E6690 Project

Dec 17, 2018

```
library(knitr)
opts_chunk$set(tidy.opts = list(width.cutoff = 60), tidy = TRUE)
```

0. Import libraries and data

```
library(class)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
library(gains)
library(rlang)
library(geiger)
```

```
## Loading required package: ape
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(adabag)
```

```
## Loading required package: rpart

## Loading required package: foreach

## Loading required package: doParallel

## Loading required package: iterators

## Loading required package: parallel
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following object is masked from 'package:rlang':
##
##     :=
```

```
library(ggplot2)
library(Ckmeans.1d.dp)
library(xgboost)

setwd("/Users/qinqingao/Desktop/Columbia/Courses/Fall 2018/EECS 6690/Project")
require(gdata)
```

```
## Loading required package: gdata

## gdata: read.xls support for 'XLS' (Excel 97-2004) files ENABLED.

##
```

```
## gdata: read.xls support for 'XLSX' (Excel 2007+) files ENABLED.

##
## Attaching package: 'gdata'

## The following objects are masked from 'package:data.table':
##
##     first, last

## The following objects are masked from 'package:rlang':
##
##     env, ll

## The following object is masked from 'package:stats':
##
##     nobs

## The following object is masked from 'package:utils':
##
##     object.size

## The following object is masked from 'package:base':
##
##     startsWith
```

```
raw <- read.xls("default of credit card clients.xls", sheet = 1,
    skip = 1, row.names = 1)
dim(raw)
```

```
## [1] 30000    24
```

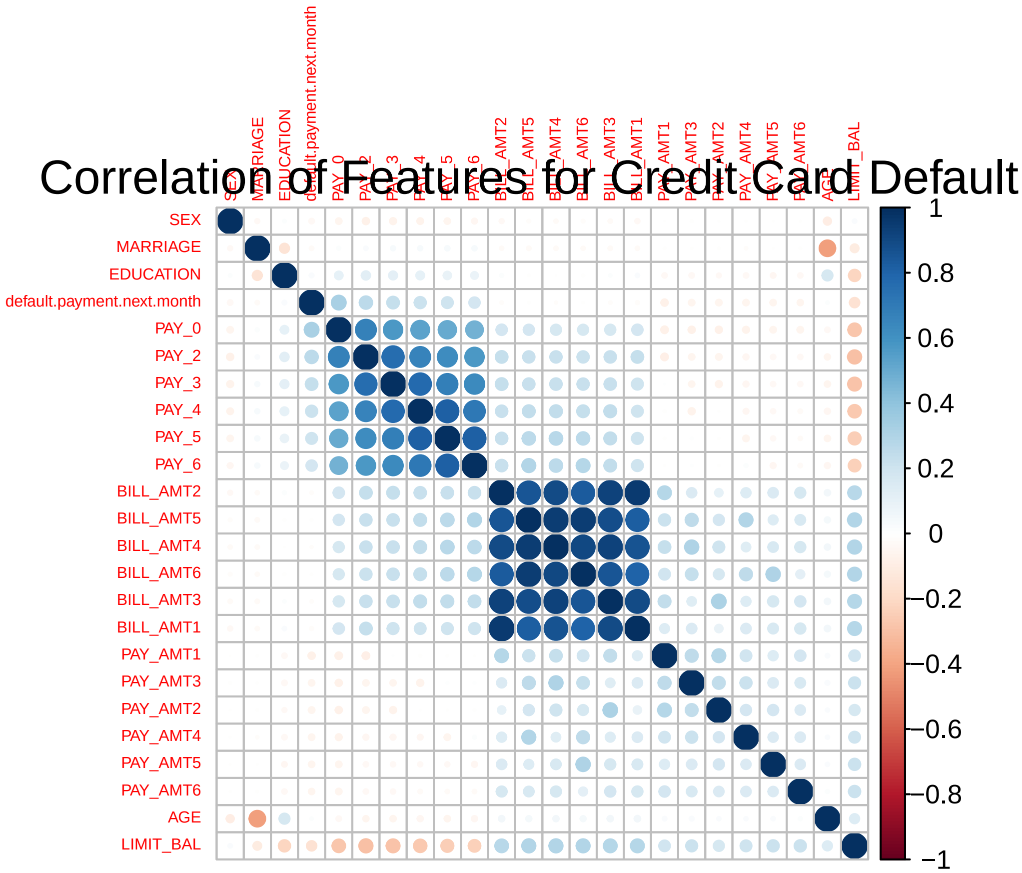
```
# [1] 30000 24

# check any null value, none (30000 * 24 = 720000)
table(is.na(raw))
```

```
##
##  FALSE
## 720000
```

```
# FALSE 720000
```

1. Exploratory Data Analysis with correlation plot

```
# png('/Users/qinqingao/Desktop/Columbia/Courses/Fall
# 2018/EECS 6690/Project/figs/corr_plot.png')
corrplot(cor(raw), order = "AOE", tl.cex = 0.5)
mtext("Correlation of Features for Credit Card Default", at = 12,
    line = -1, cex = 1.5)
```




```
# dev.off()
```

2. Train, test split, prepare for model training

```
# train-test split, 80%-20%
set.seed(2018)
sample_row_num <- sample(nrow(raw), nrow(raw) * 0.8)

train <- raw[sample_row_num, ]
test <- raw[-sample_row_num, ]

train_label <- train[, ncol(train)]
test_label <- test[, ncol(test)]
```

3. Feature engineering with variable normalization

```
# normalizing numerical variables
scale01 <- function(x) {
    (x - min(x))/(max(x) - min(x))
}

train_norm = train
valid_norm = test
```

```r
for (name in names(train)) {
    if (name != "default.payment.next.month") {
        train_norm[name] <- scale01(train_norm[name])
        valid_norm[name] <- scale01(valid_norm[name])
    }
}
```

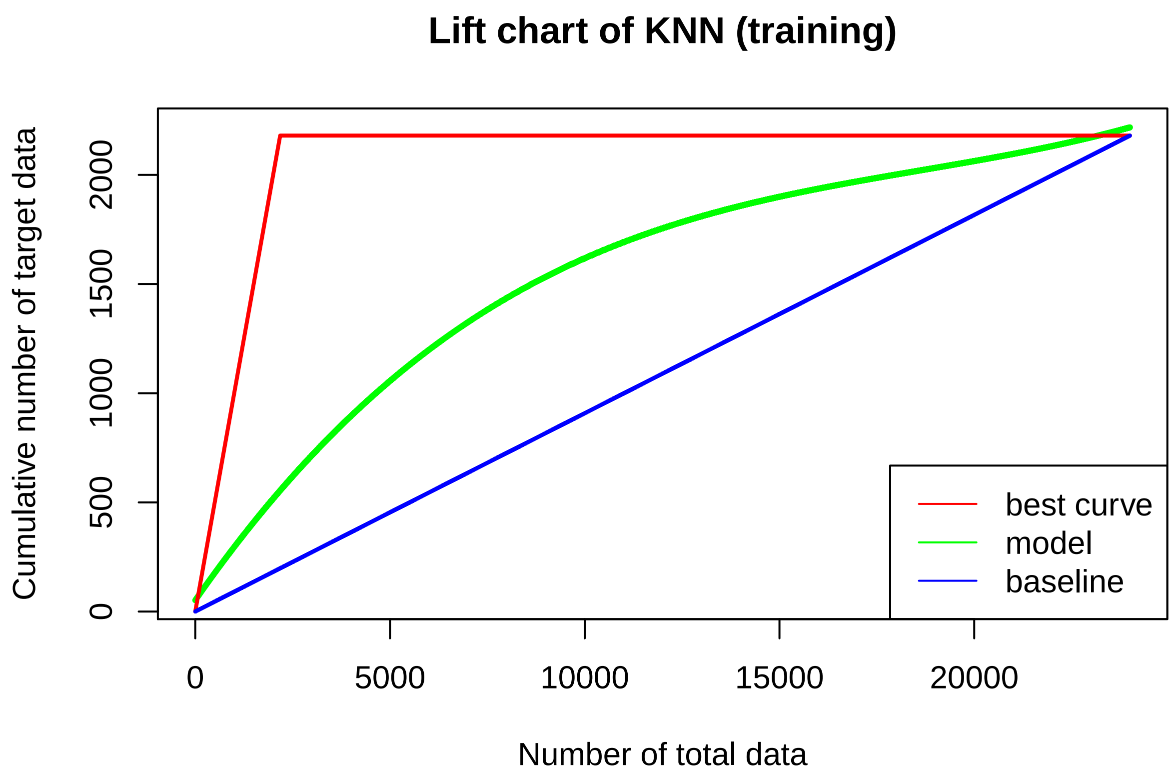
4. Modeling and analysis with kNN

```r
#################################

# 1. kNN #

#################################

########################

# 1.1 kNN Train #

########################

# Part 1.1.1: train data with kNN
train_knn = train_norm
train_knn$default.payment.next.month <- NULL
knnt = knn(train_knn, train_knn, train$default.payment.next.month,
    k = 100, prob = TRUE)
tt = table(pred = knnt, actual = train$default.payment.next.month)
error_train = 1 - sum(diag(tt))/sum(tt)
error_train
```

```
## [1] 0.1899583
```

```r
# [1] 0.1899583


# Lift chart for train - knn
PredKNNLabel = data.frame(knnt)
names(PredKNNLabel) <- "PredKNNLabel"
PredKNNScore = attr(knnt, "prob")  # its the propotion of the wining class

# convert it into the probablity of default
for (i in 1:length(PredKNNScore)) {
    if (knnt[i] == 0) {
        PredKNNScore[i] = 1 - PredKNNScore[i]
    }
}
PredKNNScore = data.frame(PredKNNScore)
names(PredKNNScore) <- "PredKNNScore"
train_norm = data.frame(train_norm, PredKNNLabel, PredKNNScore)
```

```r
# Part 1.1.2: plot train lift curve for kNN
gtt = gains(actual = train_norm$default.payment.next.month, predicted = train_norm$PredKNNScore,
    optimal = TRUE)
cpt_y = gtt$cume.pct.of.total
cpt_x = gtt$depth

predicted = table(train_norm$PredKNNLabel)[2]
xx = cpt_x/100 * 24000
yy = cpt_y * predicted
# plot(xx,yy)
xx = prepend(xx, 0, before = 1)
yy = prepend(yy, 0, before = 1)
fit = lm(yy ~ poly(xx, 3, raw = TRUE))
xx = 0:24000
model_yy = predict(fit, data.frame(xx))

# png('KNN_lift_chart_train.png')
plot(xx, model_yy, col = "green", xlab = "Number of total data",
    ylab = "Cumulative number of target data", type = "l", lwd = 3)

best_yy = rep(predicted, 24001)
for (i in 0:predicted) {
    best_yy[i + 1] = i
}

lines(xx, best_yy, col = "red", lwd = 2)

base_yy = predicted/24000 * xx
lines(xx, base_yy, col = "blue", lwd = 2)

legend("bottomright", legend = c("best curve", "model", "baseline"),
    col = c("red", "green", "blue"), lwd = c(1, 1, 1), cex = 1)
title("Lift chart of KNN (training)")
```



