main_proj5

Group 8

April 24, 2018

```r
packages.used <- c("readxl", "ggplot2", "caret", "reshape2","randomForest",
                   "xgboost","pROC","e1071","InformationValue","devtools","nnet")

# check packages that need to be installed.
packages.needed <- setdiff(packages.used,
                           intersect(installed.packages()[,1],
                                     packages.used))
# install additional packages
if(length(packages.needed) > 0) {
  install.packages(packages.needed,dependencies = TRUE,
  repos = 'http://cran.us.r-project.org')
}

library(readxl)
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```r
library(reshape2)
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(xgboost)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(e1071)
library(InformationValue)
```

```
##
## Attaching package: 'InformationValue'

## The following objects are masked from 'package:caret':
```

```
##
##     confusionMatrix, precision, sensitivity, specificity
```

```r
library(devtools)
library(nnet)
```

Step 0: Specify directories.

Set the working directory to the data folder. Specify the training and the testing set. For data without an independent test/validation set, you need to create your own testing data by random subsampling. In order to obain reproducible results, set.seed() whenever randomization is used.

```r
#setwd("")
# here replace it with your own path or manually set it in RStudio to where this rmd file is located.
```

Provide directories for raw images. Training set and test set should be in different subfolders.

```r
#data <- read_xls("../data/data.xls",sheet = "Data",range = "A1:Y30002")
#data <- apply(data,2,as.numeric)
#data <- as.data.frame(data)

#set.seed(04182018)
#test_idx <- sample(1:30000,6000)
#train_idx <- setdiff(1:30000, test_idx)
#train <- data[train_idx,]
#test <- data[test_idx,]

#write.csv(train,"train.csv")
##write.csv(test,"test.csv")

train <- read.csv("../data/train.csv")
test <- read.csv("../data/test.csv")
train <- train[,-1]
test <- test[,-1]
```

Step 1: Set up controls for evaluation experiments.

In this chunk, ,we have a set of controls for the evaluation experiments.

- (T/F) cross-validation on the training set
- (number) K, the number of CV folds
- (T/F) run evaluation on an independent test set

```r
run.cv=FALSE # do not run cross-validation on the training set
K <- 5  # number of CV folds
run.test=TRUE # run evaluation on an independent test set
```

Using cross-validation or independent test set evaluation, we compare the performance of different classifiers or classifiers with different specifications.

Step 2: Process training and testing data.

```r
train$X4[which(train$X4!=1)] <- 2 # other marital status merged to unmarried
train$Y <- as.factor(train$Y)
```

```
test$X4[which(test$X4!=1)] <- 2 # other marital status merged to unmarried
test$Y <- as.factor(test$Y)

#train_new <- train[,c("X1","X5","X24","X25","Y")]
#test_new <- test[,c("X1","X5","X24","X25","Y")]

#write.csv(train_new,file = "train_new.csv")
#write.csv(test_new,file = "test_new.csv")
```

Step 3: Train classification models with training data.

Five different models are used to make prediction on wether a credict card client will default based on related information.

