ProjectDescription-Team5

Team 5: Lanxin (Lancy) Mao, Xi (Athena) Li, Elena Lopez

1. Data processing:

➢ Select features:

Basic Idea:

There are three dimensions we need to consider about feature selection:

- If the categories of one feature almost uniquely identify every observation, i.e. there are too many unique categories, this feature maybe useless for prediction;
- Some features can be transformed to represent more useful information;
- There are some new categories appeared in Test dataset, which should be recoded.

Methodology:

- Select “hour” from time: because we assumed hour in each day affects the click rate, so we included “hour” as a predictor.
- “Id” was the identifier of each observation, so we didn’t use “id’ as predictor.
- We calculated the number and percentage of new categories in Test data.

Result:

From the result, we can see that device_ip and device_id had over 50% new categories in Test data, so we didn’t use them as predictors.

Feature name	***# New category***	***New category %***
C1	0	0
banner_pos	0	0
site_id	261	6.6
site_domain	571	10.8
site_category	0	0
app_id	1048	15.8
app_domain	54	13.7

app_category	0	0
device_id	599808	58.8
device_ip	1575730	49
device_model	245	3.5
device_type	0	0
device_conn_type	0	0
C14	420	15.1
C15	0	0
C16	0	0
C17	67	14.2
C18	0	0
C19	3	4.4
C20	1	0.6
C21	7	11.3
Hour	0	0

➢ Group small categories as others:

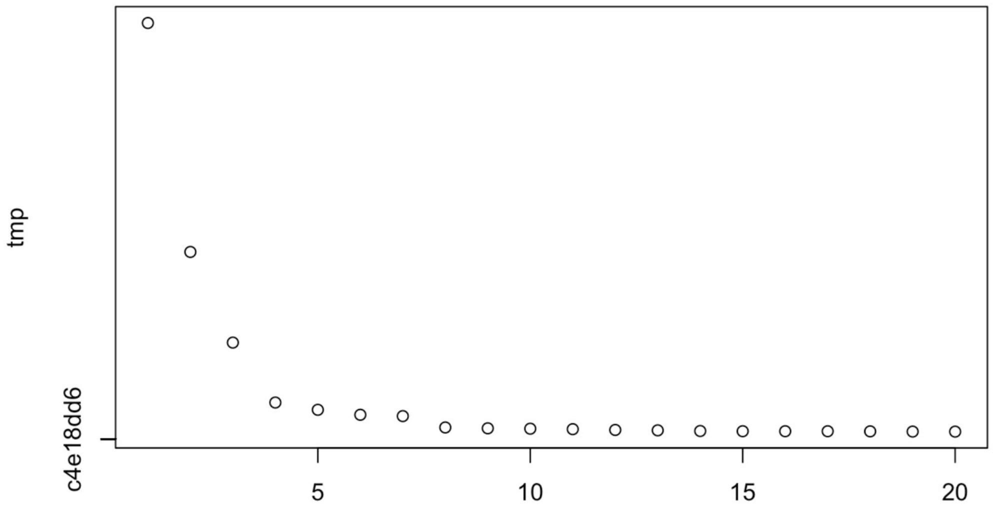
Basic Idea: The dimensions for some features were too large to compute. We calculated category distribution of each feature in Training data, and group categories that account for less than 20% of observations as ‘others’.

Methodology: We looked into top 20 categories of each feature and plotted their frequency. We selected top N categories for each feature. Since each N depend on the distribution of each feature , the value varied across features. Finally we grouped all categories with low occurrence (not within these top N) as ‘others’.

Example: site category
Frequency dropped to 2700 from 12000 since top 14, so we grouped all categories after top 13 as ‘others’. So there were 14 categories in total including ‘others’.

```
[1] "site_category"
number of category =  26

50e219e0 f028772b 28905ebd 3e814130 f66779e6 75fa27f6 335d28a8 76b2941d c0dd3be3 72722551 dedf689d 0569f928 70fb0e29
12731910 10210332  5952006  2442362   212135   135240   107151    89924    33689    21084    19331    14137    12309
a818d37a 8fd0aea4 42a36e14 e787de0e bcf865d9 5378d028 9ccfa2ea
    2674     2082     2039     1080      909      372      279
```