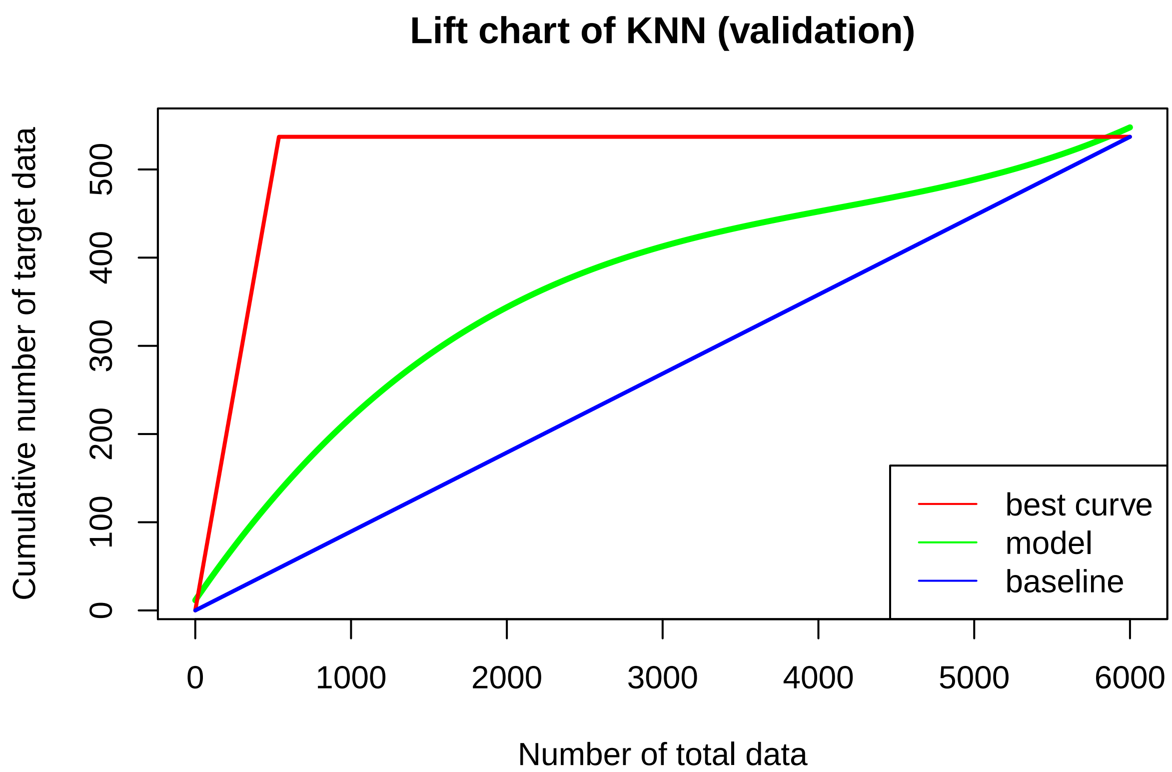

```
# dev.off()


# Calculate area ratio
a1t = sum(model_yy - base_yy)
a2t = sum(best_yy - base_yy)
a1t/a2t
```

```
## [1] 0.4627081
```

```
# [1] 0.4627081



#########################

# 1.2 kNN Test #

#########################


# Part 1.2.1: test data with kNN
valid_knn = valid_norm
valid_knn$default.payment.next.month <- NULL
knnv = knn(train_knn, valid_knn, train$default.payment.next.month,
    k = 100, prob = TRUE)

tv = table(pred = knnv, actual = test$default.payment.next.month)
error_valid = 1 - sum(diag(tv))/sum(tv)
error_valid
```

```
## [1] 0.1943333
```

```r
# [1] 0.1938333


PredKNNLabelV = data.frame(knnv)
names(PredKNNLabelV) <- "PredKNNLabelV"
PredKNNScoreV = attr(knnv, "prob")  # its the propotion of the wining class

# convert it into the probablity of default
for (i in 1:length(PredKNNScoreV)) {
    if (knnv[i] == 0) {
        PredKNNScoreV[i] = 1 - PredKNNScoreV[i]
    }
}
PredKNNScoreV = data.frame(PredKNNScoreV)
names(PredKNNScoreV) <- "PredKNNScoreV"
valid_norm = data.frame(valid_norm, PredKNNLabelV, PredKNNScoreV)



# Part 1.2.3: plot test lift curve for kNN
gtv = gains(actual = valid_norm$default.payment.next.month, predicted = valid_norm$PredKNNScoreV,
    optimal = TRUE)
cpv_y = gtv$cume.pct.of.total
cpv_x = gtv$depth

predictedV = table(valid_norm$PredKNNLabelV)[2]
xxv = cpv_x/100 * 6000
yyv = cpv_y * predictedV

xxv = prepend(xxv, 0, before = 1)
yyv = prepend(yyv, 0, before = 1)
fitv = lm(yyv ~ poly(xxv, 3, raw = TRUE))
xxv = 0:6000
model_yyv = predict(fitv, data.frame(xxv))

# png('KNN_lift_chart_train.png')
plot(xxv, model_yyv, col = "green", xlab = "Number of total data",
    ylab = "Cumulative number of target data", type = "l", lwd = 3)

best_yyv = rep(predictedV, 6001)
for (i in 0:predictedV) {
    best_yyv[i + 1] = i
}

lines(xxv, best_yyv, col = "red", lwd = 2)

base_yyv = predictedV/6000 * xxv
lines(xxv, base_yyv, col = "blue", lwd = 2)

legend("bottomright", legend = c("best curve", "model", "baseline"),
    col = c("red", "green", "blue"), lwd = c(1, 1, 1), cex = 1)
title("Lift chart of KNN (validation)")
```




```
# dev.off()

# Calculate area ratio
a1v = sum(model_yyv - base_yyv)
a2v = sum(best_yyv - base_yyv)
a1v/a2v
```

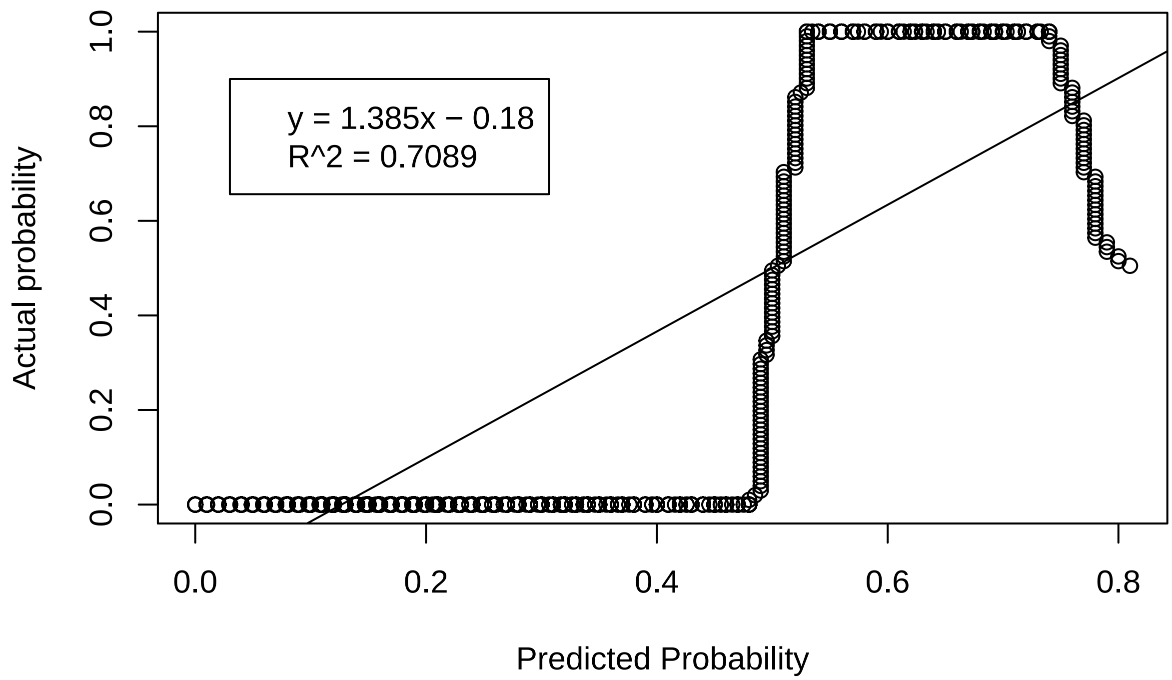
```
## [1] 0.4087533
```

```
# [1] 0.408529



# Part 1.1.3: Use SSM(Sorting Smoothing Method) to estimate
# real probability

# 1. order the valid data according to predictive probability
valid_norm_sort = valid_norm[order(PredKNNScoreV), ]
# 2. use SSM formula to evaluate actural probability 'Pi', we
# choose n =50 according to the paper
VALIDSIZE = dim(valid_norm)[1]
n = 50
actural_p_valid = rep(0, VALIDSIZE)
# pred_valid = valid_sort$PredTreeScoreValid
pred_valid = round(valid_norm_sort$PredKNNScoreV)
pred_valid = prepend(pred_valid, rep(0, n), before = 1)
pred_valid = append(pred_valid, rep(0, n))
for (i in 1:VALIDSIZE) {
    actural_p_valid[i] = sum(pred_valid[i:(i + 2 * n)])/(2 *
        n + 1)
```

```r
}
valid_norm_sort = data.frame(valid_norm_sort, actural_p_valid)


# png('Scatter plot diagram of KNN.png')
plot(valid_norm_sort$PredKNNScoreV, valid_norm_sort$actural_p_valid,
    xlab = "Predicted Probability", ylab = "Actual probability")

yy = valid_norm_sort$actural_p_valid
xx = valid_norm_sort$PredKNNScoreV
actual_fit = lm(yy ~ xx)

xx = seq(0, 1:0.1)
yy = predict(actual_fit, data.frame((xx)))
lines(xx, yy)

summary(actual_fit)
```

```
##
## Call:
## lm(formula = yy ~ xx)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.47324 -0.05798  0.00899  0.07597  0.45978
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.169739   0.002908  -58.37   <2e-16 ***
## xx           1.339546   0.011379  117.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1498 on 5998 degrees of freedom
## Multiple R-squared:  0.6979, Adjusted R-squared:  0.6979
## F-statistic: 1.386e+04 on 1 and 5998 DF,  p-value: < 2.2e-16
```

```r
legend(0.03, 0.9, legend = c("y = 1.385x - 0.18", "R^2 = 0.7089"))
```




```
# dev.off()
```

5. Modeling and analysis with Logistic Regression

```
#################################

# 2. logistic regression #

#################################


#########################

# 2.1 LR Train #

#########################


# Part 2.1.1: train data with LR
lr <- glm(default.payment.next.month ~ ., family = binomial(link = "logit"),
    data = train)
# summary(lr)

# check p value, and eliminate the ones have high pvalue

# Call: glm(formula = default.payment.next.month ~ ., family
# = binomial(link = 'logit'), data = train)

# Deviance Residuals: Min 1Q Median 3Q Max -3.1541 -0.7014
# -0.5465 -0.2910 3.9405
```

```r
# Coefficients: Estimate Std. Error z value Pr(>|z|)
# (Intercept) -7.533e-01 1.324e-01 -5.689 1.28e-08 ***
# LIMIT_BAL -8.199e-07 1.753e-07 -4.677 2.91e-06 *** SEX
# -1.150e-01 3.430e-02 -3.353 0.000799 *** EDUCATION
# -1.005e-01 2.352e-02 -4.272 1.94e-05 *** MARRIAGE
# -1.360e-01 3.537e-02 -3.844 0.000121 *** AGE 9.154e-03
# 1.976e-03 4.634 3.59e-06 *** PAY_0 5.749e-01 1.968e-02
# 29.218 < 2e-16 *** PAY_2 8.570e-02 2.253e-02 3.805 0.000142
# *** PAY_3 5.946e-02 2.538e-02 2.342 0.019156 * PAY_4
# 2.151e-02 2.821e-02 0.763 0.445693 PAY_5 4.622e-02
# 3.011e-02 1.535 0.124742 PAY_6 5.466e-03 2.479e-02 0.220
# 0.825497 BILL_AMT1 -5.568e-06 1.285e-06 -4.334 1.47e-05 ***
# BILL_AMT2 1.884e-06 1.689e-06 1.115 0.264799 BILL_AMT3
# 1.558e-06 1.481e-06 1.052 0.292782 BILL_AMT4 7.592e-07
# 1.528e-06 0.497 0.619230 BILL_AMT5 -8.107e-07 1.788e-06
# -0.453 0.650287 BILL_AMT6 1.252e-06 1.406e-06 0.890
# 0.373233 PAY_AMT1 -1.310e-05 2.569e-06 -5.099 3.41e-07 ***
# PAY_AMT2 -8.017e-06 2.180e-06 -3.678 0.000235 *** PAY_AMT3
# -3.175e-06 1.946e-06 -1.632 0.102756 PAY_AMT4 -5.093e-06
# 2.186e-06 -2.330 0.019814 * PAY_AMT5 -4.207e-06 2.045e-06
# -2.057 0.039649 * PAY_AMT6 -2.301e-06 1.469e-06 -1.567
# 0.117105 --- Signif. codes: 0 '***' 0.001 '**' 0.01 '*'
# 0.05 '.' 0.1 ' ' 1

# (Dispersion parameter for binomial family taken to be 1)

# Null deviance: 25390 on 23999 degrees of freedom Residual
# deviance: 22336 on 23976 degrees of freedom AIC: 22384

# Number of Fisher Scoring iterations: 6


names = names(train)
f2 <- as.formula(paste("default.payment.next.month ~", paste(names[!names %in%
    c("PAY_AMT6", "PAY_AMT3", "BILL_AMT6", "BILL_AMT5", "BILL_AMT4",
        "BILL_AMT3", "BILL_AMT2", "PAY_6", "PAY_5", "PAY_4")],
    collapse = " + ")))
lr2 = glm(f2, data = train_norm, family = binomial(link = "logit"))
```

```
## Warning in model.matrix.default(mt, mf, contrasts): the response appeared
## on the right-hand side and was dropped

## Warning in model.matrix.default(mt, mf, contrasts): problem with term 14 in
## model.matrix: no columns are assigned
```

```r
# summary(lr2)


pred = predict(lr2, train_norm, type = "response")
```

