
XTest_t <- XRemain_t[-(1:nVal),]
```

```r
YTrain <- train_clean_1m$click
YVal <- val_clean_40$click

#fit in the model
outl1_t <- glmnet(XTrain_t,YTrain, family = "binomial", lambda=1.743329e-0,alpha=1)
#pred_t <- predict(outl1_t,newx=XVal_t, type = "response" )
#Log_Loss(y,pred_t)

#split the test because it is too large
XTest_1<- XTest_t[1:1000000,]
XTest_2<- XTest_t[1000001:3000000,]
XTest_3<- XTest_t[3000001:5000000,]
XTest_4<- XTest_t[5000001:7000000,]
XTest_5<- XTest_t[7000001:9000000,]
XTest_6<- XTest_t[9000001:11000000,]
XTest_7<- XTest_t[11000001:13015341,]

#make predictions on the test
pred_1<-predict(outl1_t,newx=XTest_1, type = "response" )
pred_1<- as.data.frame(pred_1)
#fwrite(pred_1, "pred_1.csv")

pred_2<-predict(outl1_t,newx=XTest_2, type = "response" )
pred_2<- as.data.frame(pred_2)
#fwrite(pred_2, "pred_2.csv")

pred_3<-predict(outl1_t,newx=XTest_3, type = "response" )
pred_3<- as.data.frame(pred_3)
#fwrite(pred_3, "pred_3.csv")

pred_4<-predict(outl1_t,newx=XTest_4, type = "response" )
pred_4<- as.data.frame(pred_4)
#fwrite(pred_4, "pred_4.csv")

pred_5<-predict(outl1_t,newx=XTest_5, type = "response" )
pred_5<- as.data.frame(pred_5)
#fwrite(pred_5, "pred_5.csv")

pred_6<-predict(outl1_t,newx=XTest_6, type = "response" )
pred_6<- as.data.frame(pred_6)
#fwrite(pred_6, "pred_6.csv")

pred_7<-predict(outl1_t,newx=XTest_7, type = "response" )
pred_7<- as.data.frame(pred_7)
#fwrite(pred_7, "pred_7.csv")

predict_test_lasso<- rbind(pred_1, pred_2, pred_3, pred_4, pred_5, pred_6, pred_7)

#fwrite(predict_test_lasso, "predict_test_lasso1.csv")




### Decision Tree

predict_test_decisiontree <- predict(decisiontreePrune, newdata = test_clean_1m, type = "prob")
predict_test_decisiontree <- predict_test_decisiontree[,2]


### Random Forest

# convert all variables as a factor level for test set.
test_clean_1m[] <- lapply(test_clean_1m,as.factor)

## since the test set is too big to run in a off-line laptop, we decided to split it into 4,
##then run the prediction separtely, then rbind together.
# predict the test set with the random forest model "randomforest_1m_2"
test_clean_1m1 <- rbind(train_clean_1m[1,c(1,3:23)] , test_clean_1m[1:4000000,])
test_clean_1m1 <- test_clean_1m1[-1,]
rf_pred_test2_1 <- predict(randomforest_1m_2, newdata =test_clean_1m1[,c(2:22)])
rf_pred_test2_1 <- data.frame(rf_pred_test2_1)
names(rf_pred_test2_1) <- "pred_test"

test_clean_1m2 <- rbind(train_clean_1m[1,c(1,3:23)] , test_clean_1m[4000001:8000000,])
test_clean_1m2 <- test_clean_1m2[-1,]
rf_pred_test2_2 <- predict(randomforest_1m_2, newdata =test_clean_1m2[,c(2:22)])
rf_pred_test2_2 <- data.frame(rf_pred_test2_2)
names(rf_pred_test2_2) <- "pred_test"

test_clean_1m3 <- rbind(train_clean_1m[1,c(1,3:23)] , test_clean_1m[8000001:12000000,])
test_clean_1m3 <- test_clean_1m3[-1,]
rf_pred_test2_3 <- predict(randomforest_1m_2, newdata =test_clean_1m3[,c(2:22)])
rf_pred_test2_3 <- data.frame(rf_pred_test2_3)
names(rf_pred_test2_3) <- "pred_test"

test_clean_1m4 <- rbind(train_clean_1m[1,c(1,3:23)] , test_clean_1m[12000001:13015341,])
test_clean_1m4 <- test_clean_1m4[-1,]
rf_pred_test2_4 <- predict(randomforest_1m_2, newdata =test_clean_1m4[,c(2:22)])
rf_pred_test2_4 <- data.frame(rf_pred_test2_4)
names(rf_pred_test2_4) <- "pred_test"

# then rbind together and names final prediction with "rf_pred_test2"
rf_pred_test2 <- rbind(rf_pred_test2_1,rf_pred_test2_2,rf_pred_test2_3,rf_pred_test2_4)
dim(rf_pred_test2)

#write.csv(rf_pred_test2,file="pred_test.csv")


# Final Submission --------------------------------------------------------

## After confirming the final model: ensemble learning algorithm, which average predictions of all three models: DT, RF, LASSO.

# combine all three different predictions together and take their average as our final submission click probability.
outcome <- data.frame(test_clean_1m$id,predict_test_lasso,predict_test_decisiontree, rf_pred_test2)
names(outcome) <- c("id","lasso","dt","rf")
outcome$Mean <- rowMeans(outcome[,2:4])
head(outcome)

# check 10 random selected lines to see any unusual output.
outcome[c(1,3791,72649,12345,123456,432875,987,3458,90,13000000,2098367),]

# assign our final submission click probability to submission dataset.
submit$`P(click)`<- outcome$Mean

# finally fwrite out our csv submission file.
fwrite(submit,"ProjectSubmission-Team9.csv")
```