Result:

Feature name	*Orig category number*	*Revised category number*
C1	7	6
banner_pos	7	5
site_id	4581	16
site_domain	7341	16
site_category	26	14
app_id	8088	16
app_domain	526	16
app_category	36	14
device_model	8058	16
device_type	5	5
device_conn_type	4	4
C14	2465	16

C15	8	8
C16	9	6
C17	407	16
C18	4	4
C19	66	16
C20	171	16
C21	55	16
Hour	24	24

➢ Categorize new category as “others”:

There are 12 features containing new categories in Test data, including site_id, dite_domain, app_id, app_domain, device_id, device_ip, device_model, C14, C17,C19,C20,C21. Here, we categorize these new categories as “others” using the same code in the 2nd step.

2. Data sampling:

➢ The raw dataset was so large, so we had to use sample data to train and test model. So we split Training data into small files with each of which contains 1 million observations.

➢ There is no validation dataset, so we split the raw Train data into train and validation dataset. So we used one sample we split above as training and one as validation data to run the models.

➢ Also, we used the same command lines to split test data to speed prediction.

```
$ split -l 1000000 -d Train_v2.csv Train_v
```

3. Data Modeling:

➢ Lasso regression

- **Initiative:**

Since lasso has built-in feature selection in the model, we may have it as a reference for later models’ feature engineering.

- **Steps:**

1. Created a grid of lambdas in order to find the best complexity of the model. Train the lasso model using the training data based on the grid of lambdas.
2. Fit the model on validation data to get predictions for each corresponding lamba.
3. Compared the predictions with the actual results of validation and get a log loss for each lasso model. Draw a plot with lambdas against log loss results and try to find the lambda who has the lowest log loss. The plot should display a valley-like pattern. If the plot is solely going upward or solely going downward, it indicates that we need to have a grid with smaller lambda or larger lambdas respectively.
4. After iterating the above process several times, we got our best lasso logistic regression model.
5. Take a look at

- **Result:**

In our best model, lasso automatically make the coefficient of some features as 0. Feature such as device_type5 appeared to have little contribution in predicting the clicks. The best model, eventually, has a lambda of 2.45 e -6 and log loss of 0.415 based on the sample validation data.

➢ KNN

- **Initiative:**

K-nearest Neighbor Classification classify each object based on the majority vote of its K neighbors. In this case, similar scenario tend to have similar action (click or unclick).

- **Steps:**

We run this KNN classifier on one of our sample training and validation dataset.

1. specified the range for k, and k should not be so small because that will tend to cause overfitting.
2. Run a loop to find the best k based on log loss using training and validation data.
3. Once find the best k, run the model to get prediction on validation data and compare it with actual result of validation data to get Logloss performance.

- **Result:**

The performance was the worse of all three models and was worse than random guessing. So we decided not to run the model on test dataset.

➢ Decision Tree

➢ Initiative:

Since this is a class imbalance dataset, decision tree may do a better job in predicting the right class. The splitting process to from a tree forces the model to look at both classes, 0 and 1.

➢ Steps:

1. Remove features/columns that were identified as less significant in lasso model
2. Define a tree control parameter with mincut and minsize
3. Train a decision tree model using the defined control parameter
4. Predict on validation dataset and compare the result with the actual click result to get a Logloss performance
5. Also try using “gini” as the splitting criterion when training the data and try to find a better model
6. Try and change the tree control parameter several times to find the best parameters

➢ Result:

7. Finally decided to use mincut = 5000 and minsize = 20000
8. The best Logloss performance is 0.4376

Predict on the entire Test data

1. We first make a prediction using lasso
2. Then we make a second prediction using decision tree
3. Ensemble method: We cbind the two predictions and get the average of the two columns and take this as the final result of our prediction.