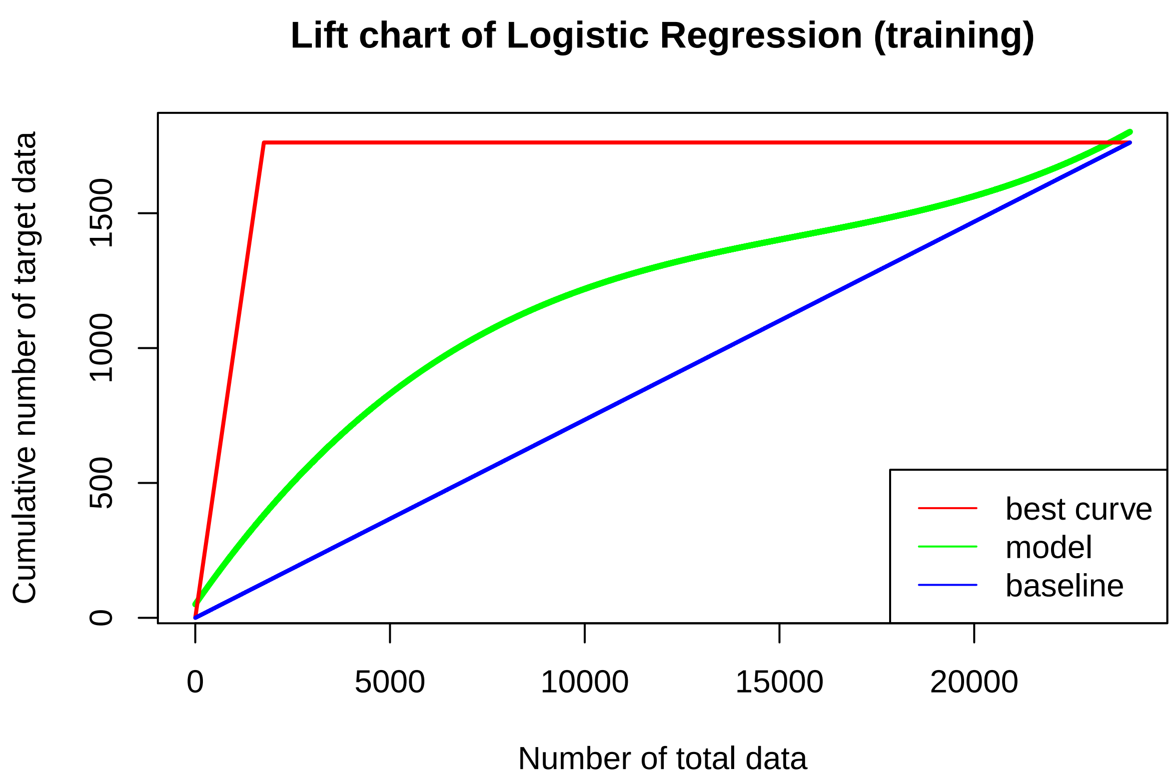
```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```r
PredLabel = data.frame(round(pred))
names(PredLabel) <- "PredLabel"
```

```r
PredScore = data.frame(pred)
names(PredScore) <- "PredScore"

train_norm = data.frame(train_norm, PredLabel, PredScore)
tt = table(pred = train_norm$PredLabel, actual = train_norm$default.payment.next.month)
error_train = 1 - sum(diag(tt))/sum(tt)
error_train
```

```
## [1] 0.191625
```

```r
# [1] 0.191625



# Part 2.1.2: plot train lift curve for LR
gtt = gains(actual = train_norm$default.payment.next.month, predicted = train_norm$PredScore,
    optimal = TRUE)
cpt_y = gtt$cume.pct.of.total
cpt_x = gtt$depth

predicted = table(train_norm$PredLabel)[2]
xx = cpt_x/100 * 24000
yy = cpt_y * predicted
# plot(xx,yy)
xx = prepend(xx, 0, before = 1)
yy = prepend(yy, 0, before = 1)
fit = lm(yy ~ poly(xx, 3, raw = TRUE))
xx = 0:24000
model_yy = predict(fit, data.frame(xx))

# png('LR_lift_chart_train.png')
plot(xx, model_yy, col = "green", xlab = "Number of total data",
    ylab = "Cumulative number of target data", type = "l", lwd = 3)

best_yy = rep(predicted, 24001)
for (i in 0:predicted) {
    best_yy[i + 1] = i
}

lines(xx, best_yy, col = "red", lwd = 2)

base_yy = predicted/24000 * xx
lines(xx, base_yy, col = "blue", lwd = 2)

legend("bottomright", legend = c("best curve", "model", "baseline"),
    col = c("red", "green", "blue"), lwd = c(1, 1, 1), cex = 1)
title("Lift chart of Logistic Regression (training)")
```




```r
# dev.off()


# Calculate area ratio
a1t = sum(model_yy - base_yy)
a2t = sum(best_yy - base_yy)
a1t/a2t
```

```
## [1] 0.3661037
```

```r
# [1] 0.3661037



#########################

# 2.2 LR Test #

#########################


# Part 2.2.1: test data with LR
predV = predict(lr2, valid_norm, type = "response")
```

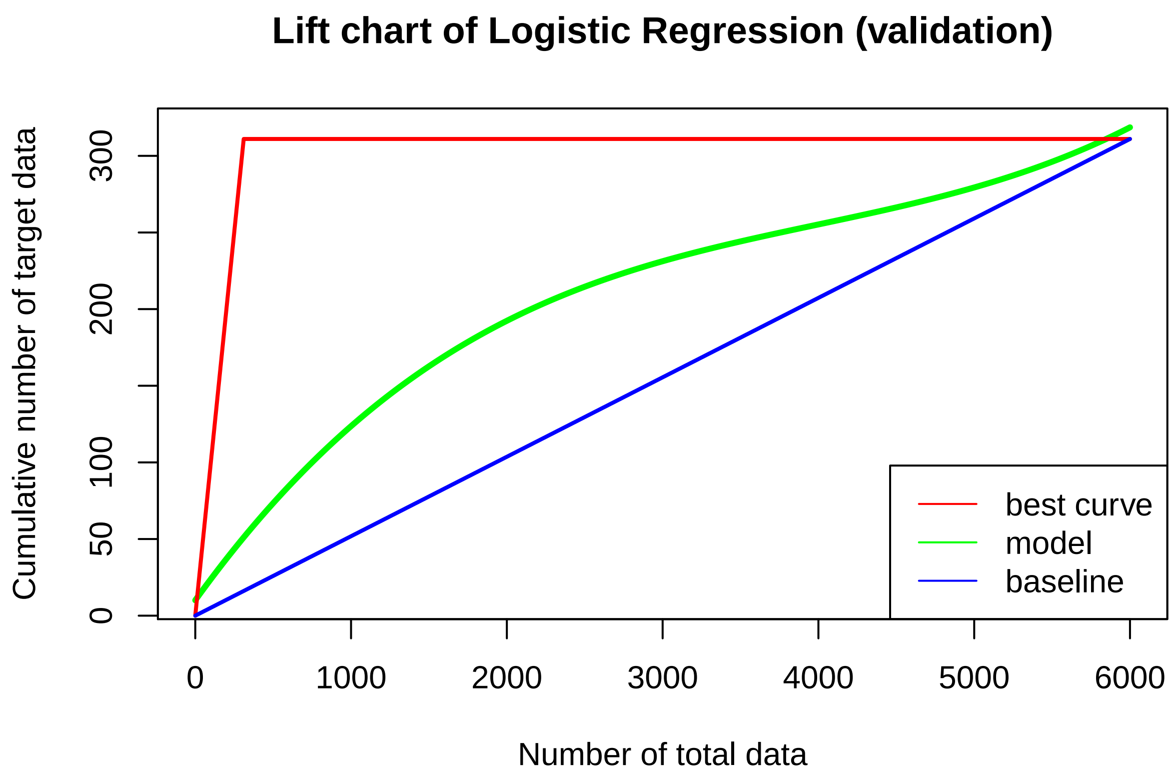
```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```r
PredLabelV = data.frame(round(predV))
names(PredLabelV) <- "PredLabelV"

PredScoreV = data.frame(predV)
```

```r
names(PredScoreV) <- "PredScoreV"

valid_norm = data.frame(valid_norm, PredLabelV, PredScoreV)
tv = table(pred = valid_norm$PredLabelV, actual = valid_norm$default.payment.next.month)
error_valid = 1 - sum(diag(tv))/sum(tv)
error_valid
```

```
## [1] 0.195
```

```r
# [1] 0.195



# Part 2.2.2: plot test lift curve for LR
gtv = gains(actual = valid_norm$default.payment.next.month, predicted = valid_norm$PredScoreV,
    optimal = TRUE)
cpv_y = gtv$cume.pct.of.total
cpv_x = gtv$depth

predictedV = table(valid_norm$PredLabelV)[2]
xxv = cpv_x/100 * 6000
yyv = cpv_y * predictedV

xxv = prepend(xxv, 0, before = 1)
yyv = prepend(yyv, 0, before = 1)
fitv = lm(yyv ~ poly(xxv, 3, raw = TRUE))
xxv = 0:6000
model_yyv = predict(fitv, data.frame(xxv))

# png('KNN_lift_chart_train.png')
plot(xxv, model_yyv, col = "green", xlab = "Number of total data",
    ylab = "Cumulative number of target data", type = "l", lwd = 3)

best_yyv = rep(predictedV, 6001)
for (i in 0:predictedV) {
    best_yyv[i + 1] = i
}

lines(xxv, best_yyv, col = "red", lwd = 2)

base_yyv = predictedV/6000 * xxv
lines(xxv, base_yyv, col = "blue", lwd = 2)

legend("bottomright", legend = c("best curve", "model", "baseline"),
    col = c("red", "green", "blue"), lwd = c(1, 1, 1), cex = 1)
title("Lift chart of Logistic Regression (validation)")
```




```
# dev.off()

# Calculate area ratio
a1v = sum(model_yyv - base_yyv)
a2v = sum(best_yyv - base_yyv)
a1v/a2v
```

