Model Selection for Predicting Red Hat Business Value

In this project, we conducted a couple of machine learning algorithms on Red Hat Business data set, which is pretty large(almost two millions*56). We mainly compared SVM(linear and nonlinear), Random Forest, Light GBM, Xgboost, Neural Network, Multinomial log-linear Model. And it proved Xgoost and Light GBM have the highest accuracy around 98%.

Step 0: Load the packages

```
#list.of.packages <- c("data.table","FeatureHashing","xgboost","dplyr","Matrix","caret","randomForest","e1071","lightgbm","magr
#new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
#if(length(new.packages))
  #{
   #install.packages(new.packages)
  #}

#library(data.table)
#library(FeatureHashing)
#library(xgboost)
#library(dplyr)
#library(Matrix)
#library(e1071)
#library(lightgbm)
#library(magrittr)
#library(randomForest)
```

Step 1: Data processing: turn categorical variables into numeric variables

The process is showed in DOC file: data_processing.Rmd and data_processing_uniquea_same.Rmd and the results are in Output links (Since the limit size of file is 25MB)

In the data processing step, we converted all values of features into numeric values, and get the results : train_num. The code is in /doc/data_processing.Rmd

Since many features are categorical variables, in order to avoid the dummy variable trap, we set all frequency =1 character values in one column to be same numeric values. The code is in /doc/data_processing_uniquea_same.Rmd

```
#train_num<- fread("../output/train_num.csv") %>% as.data.frame()
#train_num_runique<- fread("../output/train_num_runique.csv") %>% as.data.frame()
```

Step 2: Implement models

Model 1: SVM

```
# Linear SVM
#source("/Users/yg2477/Desktop/fall2017-project5-proj5-grp3-master/lib/svm_linear.R")
# svm_linear(train_num_runique, 5)

#RBF Kernel
#source("/Users/yg2477/Desktop/fall2017-project5-proj5-grp3-master/lib/svm_rbf.R")
#svm_rbf(train_num_runique, 5)
```

We can find the accuracy of linear svm is 83.09%, while the RBF Kernel behaves even better, the accuracy is 88.22%.

Model 2: Random Forest

We need to do some pre-processing before using the Random Forest Model. Indeed, all the variables that we use in this model are categorical variables except ‘people date’ and ‘activity date’. RF implementation in R cannot deal with categorical variables if some variables have more than 52 different labels.

Pre-processing: For this model we decided to separate ‘activitygroup’=1 with ‘activity_group’!=1, since these activities in these two groups of categories doesn’t have the same the same variables (variables for which ‘activity group’=1 has 9 additional variables). We also decided to tranform variables with more than 52 labels into numeric values.

Model: We run a Random Forest model with parameter mtrt=sqrt(44) and pruning (importance=TRUE) for bothe activity1 and without_activity1

```
  #trainingIndex=sample(nrow(thisdata), round((nrow(thisdata)*0.40)))
  #data.test=thisdata[-trainingIndex,]
  #data.train=thisdata[trainingIndex,]

  #rf.model = randomForest(as.factor(outcome)~., data=thisdata, subset= trainingIndex, mtry =7, importance=TRUE)
  #importance(rf.model)
#list(rf.model, data.test,trainingIndex)


  #train_num = train_num[, !names(train_num) %in% c("V1")]
  #length(train_num)
  #fit <- randomForest(as.factor(outcome)~., data= train_num, mtry=8,
                      #importance=TRUE,
                      #ntree=25)
```

Model 3: Light GBM

```
# Data Pre-processing, separation into two data set for group activity =1 and group_activity !=1
#source("../lib/lightGBM.R")
#lightGBM(train_num_runique)
```

Pre-processing: We first pre-process the file separating the data set into two part, one corresponding to activity_group=1 and the other one corresponding to activity_group!=1.

Results:

We run the Light GBM model on the first data set: activity =1 with the following parameter: max_depth = 8, learning_rate=0.1, 1500 iterations We obtain the following accuracy: train’s l2: 0.0165753 valid’s l2: 0.0187008, so an accuracy of 98.13% on the test set We run the code in 2.93 mins.

We then run the Light GBM model on the second data set: activity !=1 with the following parameter: max_depth = 8, learning_rate=0.1, 1500 iterations We obtain the following accuracy: train’s l2:0.0264289 valid’s l2:0.0285721, so an accuracy of 97.15% on the test set We run the code in 7.04 mins.

Model 4: Xgboost

```
#source("/Users/linhan/Desktop/fall2017-project5-proj5-grp3/lib/xgboost.R")
```

In order to make computation more efficient, we converted all non-sparse columns to sparse matrix and rbind them with other sparse columns.