Appendix of all the codes

~~~
---
title: "R Notebook"
output: html_notebook
---
```{r}

rm(list = ls(all = TRUE))

library(data.table)
TrainData <- fread(file = 'ProjectTrainingData.csv')
TestData <- fread(file = 'ProjectTestData.csv')

TrainData <- TrainData[,-1]   # drop 'id' column
TestData <- TestData[,-1]

# we get the hour for each obs
TrainData$hr<-substr(TrainData$hour,7,8)
TestData$hr<-substr(TestData$hour,7,8)

# create list containing unique category of every variable---------------------
TrainDataUnique <- lapply(TrainData,FUN=unique)
TestDataUnique <- lapply(TestData,FUN=unique)

# new categories in test but not training data
#NewCats <- list(NULL)
for (i in 3:ncol(TrainData)){
  wh <- !(TestDataUnique[[i-1]] %in% TrainDataUnique[[i]])
  cat('i=',i,names(TestDataUnique[i-1]),', number of new cats = ',sum(wh),'\n')
  cat('percent in test data = ',round(sum(wh)/length(wh)*100,1),'\n')
  # NewCats[[i]] <- TestDataUnique[[i-1]][wh]
}

# new categories occupy 50% in test of device_id and device_ip, so we will not use them as predictors
TrainData <- TrainData[,c(-2,-11,-12)]
TestData <- TestData[,c(-1,-10,-11)]

fwrite(TrainData,file='Train_v1.csv')
fwrite(TestData,file='Test_v1.csv')
```

```{r prep for scan, run before assign others}
rm(list = ls(all = TRUE))
library(data.table)
TrainData <- fread(file = 'Train_v1.csv')
TestData <- fread(file = 'Test_v1.csv')
~~~

~~~
TrainData <- as.data.frame(TrainData)
TestData <- as.data.frame(TestData)
TestData[] <- lapply(TestData, as.character)
TrainData[] <- lapply(TrainData, as.character)
TrainData$click <- as.numeric(TrainData$click)

```

```{r scan unseen category}
# Distribution of different categories, and then define which category should be coded as 'others'
for (i in 3:ncol(TrainData) ){
print(names(TrainData)[i])

tmp <- sort(table(TrainData[[i]]),decreasing=T)
cat('number of category = ', length(tmp),'\n')

p <- min(length(tmp),20)
tmp <- tmp[1:p]
print(tmp)
plot(1:length(tmp),tmp)
print('----------')
scan()
}

# variables should be coded & top n categories should remained
# c1 5
# banner_pos 4
# site_id 15
# site_domain 15
# site_category 13
# app_id 15
# app_domain 15
# app_category 13
# device_model 15
# device_type 4
# device_conn_type 4
# C14 15
# C15 14
# C16 5
# C17 15
# C18 4
# C19 15
# C20 15
# C21 15
~~~

~~~~
```

```{r assign "others"}
var <- c(2:20)
cate <- c(5,4,15,15,13,15,15,13,15,4,4,15,14,5,15,4,15,15,15)

top=list(NULL)
for (i in 1: length(var)){
  cat <- sort(table(TrainData[,var[i]]),decreasing=T)
  top[[i]] <- c(names(cat[1:cate[i]]),'others')
}

# code small categories as others in both training and test dataset
for (i in 1: length(var)){
TrainData[,var[i]][!TrainData[,var[i]] %in% top[[i]] ] <- 'others'
TestData[,(var[i]-1)][!TestData[,(var[i]-1)] %in% top[[i]] ] <- 'others'
}

#since logistic regression can only train numeric data, we transfer all catogorical variables into numerical type
TrainData[] <- lapply(TrainData, as.factor)
TrainData[] <- lapply(TrainData, as.numeric)
TestData[] <- lapply(TestData, as.factor)
TestData[] <- lapply(TestData, as.numeric)

TrainData <- as.data.table(TrainData)
TestData <- as.data.table(TestData)

fwrite(TrainData,file='Train_v2.csv')
fwrite(TestData,file='Test_v2.csv')
```

```{r load data}
rm(list = ls(all = TRUE))
library(data.table)
TrainData <- fread(file = 'Train_v2.csv')
Trainy <- TrainData[,1]
TrainData <- TrainData[,-1]
TrainData[] <- lapply(TrainData,as.factor)
TrainData <- cbind(Trainy,TrainData)
```