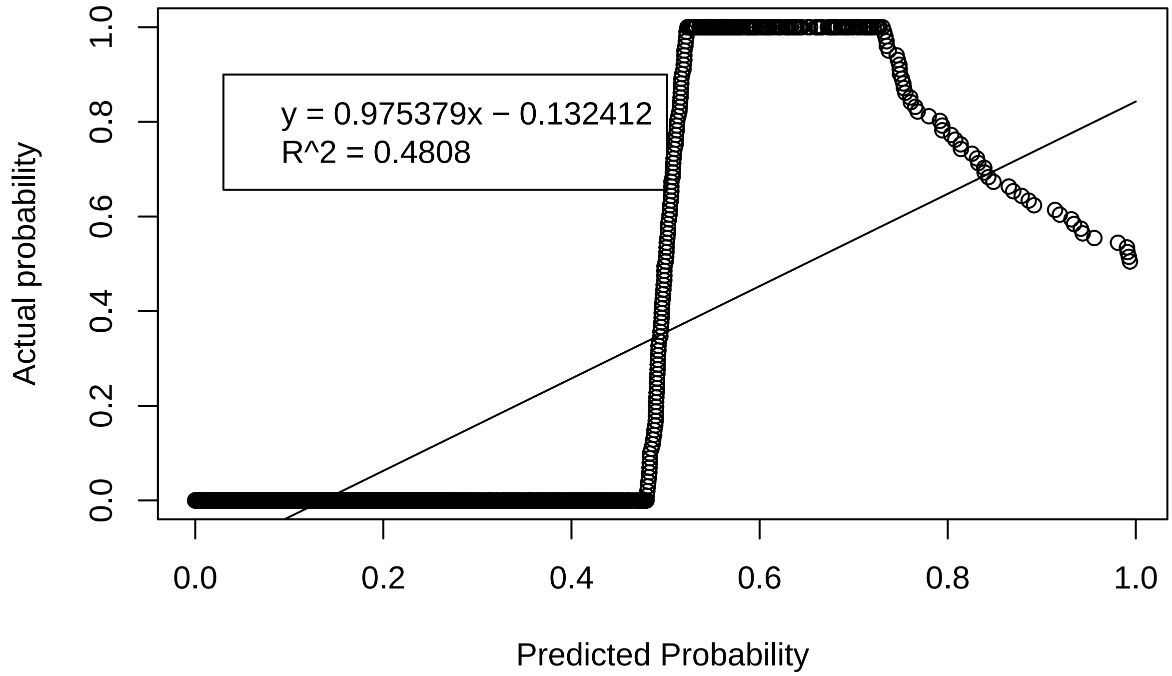
```
## [1] 0.3624596
```

```
# [1] 0.3624596



# Part 2.2.3: Use SSM(Sorting Smoothing Method) to estimate
# real probability

# 1. order the valid data according to predictive probability
valid_norm_sort = valid_norm[order(PredScoreV), ]
# 2. use SSM formula to evaluate actural probability 'Pi', we
# choose n =50 according to the paper
VALIDSIZE = dim(valid_norm)[1]
n = 50
actural_p_valid = rep(0, VALIDSIZE)
pred_valid = round(valid_norm_sort$PredScoreV)
pred_valid = prepend(pred_valid, rep(0, n), before = 1)
pred_valid = append(pred_valid, rep(0, n))
for (i in 1:VALIDSIZE) {
    actural_p_valid[i] = sum(pred_valid[i:(i + 2 * n)])/(2 *
        n + 1)
}
```

```r
valid_norm_sort = data.frame(valid_norm_sort, actural_p_valid)


# png('Scatter plot diagram of KNN.png')
plot(valid_norm_sort$PredScoreV, valid_norm_sort$actural_p_valid,
    xlab = "Predicted Probability", ylab = "Actual probability")

yy = valid_norm_sort$actural_p_valid
xx = valid_norm_sort$PredScoreV
actual_fit = lm(yy ~ xx)

xx = seq(0, 1:0.1)
yy = predict(actual_fit, data.frame((xx)))
lines(xx, yy)

summary(actual_fit)
```

```
##
## Call:
## lm(formula = yy ~ xx)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.33555 -0.06601 -0.00478  0.05829  0.62198
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.132412   0.003114  -42.52   <2e-16 ***
## xx           0.975379   0.013086   74.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1495 on 5998 degrees of freedom
## Multiple R-squared:  0.4808, Adjusted R-squared:  0.4808
## F-statistic:  5555 on 1 and 5998 DF,  p-value: < 2.2e-16
```

```r
legend(0.03, 0.9, legend = c("y = 0.975379x - 0.132412", "R^2 = 0.4808"))
```




```
# dev.off()
```

6. Modeling and analysis with AdaBoost

```
#################################

# 3. AdaBoost #

#################################


#########################

# 3.1 AB Train #

#########################
scale01 <- function(x) {
    (x - min(x))/(max(x) - min(x))
}

train_norm = train

for (name in names(train)) {
    if (name != "default.payment.next.month") {
        train_norm[name] <- scale01(train_norm[name])
    }
}

# Part 3.1.1: train data with AB library(fastAdaboost) ab <-
# adaboost(default.payment.next.month ~ ., data = train, 10)
# summary(ab)
```

```r
# Another package (slow - train time: takes ~10 min, but has
# more attachments, shows confusion matrix and error)
# library(adabag)

train_norm$default.payment.next.month <- as.factor(train_norm$default.payment.next.month)
abt <- boosting(default.payment.next.month ~ ., data = train_norm)
# Variable importance with respect to most important variable
sort(abt$importance/max(abt$importance), decreasing = TRUE)
```

```
##        PAY_0     PAY_AMT2    LIMIT_BAL        PAY_4        PAY_5
## 1.0000000000 0.0589186658 0.0406698686 0.0273924577 0.0272238375
##     PAY_AMT1    BILL_AMT1        PAY_3     PAY_AMT4    BILL_AMT2
## 0.0256137520 0.0205879839 0.0127767951 0.0120870812 0.0100350259
##    EDUCATION     PAY_AMT3    BILL_AMT3     MARRIAGE     PAY_AMT5
## 0.0080874240 0.0047424856 0.0045792388 0.0037993153 0.0033236421
##        PAY_6        PAY_2          AGE    BILL_AMT4     PAY_AMT6
## 0.0031584953 0.0027917771 0.0027301225 0.0021569994 0.0020150974
##    BILL_AMT6    BILL_AMT5          SEX
## 0.0018726095 0.0011605880 0.0009311602
```

```r
# PAY_0 PAY_AMT3 PAY_AMT1 PAY_5 LIMIT_BAL 1.0000000000
# 0.0668812095 0.0469750718 0.0464909929 0.0384774264 PAY_6
# BILL_AMT1 PAY_3 EDUCATION PAY_AMT2 0.0315244971
# 0.0312812159 0.0237877992 0.0180606923 0.0165200485
# PAY_AMT4 PAY_AMT5 BILL_AMT4 BILL_AMT3 MARRIAGE 0.0067207012
# 0.0061924826 0.0057661431 0.0051095984 0.0047488679
# BILL_AMT2 PAY_2 PAY_AMT6 SEX AGE 0.0038661903 0.0025736510
# 0.0022537677 0.0021115763 0.0016453458 PAY_4 BILL_AMT6
# BILL_AMT5 0.0014192298 0.0012443866 0.0008954529

tt = table(pred = abt$class, actual = train$default.payment.next.month)
error_train = 1 - sum(diag(tt))/sum(tt)
error_train
```

```
## [1] 0.1815
```

```r
# [1] 0.1779583



scale01 <- function(x) {
    (x - min(x))/(max(x) - min(x))
}

train_norm = train

for (name in names(train)) {
    if (name != "default.payment.next.month") {
        train_norm[name] <- scale01(train_norm[name])
    }
}
```

```r
pred = predict(abt, train_norm)

PredABLabel = data.frame(as.numeric(pred$class))
names(PredABLabel) <- "PredABLabel"

# sum(predict(abt, train_norm, type='prob')$prob[,2] > 0.5) #
# [1] 2900

PredABScore = predict(abt, train_norm, type = "prob")$prob[,
    2]
length(PredABScore)
```

```
## [1] 24000
```

```r
train_norm = data.frame(train_norm, PredABLabel, PredABScore)
head(train_norm)
```

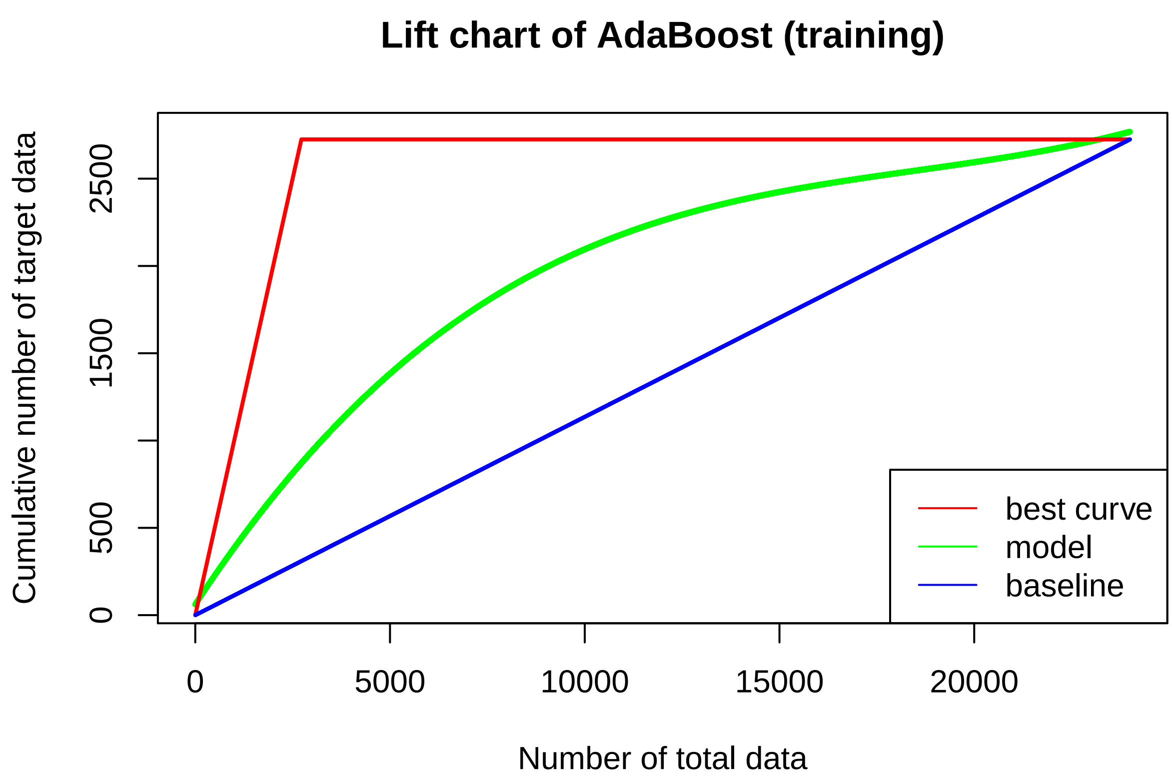
```
##        LIMIT_BAL SEX EDUCATION  MARRIAGE        AGE PAY_0 PAY_2 PAY_3
## 10085 0.33333333   0 0.3333333 0.3333333 0.24137931   0.2   0.2   0.2
## 13912 0.44444444   0 0.1666667 0.3333333 0.46551724   0.0   0.0   0.0
## 1818  0.17171717   1 0.3333333 0.3333333 0.39655172   0.1   0.1   0.1
## 5923  0.07070707   0 0.1666667 0.6666667 0.13793103   0.1   0.2   0.2
## 14228 0.10101010   0 0.3333333 0.6666667 0.13793103   0.1   0.1   0.1
## 9030  0.06060606   1 0.1666667 0.6666667 0.05172414   0.3   0.4   0.2
##       PAY_4 PAY_5 PAY_6 BILL_AMT1  BILL_AMT2  BILL_AMT3  BILL_AMT4
## 10085   0.2   0.2   0.2 0.3995368 0.15054835 0.13689384 0.17750791
## 13912   0.0   0.0   0.0 0.1571358 0.14230982 0.10688812 0.10625951
## 1818    0.1   0.1   0.1 0.1391614 0.06698250 0.08777156 0.08359783
## 5923    0.2   0.2   0.2 0.1541201 0.08302680 0.09492668 0.09737286
## 14228   0.1   0.1   0.1 0.1481147 0.06906278 0.08767274 0.08793426
## 9030    0.2   0.4   0.2 0.2018698 0.12738918 0.12158324 0.11479361
##        BILL_AMT5 BILL_AMT6     PAY_AMT1    PAY_AMT2    PAY_AMT3
## 10085 0.16949346 0.2527942 0.0044691100 0.002795295 0.003529976
## 13912 0.10139761 0.1898310 0.0917987710 0.022215704 0.024641757
## 1818  0.08293960 0.1804410 0.0009192355 0.001662452 0.000000000
## 5923  0.09373379 0.1884959 0.0034514259 0.001192216 0.001126066
## 14228 0.09027025 0.1845052 0.0034285309 0.001436240 0.004708495
## 9030  0.11022256 0.2039352 0.0000000000 0.001074063 0.002790054
##          PAY_AMT4    PAY_AMT5     PAY_AMT6 default.payment.next.month
## 10085 0.005167472 0.007272659 0.0060416217                          0
## 13912 0.033697262 0.031268683 0.1678640200                          0
## 1818  0.003721417 0.005959735 0.0004464066                          0
## 5923  0.003228663 0.001188665 0.0049312799                          0
## 14228 0.015626409 0.016299009 0.0095296463                          0
## 9030  0.000000000 0.004689013 0.0028373302                          0
##       PredABLabel PredABScore
## 10085           0   0.2559472
## 13912           0   0.1218566
## 1818            0   0.3130939
## 5923            0   0.2522802
## 14228           0   0.2431939
## 9030            0   0.4817565
```

```r
gtt = gains(actual = as.numeric(train_norm$default.payment.next.month),
    predicted = train_norm$PredABScore, optimal = TRUE)
```

```r
cpt_y = gtt$cume.pct.of.total
cpt_x = gtt$depth

predicted = table(train_norm$PredABLabel)[2]
predicted
```

```
##    1
## 2725
```

```r
# 1 2900
xx = cpt_x/100 * 24000
yy = cpt_y * predicted
# plot(xx,yy)

xx = prepend(xx, 0, before = 1)
yy = prepend(yy, 0, before = 1)
fit = lm(yy ~ poly(xx, 3, raw = TRUE))

xx = 0:24000
model_yy = predict(fit, data.frame(xx))


# png('/Users/qinqingao/Desktop/Columbia/Courses/Fall
# 2018/EECS 6690/Project/figs/AB_lift_chart_test.png')
plot(xx, model_yy, col = "green", xlab = "Number of total data",
    ylab = "Cumulative number of target data", type = "l", lwd = 3)

best_yy = rep(predicted, 24001)
for (i in 0:predicted) {
    best_yy[i + 1] = i
}

lines(xx, best_yy, col = "red", lwd = 2)

base_yy = predicted/24000 * xx
lines(xx, base_yy, col = "blue", lwd = 2)

legend("bottomright", legend = c("best curve", "model", "baseline"),
    col = c("red", "green", "blue"), lwd = c(1, 1, 1), cex = 1)
title("Lift chart of AdaBoost (training)")
```




```
# dev.off()


# Calculate area ratio
a1v = sum(model_yy - base_yy)
a2v = sum(best_yy - base_yy)
a1v/a2v
```

```
## [1] 0.5100937
```

```
# [1] 0.5152811






########################

# 3.2 AB Test #

########################


# Part 3.2.1: test data with AB
predv = predict(abt, valid_norm)
# predv $confusion Observed Class Predicted Class 0 1 0 4468
# 852 1 215 465

# $error [1] 0.1778333
```

```r
# Part 3.2.2: plot test lift curve for AB
PredABLabelV = data.frame(as.numeric(predv$class))
names(PredABLabelV) <- "PredABLabelV"

# sum(predict(abt, valid_norm, type='prob')$prob[,2] > 0.5) #
# [1] 680

PredABScoreV = predict(abt, valid_norm, type = "prob")$prob[,
    2]
length(PredABScoreV)
```

```
## [1] 6000
```