In xgboost model, I trained the model using different parameters combinations like size of each boosting step of 0.02,0.05,0.1 and maximum depth of the tree of 5,10,15. I find that when eta= 0.02, max_depth=10, the model have the highest accuracy. So I used it as the best model to do the following step—(5 folds cross-validation), and get the results:

```
#run.xgb<- TRUE
#if(run.xgb){
 #system.time(xgb(train_num_runique, K=5))
#}
#Results:
# [1]   train-auc:0.880851+0.000064 test-auc:0.880566+0.000348
# [2]   train-auc:0.889844+0.000052 test-auc:0.889530+0.000345
# [3]   train-auc:0.898670+0.000042 test-auc:0.898327+0.000352
# [4]   train-auc:0.906955+0.000033 test-auc:0.906586+0.000337
# [5]   train-auc:0.914586+0.000032 test-auc:0.914195+0.000327
# [6]   train-auc:0.921505+0.000030 test-auc:0.921093+0.000312
# [7]   train-auc:0.927853+0.000030 test-auc:0.927423+0.000300
# [8]   train-auc:0.933761+0.000032 test-auc:0.933312+0.000286
# [9]   train-auc:0.939214+0.000034 test-auc:0.938748+0.000266
# [10]  train-auc:0.944225+0.000037 test-auc:0.943742+0.000252
# [11]  train-auc:0.948777+0.000040 test-auc:0.948280+0.000240
# [12]  train-auc:0.952875+0.000042 test-auc:0.952366+0.000227
# [13]  train-auc:0.956535+0.000043 test-auc:0.956016+0.000218
# [14]  train-auc:0.959791+0.000043 test-auc:0.959264+0.000211
# [15]  train-auc:0.962671+0.000044 test-auc:0.962139+0.000205
# [16]  train-auc:0.965214+0.000044 test-auc:0.964677+0.000199
# [17]  train-auc:0.967458+0.000044 test-auc:0.966918+0.000192
# [18]  train-auc:0.969441+0.000043 test-auc:0.968898+0.000185
# [19]  train-auc:0.971194+0.000042 test-auc:0.970650+0.000178
# [20]  train-auc:0.972748+0.000042 test-auc:0.972203+0.000171
# [21]  train-auc:0.974136+0.000042 test-auc:0.973589+0.000163
# [22]  train-auc:0.975379+0.000042 test-auc:0.974832+0.000157
# [23]  train-auc:0.976492+0.000041 test-auc:0.975945+0.000152
# [24]  train-auc:0.977499+0.000041 test-auc:0.976951+0.000149
# [25]  train-auc:0.978412+0.000040 test-auc:0.977865+0.000144
# [26]  train-auc:0.979244+0.000039 test-auc:0.978697+0.000140
# [27]  train-auc:0.979999+0.000039 test-auc:0.979451+0.000136
# [28]  train-auc:0.980687+0.000038 test-auc:0.980140+0.000134
# [29]  train-auc:0.981319+0.000037 test-auc:0.980772+0.000131
# [30]  train-auc:0.981898+0.000036 test-auc:0.981351+0.000129
# [31]  train-auc:0.982431+0.000036 test-auc:0.981884+0.000126
# [32]  train-auc:0.982921+0.000035 test-auc:0.982373+0.000125
# [33]  train-auc:0.983368+0.000034 test-auc:0.982821+0.000123
# [34]  train-auc:0.983779+0.000034 test-auc:0.983232+0.000121
# [35]  train-auc:0.984159+0.000033 test-auc:0.983612+0.000120
# [36]  train-auc:0.984509+0.000032 test-auc:0.983962+0.000118
# [37]  train-auc:0.984833+0.000032 test-auc:0.984285+0.000117
# [38]  train-auc:0.985132+0.000031 test-auc:0.984585+0.000116
# [39]  train-auc:0.985408+0.000031 test-auc:0.984861+0.000114
# [40]  train-auc:0.985665+0.000031 test-auc:0.985119+0.000113
# [41]  train-auc:0.985904+0.000030 test-auc:0.985358+0.000112
# [42]  train-auc:0.986126+0.000031 test-auc:0.985579+0.000110
# [43]  train-auc:0.986331+0.000030 test-auc:0.985785+0.000110
# [44]  train-auc:0.986524+0.000029 test-auc:0.985977+0.000109
# [45]  train-auc:0.986704+0.000029 test-auc:0.986158+0.000109
# [46]  train-auc:0.986872+0.000029 test-auc:0.986326+0.000108
# [47]  train-auc:0.987030+0.000028 test-auc:0.986484+0.000108
# [48]  train-auc:0.987177+0.000028 test-auc:0.986632+0.000107
# [49]  train-auc:0.987316+0.000028 test-auc:0.986770+0.000107
# [50]  train-auc:0.987446+0.000028 