```{r log loss function}
LL <- function(Pred,YVal){
  ll <- -mean(YVal*log(Pred)+(1-YVal)*log(1-Pred))
  return(ll)
}
```
~~~~

~~~
```{r split data into training and validation}
traintable <- table(TrainData$click)
numclick <- unname(traintable[which(names(traintable)==1)])

valid<- sample(nrow(TrainData), size=100000)
ValData <- TrainData[valid,]
table(ValData$click)

trainid<- sample(nrow(TrainData)-100000, size=300000)
TraData <- TrainData[-valid,][trainid,]
table(TraData$click)

# split X and Y
XTrain <- TraData[,-1]
YTrain <- TraData$click

XVal <- ValData[,-1]
YVal <- ValData$click
```

```{r lasso}
library(glmnet)
library(ModelMetrics)
XTrain <- model.matrix(~ .,XTrain)
XVal <- model.matrix(~ .,XVal)
# XTrain <- as.matrix(XTrain)
# XVal <- as.matrix(XVal)

grid <- 10^(seq(-6,-2,length=50))
outll <- glmnet(XTrain,YTrain, family = "binomial", lambda = grid, alpha=1)
p <- predict(outll, newx = as(XVal, "dgCMatrix"), type = "response" )

laout <- matrix(, ncol = 2)
for (i in 1:50)
{ a<- LL(p[,i],YVal)
r <- c(grid[51-i],a)
laout <- rbind(laout,r)
}

plot(laout[,1],laout[,2],xlab='lambda',ylab='log loss')
bestlamR <- laout[which.min(laout[,2]),]

outll <- glmnet(XTrain,YTrain, family = "binomial", lambda = 0.000001, alpha=1)
n <- sum(abs(coef(outll)) != 0) # select 230 variables
# lambda = 2.45e-06, ll = 0.415

outll <- glmnet(XTrain,YTrain, family = "binomial",lambda=2.45e-06, alpha=1)
~~~

~~~
save(outll,file='outll.rda')
```

```{r decision tree}
if(!require("tree")) { install.packages("tree"); require("tree") }
if(!require("ISLR")) { install.packages("ISLR"); require("ISLR") }
if(!require("rgl")) { install.packages("rgl"); require("rgl") }

tc <- tree.control(nrow(TraData), mincut = 5000, minsize = 20000)
DT <- as.formula(paste("click", paste(colnames(TraData[,-1]),collapse=" + "), sep=" ~ "))
outtree <- tree(DT, data=TraData, control=tc, spilt = "gini")

p <- predict(outtree, XVal)
treell <- LL(predict(outtree, XVal), YVal)

save(outtree,file='outtree.rda')
```

```{r predict in entire test dataset}
# import test samples
XTest1 <- fread('Test_v00')
XTest1 <- fread('Test_v01')
XTest1 <- fread('Test_v02')
XTest1 <- fread('Test_v03')
XTest1 <- fread('Test_v04')

XTest1 <- fread('Test_v05')
XTest1 <- fread('Test_v06')
XTest1 <- fread('Test_v07')
XTest1 <- fread('Test_v08')
XTest1 <- fread('Test_v09')

XTest1 <- fread('Test_v10')
XTest1 <- fread('Test_v11')
XTest1 <- fread('Test_v12')
XTest1 <- fread('Test_v13')

#cl <- colnames(XTest1)
colnames(XTest1)<-cl
XTest1[] <- lapply(XTest1,as.factor)

# lasso prediction----------------
XTest1 <- as.matrix(XTest1)
#Ypre_lasso=NULL
Ypre_lasso <- rbind(Ypre_lasso,as.data.frame(predict(outll, newx=as(XTest1, "dgCMatrix"), type =
'response')))
colnames(Ypre_lasso ) <- 'click_lasso'
~~~

~~~
fwrite(Ypre_lasso,file='Ypre_lasso.csv')

# tree prediction----------------
#XTest1 <- model.matrix(~.,XTest1)
#Ypre_tree=NULL
Ypre_tree <- rbind(Ypre_tree,as.data.frame(predict(outtree, XTest1)))
colnames(Ypre_tree ) <- 'click_tree'
fwrite(Ypre_tree,file='Ypre_tree.csv')

# column bind these two predictions
TestData <- cbind(Ypre_lasso,Ypre_tree)
fwrite(TestData,file='Test_pred.csv')
```
~~~