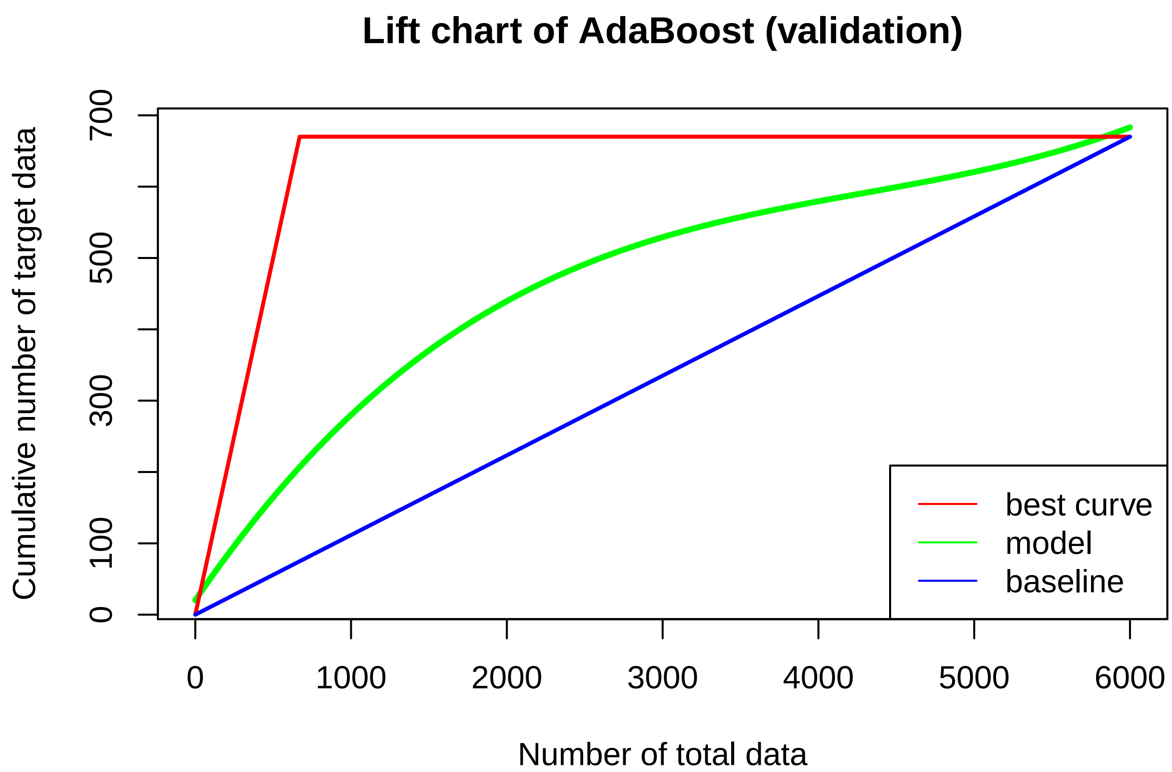
```r
valid_norm = data.frame(valid_norm, PredABLabelV, PredABScoreV)
head(valid_norm)
```

```
##     LIMIT_BAL SEX EDUCATION  MARRIAGE        AGE PAY_0     PAY_2 PAY_3
## 3  0.10126582   1 0.3333333 0.6666667 0.25000000   0.2 0.2222222   0.2
## 13 0.78481013   1 0.3333333 0.6666667 0.38461538   0.1 0.2222222   0.1
## 19 0.44303797   1 0.1666667 0.3333333 0.53846154   0.3 0.0000000   0.0
## 23 0.07594937   1 0.3333333 0.6666667 0.09615385   0.4 0.2222222   0.2
## 27 0.06329114   0 0.1666667 0.6666667 0.11538462   0.3 0.0000000   0.1
## 30 0.05063291   0 0.1666667 0.6666667 0.09615385   0.2 0.2222222   0.2
##        PAY_4     PAY_5     PAY_6 BILL_AMT1 BILL_AMT2  BILL_AMT3 BILL_AMT4
## 3  0.2222222 0.2222222 0.2222222 0.2499474 0.1225773 0.04804794 0.2307891
## 13 0.1111111 0.1111111 0.1111111 0.2280060 0.1112639 0.03653854 0.2209844
## 19 0.0000000 0.0000000 0.0000000 0.2124346 0.1014942 0.02594057 0.2128461
## 23 0.4444444 0.4444444 0.4444444 0.2651481 0.1652906 0.09934374 0.2679432
## 27 0.1111111 0.1111111 0.1111111 0.2122947 0.1008554 0.02636286 0.2127748
## 30 0.2222222 0.2222222 0.2222222 0.2321013 0.1264070 0.05446704 0.2352664
##     BILL_AMT5 BILL_AMT6    PAY_AMT1    PAY_AMT2    PAY_AMT3     PAY_AMT4
## 3  0.07356351 0.3910328 0.003005941 0.002584140 0.001967617 0.0018907273
## 13 0.05988360 0.3770728 0.001980198 0.011197938 0.012789510 0.0122897275
## 19 0.04935811 0.3739129 0.000000000 0.000000000 0.000000000 0.0000000000
## 23 0.12531172 0.4245734 0.003974257 0.006170925 0.000000000 0.0068085090
## 27 0.04956376 0.3737048 0.000000000 0.001722760 0.000000000 0.0009453637
## 30 0.07911288 0.3864646 0.002970297 0.002584140 0.001967617 0.0018907273
##       PAY_AMT5    PAY_AMT6 default.payment.next.month PredKNNLabelV
## 3  0.003472439 0.011848341                          0             0
## 13 0.009965901 0.000000000                          0             0
## 19 0.000000000 0.000000000                          0             0
## 23 0.000000000 0.004312796                          1             1
## 27 0.000000000 0.002369668                          1             0
## 30 0.005555903 0.000000000                          0             0
##    PredKNNScoreV PredLabelV PredScoreV PredABLabelV PredABScoreV
## 3           0.08          0 0.18060338            0    0.2585708
## 13          0.06          0 0.05501201            0    0.1362434
## 19          0.24          0 0.24663611            0    0.3541490
## 23          0.55          0 0.38154035            1    0.7429356
## 27          0.22          0 0.28146494            0    0.3577918
## 30          0.09          0 0.21350530            0    0.2696382
```

```r
gtv = gains(actual = as.numeric(valid_norm$default.payment.next.month),
    predicted = valid_norm$PredABScoreV, optimal = TRUE)
cpv_y = gtv$cume.pct.of.total
cpv_x = gtv$depth

predictedV = table(valid_norm$PredABLabelV)[2]
predictedV
```

```
##   1
## 670
```

```r
# 1 680
xx = cpv_x/100 * 6000
yy = cpv_y * predictedV
# plot(xx,yy)

xx = prepend(xx, 0, before = 1)
yy = prepend(yy, 0, before = 1)
fit = lm(yy ~ poly(xx, 3, raw = TRUE))

xx = 0:6000
model_yy = predict(fit, data.frame(xx))


# png('/Users/qinqingao/Desktop/Columbia/Courses/Fall
# 2018/EECS 6690/Project/figs/AB_lift_chart_test.png')
plot(xx, model_yy, col = "green", xlab = "Number of total data",
    ylab = "Cumulative number of target data", type = "l", lwd = 3)

best_yy = rep(predictedV, 6001)
for (i in 0:predictedV) {
    best_yy[i + 1] = i
}

lines(xx, best_yy, col = "red", lwd = 2)

base_yy = predictedV/6000 * xx
lines(xx, base_yy, col = "blue", lwd = 2)

legend("bottomright", legend = c("best curve", "model", "baseline"),
    col = c("red", "green", "blue"), lwd = c(1, 1, 1), cex = 1)
title("Lift chart of AdaBoost (validation)")
```




```
# dev.off()


# Calculate area ratio
a1v = sum(model_yy - base_yy)
a2v = sum(best_yy - base_yy)
a1v/a2v
```