test-auc:0.986901+0.000106
# [51]  train-auc:0.987569+0.000028 test-auc:0.987024+0.000106
# [52]  train-auc:0.987686+0.000028 test-auc:0.987140+0.000106
# [53]  train-auc:0.987796+0.000027 test-auc:0.987250+0.000105
# [54]  train-auc:0.987900+0.000027 test-auc:0.987355+0.000105
# [55]  train-auc:0.987999+0.000027 test-auc:0.987454+0.000105
# [56]  train-auc:0.988093+0.000027 test-auc:0.987548+0.000104
# [57]  train-auc:0.988182+0.000027 test-auc:0.987637+0.000104
# [58]  train-auc:0.988267+0.000027 test-auc:0.987721+0.000104
# [59]  train-auc:0.988348+0.000027 test-auc:0.987802+0.000104
# [60]  train-auc:0.988424+0.000026 test-auc:0.987878+0.000104
# [61]  train-auc:0.988497+0.000026 test-auc:0.987951+0.000104
# [62]  train-auc:0.988567+0.000025 test-auc:0.988021+0.000104
# [63]  train-auc:0.988634+0.000025 test-auc:0.988088+0.000103
# [64]  train-auc:0.988699+0.000025 test-auc:0.988152+0.000104
# [65]  train-auc:0.988760+0.000025 test-auc:0.988213+0.000103
# [66]  train-auc:0.988820+0.000025 test-auc:0.988272+0.000103
# [67]  train-auc:0.988876+0.000024 test-auc:0.988328+0.000103
# [68]  train-auc:0.988931+0.000024 test-auc:0.988382+0.000103
# [69]  train-auc:0.988983+0.000024 test-auc:0.988434+0.000102
# [70]  train-auc:0.989033+0.000024 test-auc:0.988485+0.000102
# [71]  train-auc:0.989082+0.000024 test-auc:0.988533+0.000102
# [72]  train-auc:0.989130+0.000024 test-auc:0.988580+0.000102
# [73]  train-auc:0.989175+0.000024 test-auc:0.988625+0.000102
# [74]  train-auc:0.989219+0.000024 test-auc:0.988669+0.000101
# [75]  train-auc:0.989261+0.000024 test-auc:0.988711+0.000101
# [76]  train-auc:0.989302+0.000024 test-auc:0.988751+0.000102
# [77]  train-auc:0.989342+0.000024 test-auc:0.988790+0.000101
# [78]  train-auc:0.989380+0.000024 test-auc:0.988828+0.000101
# [79]  train-auc:0.989417+0.000024 test-auc:0.988864+0.000101
# [80]  train-auc:0.989453+0.000024 test-auc:0.988900+0.000101
# [81]  train-auc:0.989488+0.000024 test-auc:0.988935+0.000101
# [82]  train-auc:0.989522+0.000024 test-auc:0.988968+0.000101
# [83]  train-auc:0.989555+0.000024 test-auc:0.989000+0.000101
# [84]  train-auc:0.989587+0.000024 test-auc:0.989032+0.000101
# [85]  train-auc:0.989618+0.000024 test-auc:0.989063+0.000102
# [86]  train-auc:0.989648+0.000024 test-auc:0.989093+0.000102
# [87]  train-auc:0.989678+0.000024 test-auc:0.989122+0.000102
# [88]  train-auc:0.989706+0.000024 test-auc:0.989150+0.000102
# [89]  train-auc:0.989734+0.000024 test-auc:0.989177+0.000102
# [90]  train-auc:0.989762+0.000024 test-auc:0.989204+0.000102
# [91]  train-auc:0.989789+0.000024 test-auc:0.989230+0.000102
# [92]  train-auc:0.989815+0.000024 test-auc:0.989256+0.000103
# [93]  train-auc:0.989840+0.000024 test-auc:0.989280+0.000103
# [94]  train-auc:0.989865+0.000024 test-auc:0.989305+0.000103
# [95]  train-auc:0.989890+0.000024 test-auc:0.989329+0.000103
# [96]  train-auc:0.989914+0.000024 test-auc:0.989352+0.000103
# [97]  train-auc:0.989937+0.000024 test-auc:0.989375+0.000104
# [98]  train-auc:0.989960+0.000024 test-auc:0.989397+0.000104
# [99]  train-auc:0.989983+0.000024 test-auc:0.989419+0.000104
# [100] train-auc:0.990005+0.000024 test-auc:0.989440+0.000104
```

From the results, we can see that the accuracy of Xgboost model is 98.94%. The running time is 1357.69 seconds.

Model 5: Neural Network

The multiple-layer neural network codes is in /doc/neural_network.ipynb. We use 5 folds cross-validation to do evaluation. The results are as following:

CV Score	hidden layer	running time
75.57%	1	148.94s
82.03%	6	354.62s
80.56%	10	1639.65s
83.72%	15	4380.30s