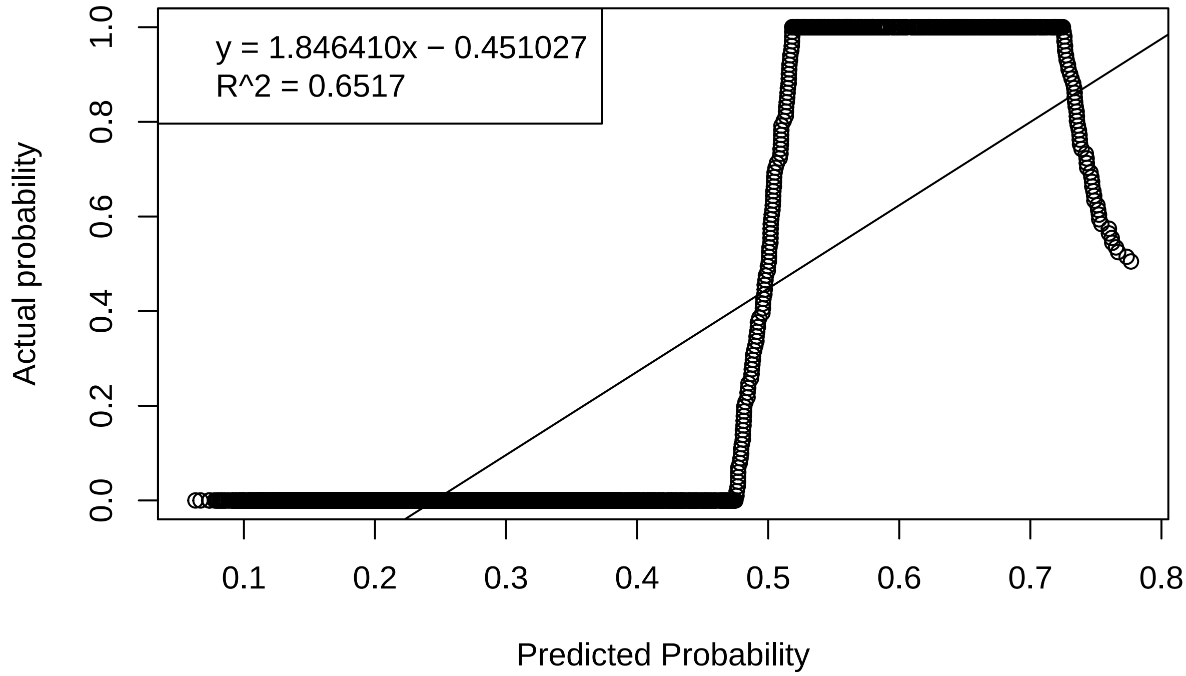
```
## [1] 0.453706
```

```
# [1] 0.4595666



# Part 3.2.3: Use SSM(Sorting Smoothing Method) to estimate
# real probability

# 1. order the valid data according to predictive probability
valid_norm_sort = valid_norm[order(PredABScoreV), ]
# 2. use SSM formula to evaluate actural probability 'Pi', we
# choose n =50 according to the paper
VALIDSIZE = dim(valid_norm)[1]

n = 50
actural_p_valid = rep(0, VALIDSIZE)
pred_valid = round(valid_norm_sort$PredABScoreV)
pred_valid = prepend(pred_valid, rep(0, n), before = 1)
pred_valid = append(pred_valid, rep(0, n))

for (i in 1:VALIDSIZE) {
```

```r
    actural_p_valid[i] = sum(pred_valid[i:(i + 2 * n)])/(2 *
        n + 1)
}
valid_norm_sort = data.frame(valid_norm_sort, actural_p_valid)


# png('/Users/qinqingao/Desktop/Columbia/Courses/Fall
# 2018/EECS 6690/Project/figs/Scatter plot diagram of
# AB.png')
plot(valid_norm_sort$PredABScoreV, valid_norm_sort$actural_p_valid,
    xlab = "Predicted Probability", ylab = "Actual probability")

yy = valid_norm_sort$actural_p_valid
xx = valid_norm_sort$PredABScoreV
actual_fit = lm(yy ~ xx)

xx = seq(0, 1:0.1)
yy = predict(actual_fit, data.frame((xx)))
lines(xx, yy)

summary(actual_fit)
```

```
##
## Call:
## lm(formula = yy ~ xx)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42957 -0.12605 -0.03024  0.12008  0.51989
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.43160    0.00594  -72.66   <2e-16 ***
## xx           1.75881    0.01763   99.76   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1874 on 5998 degrees of freedom
## Multiple R-squared:  0.6239, Adjusted R-squared:  0.6239
## F-statistic:  9951 on 1 and 5998 DF,  p-value: < 2.2e-16
```

```r
legend("topleft", legend = c("y = 1.846410x - 0.451027", "R^2 = 0.6517"))
```




```
# dev.off()
```

7. Modeling and analysis with XGBoost

```
##################################

# 4. XGBoost #

##################################

# normalizing numerical variables

scale01 <- function(x) {
    (x - min(x))/(max(x) - min(x))
}

train_norm = train
valid_norm = test

for (name in names(train)) {
    if (name != "default.payment.next.month") {
        train_norm[name] <- scale01(train_norm[name])
        valid_norm[name] <- scale01(valid_norm[name])
    }
}

# convert data frame to data table
setDT(train_norm)
setDT(valid_norm)

# new train, new test without target variable
```

```r
labels <- as.numeric(train_norm$default.payment.next.month)
ts_label <- as.numeric(valid_norm$default.payment.next.month)
new_tr <- model.matrix(~. + 0, data = train_norm[, -c("default.payment.next.month"),
    with = F])
new_ts <- model.matrix(~. + 0, data = valid_norm[, -c("default.payment.next.month"),
    with = F])


# convert data table into a matrix, to use XGBoost
dtrain <- xgb.DMatrix(data = new_tr, label = labels)
dtest <- xgb.DMatrix(data = new_ts, label = ts_label)


#########################

# 4.1 XGB Train #

#########################


# Part 4.1.1: train data with XB
params <- list(booster = "gbtree", objective = "binary:logistic",
    eta = 0.1, gamma = 0, max_depth = 6, min_child_weight = 1,
    subsample = 1, colsample_bytree = 1)

# Cross Validate to choose best nrounds, and CV error
xgbcv <- xgb.cv(params = params, data = dtrain, nrounds = 100,
    nfold = 5, showsd = T, stratified = T, print.every.n = 10,
    early.stop.round = 50, maximize = F)
```

```
## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").

## Warning: 'early.stop.round' is deprecated.
## Use 'early_stopping_rounds' instead.
## See help("Deprecated") and help("xgboost-deprecated").

## [1]  train-error:0.172625+0.001123   test-error:0.182917+0.005005
## Multiple eval metrics are present. Will use test_error for early stopping.
## Will train until test_error hasn't improved in 50 rounds.
##
## [11] train-error:0.167250+0.000643   test-error:0.179958+0.004801
## [21] train-error:0.164833+0.000720   test-error:0.179250+0.004287
## [31] train-error:0.161948+0.000791   test-error:0.178833+0.003629
## [41] train-error:0.159396+0.001205   test-error:0.178833+0.004143
## [51] train-error:0.157521+0.001277   test-error:0.178708+0.004032
## [61] train-error:0.155844+0.001012   test-error:0.179042+0.004037
## [71] train-error:0.153771+0.001457   test-error:0.179083+0.004128
## [81] train-error:0.152406+0.001493   test-error:0.179333+0.004496
## [91] train-error:0.150510+0.001877   test-error:0.179375+0.004290
## [100]    train-error:0.149313+0.001956   test-error:0.180125+0.004098
```

```r
# xgbcv
```

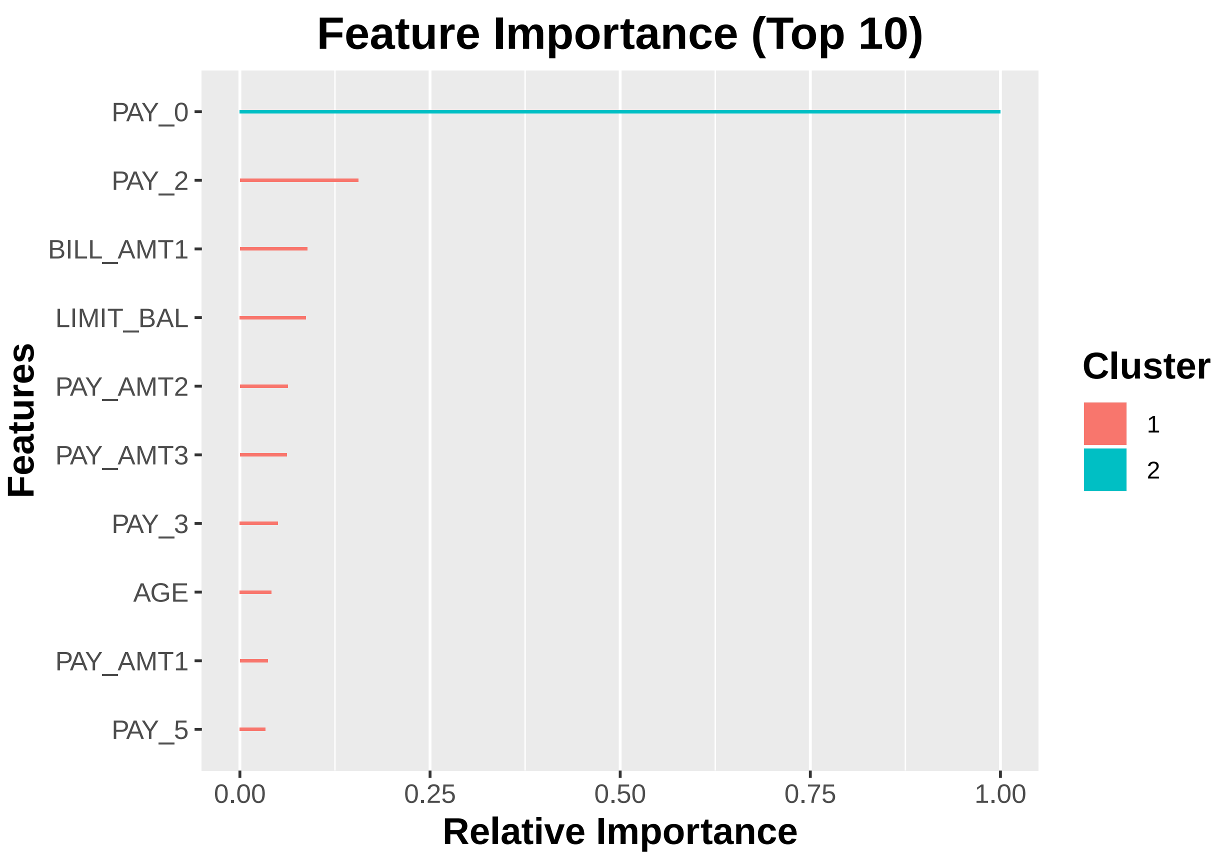
```r
xgbcv$best_iteration
```

```
## [1] 54
```

```r
# [1] 28

# xgbcv Best iteration: iter train_error_mean train_error_std
# test_error_mean test_error_std 28 0.1625418 0.0009313005
# 0.1784166 0.005228473

# test_error_mean 0.1784166


# XGBoost model training, nrounds = 28
xgb1 <- xgb.train(params = params, data = dtrain, nrounds = 28,
    watchlist = list(val = dtest, train = dtrain), print.every.n = 10,
    early.stop.round = 50, maximize = F, eval_metric = "error")
```

```
## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").

## Warning: 'early.stop.round' is deprecated.
## Use 'early_stopping_rounds' instead.
## See help("Deprecated") and help("xgboost-deprecated").

## [1]  val-error:0.194167  train-error:0.172708
## Multiple eval metrics are present. Will use train_error for early stopping.
## Will train until train_error hasn't improved in 50 rounds.
##
## [11] val-error:0.173333  train-error:0.168375
## [21] val-error:0.174333  train-error:0.166833
## [28] val-error:0.176333  train-error:0.165792
```

```r
# xgb1 niter: 28 best_iteration : 26 best_ntreelimit : 26
# best_score : 0.165792


# Plot Top 10 Feature Importance
mat <- xgb.importance(feature_names = colnames(new_tr), model = xgb1)
xgb.ggplot.importance(importance_matrix = mat[1:10], rel_to_first = TRUE) +
    labs(title = "Feature Importance (Top 10)", x = "Features",
        y = "Relative Importance") + theme(plot.title = element_text(hjust = 0.5),
    title = element_text(size = 14, face = "bold"), axis.text.x = element_text(size = 10),
    axis.text.y = element_text(size = 10))
```




```r
# Part 4.1.2: plot test lift curve for XGBoost
pred = predict(xgb1, dtrain)

PredXBLabel = ifelse(pred > 0.5, 1, 0)
confusionMatrix(table(PredXBLabel, labels))
```

```
## Confusion Matrix and Statistics
##
##            labels
## PredXBLabel     0     1
##           0 17904  3202
##           1   777  2117
##
##                Accuracy : 0.8342
##                  95% CI : (0.8294, 0.8389)
##     No Information Rate : 0.7784
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4258
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9584
##             Specificity : 0.3980
##          Pos Pred Value : 0.8483
##          Neg Pred Value : 0.7315
##              Prevalence : 0.7784
##          Detection Rate : 0.7460
```

```
##    Detection Prevalence : 0.8794
##       Balanced Accuracy : 0.6782
##
##        'Positive' Class : 0
##
```

```r
err <- 1 - confusionMatrix(table(PredXBLabel, labels))$overall["Accuracy"]
err
```

```
##  Accuracy
## 0.1657917
```

```r
# 0.1657917

PredXBLabel = data.frame(as.numeric(PredXBLabel))
names(PredXBLabel) <- "PredXBLabel"

PredXBScore = pred
length(PredXBScore)
```

```
## [1] 24000
```

```r
train_norm = data.frame(train_norm, PredXBLabel, PredXBScore)
head(train_norm)
```

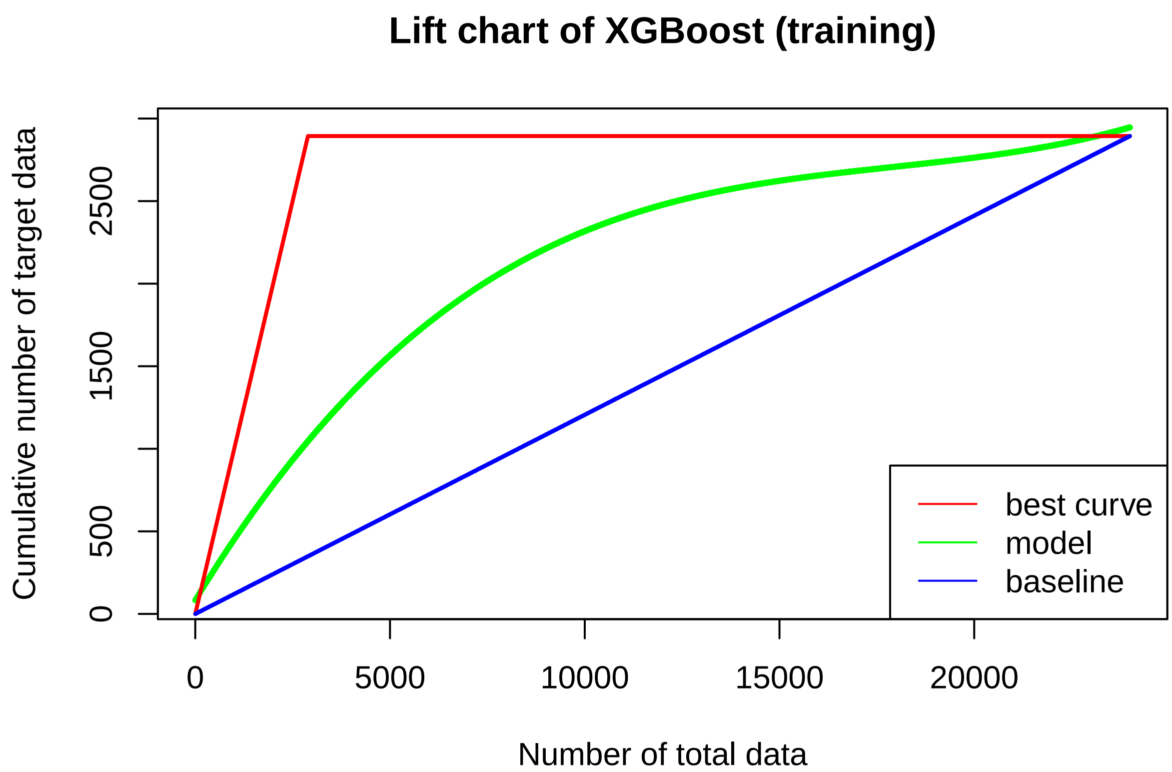
```
##    LIMIT_BAL SEX EDUCATION  MARRIAGE        AGE PAY_0 PAY_2 PAY_3 PAY_4
## 1 0.33333333   0 0.3333333 0.3333333 0.24137931   0.2   0.2   0.2   0.2
## 2 0.44444444   0 0.1666667 0.3333333 0.46551724   0.0   0.0   0.0   0.0
## 3 0.17171717   1 0.3333333 0.3333333 0.39655172   0.1   0.1   0.1   0.1
## 4 0.07070707   0 0.1666667 0.6666667 0.13793103   0.1   0.2   0.2   0.2
## 5 0.10101010   0 0.3333333 0.6666667 0.13793103   0.1   0.1   0.1   0.1
## 6 0.06060606   1 0.1666667 0.6666667 0.05172414   0.3   0.4   0.2   0.2
##   PAY_5 PAY_6 BILL_AMT1  BILL_AMT2  BILL_AMT3  BILL_AMT4  BILL_AMT5
## 1   0.2   0.2 0.3995368 0.15054835 0.13689384 0.17750791 0.16949346
## 2   0.0   0.0 0.1571358 0.14230982 0.10688812 0.10625951 0.10139761
## 3   0.1   0.1 0.1391614 0.06698250 0.08777156 0.08359783 0.08293960
## 4   0.2   0.2 0.1541201 0.08302680 0.09492668 0.09737286 0.09373379
## 5   0.1   0.1 0.1481147 0.06906278 0.08767274 0.08793426 0.09027025
## 6   0.4   0.2 0.2018698 0.12738918 0.12158324 0.11479361 0.11022256
##   BILL_AMT6     PAY_AMT1    PAY_AMT2    PAY_AMT3    PAY_AMT4    PAY_AMT5
## 1 0.2527942 0.0044691100 0.002795295 0.003529976 0.005167472 0.007272659
## 2 0.1898310 0.0917987710 0.022215704 0.024641757 0.033697262 0.031268683
## 3 0.1804410 0.0009192355 0.001662452 0.000000000 0.003721417 0.005959735
## 4 0.1884959 0.0034514259 0.001192216 0.001126066 0.003228663 0.001188665
## 5 0.1845052 0.0034285309 0.001436240 0.004708495 0.015626409 0.016299009
## 6 0.2039352 0.0000000000 0.001074063 0.002790054 0.000000000 0.004689013
##       PAY_AMT6 default.payment.next.month PredXBLabel PredXBScore
## 1 0.0060416217                          0           0  0.13959493
## 2 0.1678640200                          0           0  0.07243134
## 3 0.0004464066                          0           0  0.17713341
## 4 0.0049312799                          0           0  0.12394150
## 5 0.0095296463                          0           0  0.12678835
## 6 0.0028373302                          0           0  0.39022201
```

```r
gtt = gains(actual = as.numeric(train_norm$default.payment.next.month),
    predicted = train_norm$PredXBScore, optimal = TRUE)
cpt_y = gtt$cume.pct.of.total
```

```r
cpt_x = gtt$depth

predicted = table(train_norm$PredXBLabel)[2]
predicted
```

```
##    1
## 2894
```

```r
# 1 2894
xx = cpt_x/100 * 24000
yy = cpt_y * predicted
# plot(xx,yy)

xx = prepend(xx, 0, before = 1)
yy = prepend(yy, 0, before = 1)
fit = lm(yy ~ poly(xx, 3, raw = TRUE))

xx = 0:24000
model_yy = predict(fit, data.frame(xx))


# png('/Users/qinqingao/Desktop/Columbia/Courses/Fall
# 2018/EECS 6690/Project/figs/XB_lift_chart_train.png')
plot(xx, model_yy, col = "green", xlab = "Number of total data",
    ylab = "Cumulative number of target data", type = "l", lwd = 3)

best_yy = rep(predicted, 24001)
for (i in 0:predicted) {
    best_yy[i + 1] = i
}

lines(xx, best_yy, col = "red", lwd = 2)

base_yy = predicted/24000 * xx
lines(xx, base_yy, col = "blue", lwd = 2)

legend("bottomright", legend = c("best curve", "model", "baseline"),
    col = c("red", "green", "blue"), lwd = c(1, 1, 1), cex = 1)
title("Lift chart of XGBoost (training)")
```




```r
# dev.off()


# Calculate area ratio
a1v = sum(model_yy - base_yy)
a2v = sum(best_yy - base_yy)
a1v/a2v
```

```
## [1] 0.5578075
```

```r
# [1] 0.5578075





#########################

# 4.2 XB Test #

#########################


# Part 4.2.1: test data with XB model prediction on dtest
predv = predict(xgb1, dtest)

PredXBLabelV = ifelse(predv > 0.5, 1, 0)
confusionMatrix(table(PredXBLabelV, ts_label))
```

```
## Confusion Matrix and Statistics
##
```

```
##              ts_label
## PredXBLabelV    0    1
##            0 4427  799
##            1  256  518
##
##                Accuracy : 0.8242
##                  95% CI : (0.8143, 0.8337)
##     No Information Rate : 0.7805
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3976
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9453
##             Specificity : 0.3933
##          Pos Pred Value : 0.8471
##          Neg Pred Value : 0.6693
##              Prevalence : 0.7805
##          Detection Rate : 0.7378
##    Detection Prevalence : 0.8710
##       Balanced Accuracy : 0.6693
##
##        'Positive' Class : 0
##
```

```r
errv <- 1 - confusionMatrix(table(PredXBLabelV, ts_label))$overall["Accuracy"]
errv
```

```
##  Accuracy
## 0.1758333
```

```r
# 0.1758333

# Confusion Matrix and Statistics

# ts_label PredXBLabelV 0 1 0 4427 799 1 256 518

# Accuracy : 0.8242 95% CI : (0.8143, 0.8337) No Information
# Rate : 0.7805 P-Value [Acc > NIR] : < 2.2e-16

# Kappa : 0.3976 Mcnemar's Test P-Value : < 2.2e-16

# Sensitivity : 0.9453 Specificity : 0.3933 Pos Pred Value :
# 0.8471 Neg Pred Value : 0.6693 Prevalence : 0.7805
# Detection Rate : 0.7378 Detection Prevalence : 0.8710
# Balanced Accuracy : 0.6693

# 'Positive' Class : 0

PredXBLabelV = data.frame(as.numeric(PredXBLabelV))
names(PredXBLabelV) <- "PredXBLabelV"


PredXBScoreV = predv
length(PredXBScoreV)
```

```
## [1] 6000
```

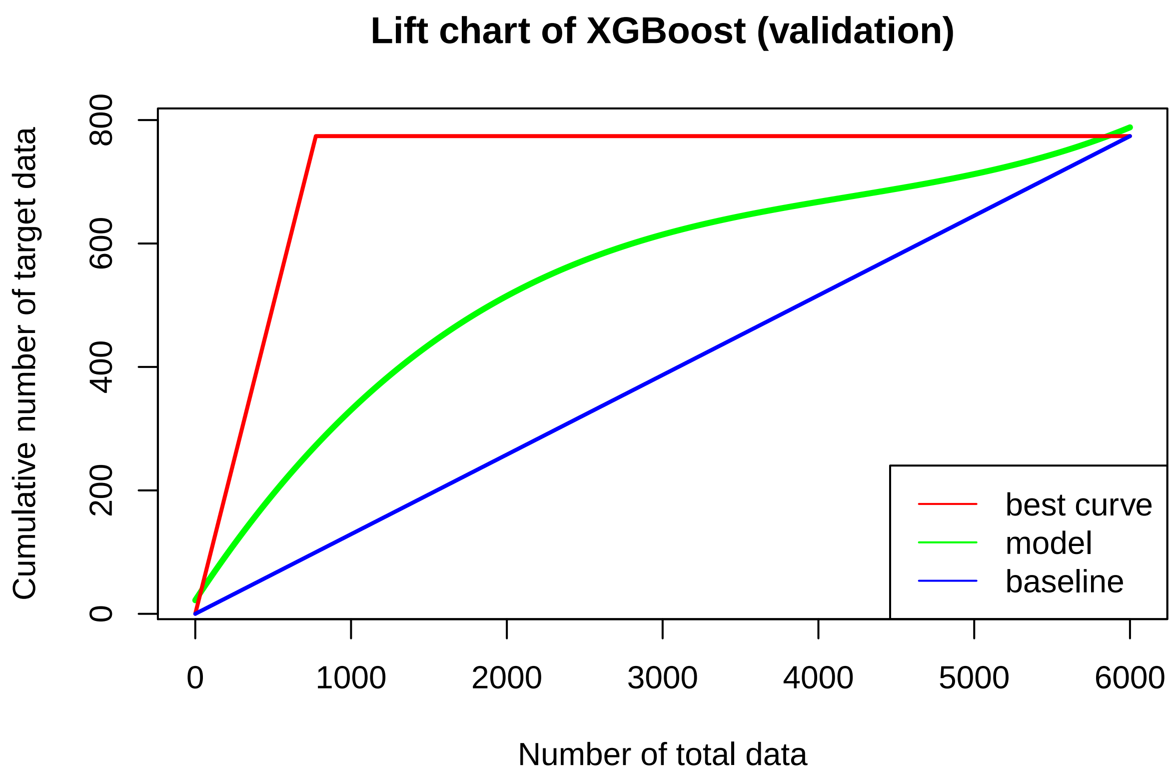
```r
# [1] 6000


valid_norm = data.frame(valid_norm, PredXBLabelV, PredXBScoreV)
head(valid_norm)
```

```
##    LIMIT_BAL SEX EDUCATION  MARRIAGE        AGE PAY_0     PAY_2 PAY_3
## 1 0.10126582   1 0.3333333 0.6666667 0.25000000   0.2 0.2222222   0.2
## 2 0.78481013   1 0.3333333 0.6666667 0.38461538   0.1 0.2222222   0.1
## 3 0.44303797   1 0.1666667 0.3333333 0.53846154   0.3 0.0000000   0.0
## 4 0.07594937   1 0.3333333 0.6666667 0.09615385   0.4 0.2222222   0.2
## 5 0.06329114   0 0.1666667 0.6666667 0.11538462   0.3 0.0000000   0.1
## 6 0.05063291   0 0.1666667 0.6666667 0.09615385   0.2 0.2222222   0.2
##       PAY_4     PAY_5     PAY_6 BILL_AMT1 BILL_AMT2  BILL_AMT3 BILL_AMT4
## 1 0.2222222 0.2222222 0.2222222 0.2499474 0.1225773 0.04804794 0.2307891
## 2 0.1111111 0.1111111 0.1111111 0.2280060 0.1112639 0.03653854 0.2209844
## 3 0.0000000 0.0000000 0.0000000 0.2124346 0.1014942 0.02594057 0.2128461
## 4 0.4444444 0.4444444 0.4444444 0.2651481 0.1652906 0.09934374 0.2679432
## 5 0.1111111 0.1111111 0.1111111 0.2122947 0.1008554 0.02636286 0.2127748
## 6 0.2222222 0.2222222 0.2222222 0.2321013 0.1264070 0.05446704 0.2352664
##    BILL_AMT5 BILL_AMT6    PAY_AMT1    PAY_AMT2    PAY_AMT3     PAY_AMT4
## 1 0.07356351 0.3910328 0.003005941 0.002584140 0.001967617 0.0018907273
## 2 0.05988360 0.3770728 0.001980198 0.011197938 0.012789510 0.0122897275
## 3 0.04935811 0.3739129 0.000000000 0.000000000 0.000000000 0.0000000000
## 4 0.12531172 0.4245734 0.003974257 0.006170925 0.000000000 0.0068085090
## 5 0.04956376 0.3737048 0.000000000 0.001722760 0.000000000 0.0009453637
## 6 0.07911288 0.3864646 0.002970297 0.002584140 0.001967617 0.0018907273
##      PAY_AMT5    PAY_AMT6 default.payment.next.month PredXBLabelV
## 1 0.003472439 0.011848341                          0            0
## 2 0.009965901 0.000000000                          0            0
## 3 0.000000000 0.000000000                          0            0
## 4 0.000000000 0.004312796                          1            1
## 5 0.000000000 0.002369668                          1            0
## 6 0.005555903 0.000000000                          0            0
##   PredXBScoreV
## 1   0.13080588
## 2   0.08169384
## 3   0.44230592
## 4   0.78582758
## 5   0.30772933
## 6   0.16233349
```

```r
# Part 4.2.2: plot test lift curve for XB
gtv = gains(actual = as.numeric(valid_norm$default.payment.next.month),
    predicted = valid_norm$PredXBScoreV, optimal = TRUE)
cpv_y = gtv$cume.pct.of.total
cpv_x = gtv$depth

predictedV = table(valid_norm$PredXBLabelV)[2]
predictedV
```

```
##   1
## 774
```

```r
# 1 774
xx = cpv_x/100 * 6000
yy = cpv_y * predictedV
# plot(xx,yy)

xx = prepend(xx, 0, before = 1)
yy = prepend(yy, 0, before = 1)
fit = lm(yy ~ poly(xx, 3, raw = TRUE))

xx = 0:6000
model_yy = predict(fit, data.frame(xx))


# png('/Users/qinqingao/Desktop/Columbia/Courses/Fall
# 2018/EECS 6690/Project/figs/XB_lift_chart_test.png')
plot(xx, model_yy, col = "green", xlab = "Number of total data",
    ylab = "Cumulative number of target data", type = "l", lwd = 3)

best_yy = rep(predictedV, 6001)
for (i in 0:predictedV) {
    best_yy[i + 1] = i
}

lines(xx, best_yy, col = "red", lwd = 2)

base_yy = predictedV/6000 * xx
lines(xx, base_yy, col = "blue", lwd = 2)

legend("bottomright", legend = c("best curve", "model", "baseline"),
    col = c("red", "green", "blue"), lwd = c(1, 1, 1), cex = 1)
title("Lift chart of XGBoost (validation)")
```




```
# dev.off()


# Calculate area ratio
a1v = sum(model_yy - base_yy)
a2v = sum(best_yy - base_yy)
a1v/a2v
```

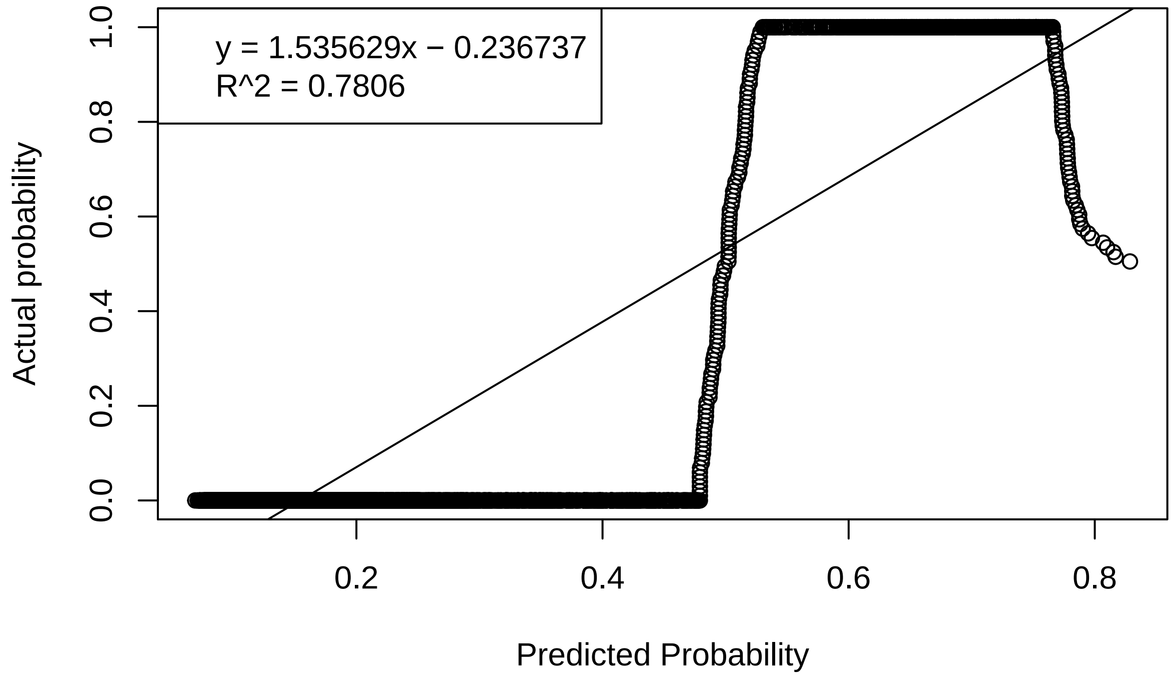
```
## [1] 0.4678744
```

```
# [1] 0.4678744



# Part 4.2.3: Use SSM(Sorting Smoothing Method) to estimate
# real probability

# 1. order the valid data according to predictive probability
valid_norm_sort = valid_norm[order(PredXBScoreV), ]
# 2. use SSM formula to evaluate actural probability 'Pi', we
# choose n =50 according to the paper
VALIDSIZE = dim(valid_norm)[1]

n = 50
actural_p_valid = rep(0, VALIDSIZE)
pred_valid = round(valid_norm_sort$PredXBScoreV)
pred_valid = prepend(pred_valid, rep(0, n), before = 1)
pred_valid = append(pred_valid, rep(0, n))

for (i in 1:VALIDSIZE) {
```

```r
    actural_p_valid[i] = sum(pred_valid[i:(i + 2 * n)])/(2 *
        n + 1)
}
valid_norm_sort = data.frame(valid_norm_sort, actural_p_valid)


# png('/Users/qinqingao/Desktop/Columbia/Courses/Fall
# 2018/EECS 6690/Project/figs/Scatter plot diagram of
# XB.png')
plot(valid_norm_sort$PredXBScoreV, valid_norm_sort$actural_p_valid,
    xlab = "Predicted Probability", ylab = "Actual probability")

yy = valid_norm_sort$actural_p_valid
xx = valid_norm_sort$PredXBScoreV
actual_fit = lm(yy ~ xx)

xx = seq(0, 1:0.1)
yy = predict(actual_fit, data.frame((xx)))
lines(xx, yy)

summary(actual_fit)
```

```
##
## Call:
## lm(formula = yy ~ xx)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.53072 -0.05315  0.02469  0.08845  0.42243
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.236737   0.003178   -74.5   <2e-16 ***
## xx           1.535629   0.010513   146.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.153 on 5998 degrees of freedom
## Multiple R-squared:  0.7806, Adjusted R-squared:  0.7805
## F-statistic: 2.134e+04 on 1 and 5998 DF,  p-value: < 2.2e-16
```

```r
legend("topleft", legend = c("y = 1.535629x - 0.236737", "R^2 = 0.7806"))
```


y = 1.535629x − 0.236737
R^2 = 0.7806
Actual probability
Predicted Probability
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8

```
# dev.off()
```