Model 1: Support Vector Machine

In parameter selecting part, for linear SVM, we select cost from 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000 and 10000. For RBF kernal, we select cost from 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10 and 100, and selet gamma from 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000 and 10000.

```
train <- train[,c(1,2,4,5,12:17,18:24)]
test <- test[,c(1,2,4,5,12:17,18:24)]

transform_df <- function(df){
  colnames(df) <- c("X1","X2","X4","X5","X12","X13","X14","X15","X16","X17","X18","X19","X20","X21","X2

  df$X2_1 <- ifelse(df$X2==1,1,0)
  df$X2_2 <- ifelse(df$X2==2,1,0)
  df$X4_1 <- ifelse(df$X4==1,1,0)
  df$X4_2 <- ifelse(df$X4==2,1,0)
  df <- df[,-c(2:3)]

  scaled_df <- scale(df[,-15])
  scaled_df <- cbind(scaled_df,df[,15])
  colnames(scaled_df)[ncol(scaled_df)] <- "Y"
  return(scaled_df)
}
```

```
train_scaled <- transform_df(train)
test_scaled <-transform_df(test)


# Corss Validation
# linear SVM
if(run.cv){
  tuned_lin <- tune.svm(as.factor(Y)~., data = as.data.frame(train_scaled),
                        cost = 10^(-4:4), kernel = "linear",
                        tunecontrol = tune.control(cross = 5))
  best_par_lin <- tuned_lin$best.parameters
} else{
  best_par_lin <- 0.1
}
```

```r
# RBF kernal SVM
if(run.cv){
  tuned_RBF <- tune.svm(as.factor(Y)~., data = as.data.frame(train_scaled),
                        gamma = 10^(-4:4), cost = 10^(-6:2),
                        kernel = "radial",
                        tunecontrol = tune.control(cross = 5))
  best_par_RBF <- tuned_RBF$best.parameters
} else{
  best_par_RBF <- c(10,0.001)
}
```

```r
load("../app/model_linear.rda")

#model_linear <- svm(as.factor(Y)~.,data = train_scaled,
#                        cost = best_par_lin,kernel = #"linear")

pred_linear <- predict(model_linear,test_scaled[,-ncol(test_scaled)])
err_lin <- mean(pred_linear!=as.numeric(test$Y))
#err_lin


#RBF kernel SVM with soft margin
load("../app/model_RBF.rda")

#model_RBF <- svm(as.factor(Y)~., data = train_scaled,
#                   cost = best_par_RBF[1], gamma = best_par_RBF[2], kernel = #"radial")

time1 <- system.time(pred_RBF <- predict(model_RBF, test_scaled[,-ncol(test_scaled)]))
err_RBFSVM <- mean(pred_RBF!=test$Y)
cat("error:",err_RBFSVM,"\n")
```

```
## error: 0.2216667
```

```r
cat("time:",time1[3])
```

```
## time: 27.86
```

Model 2: Random Forest

Data preprocessing

```r
df <- read.csv('../data/data.csv', skip=1) #Read the data
df <- df[,-1]
df$default.payment.next.month <- as.factor(df$default.payment.next.month)
#Convert target variable to factor type
df$ID <- NULL #Remove extraneous variables
```

Model building

```r
#Split data into training and test sets
set.seed(04182018)
test.i <- sample(1:nrow(df), .2*nrow(df), replace=FALSE)
test.data <- df[test.i,]
train.data <- df[-test.i,]
```

```r
#Build random forest tuning grid
rf_tune <- expand.grid(mtry=2,
                       ntree = seq(100, 1000, by = 250))
rf_tune$accuracy <- numeric(nrow(rf_tune))

#Tune parameters to find best model
if(run.cv){
  for(i in 1:nrow(rf_tune)){
        our.rf <- randomForest(default.payment.next.month ~.,
                               data=train.data, na.action = na.omit,
                               mtry=rf_tune$mtry[i],
                               ntree=rf_tune$ntree[i])
        rf.preds <- predict(our.rf, test.data)
        rf_tune$accuracy[i] <- mean(rf.preds == test.data$default.payment.next.month, na.rm = TRUE)
}
best_rf_params <- rf_tune[which.max(rf_tune$accuracy),]
} else{
  best_rf_params <- c(2,600)
}

load("../app/model_rf.rda")
#final_rf <- randomForest(default.payment.next.month ~.,
#                                data=train.data, na.action = na.omit,
#                                mtry=best_rf_params[1],
#                                ntree=best_rf_params[2])

time2 <- system.time(rf.pred_final <- predict(final_rf, test.data))
err_rf <- mean(rf.pred_final!=test.data$default.payment.next.month)
cat("error:",err_rf,"\n")
```

```
## error: 0.1856667
```

```r
cat("time:",time2)
```

```
## time: 2.87 0.03 2.92 NA NA
```

Model 3: Xgboost

```r
#Split data into predictors and target
#Note, features must be numeric, not integer, so convert them
df.features <- df[, -which(colnames(df) == 'default.payment.next.month')]
df.features <- apply(df.features, 2, function(x) as.numeric(x))

df.target <- df$default.payment.next.month

#Build XGboost tuning grid
xgb.tune <- expand.grid(eta=c(0.3, 0.5, 0.7),
                        gamma=c(0, 0.5, 1),
                        max_depth = c(2, 3, 4, 5, 10))
xgb.tune$accuracy <- numeric(nrow(xgb.tune))

# Tune parameters to find best model
#Each iteration performs 5-fold CV with 50 rounds of training XGBoost.
```

```r
if(run.cv){
set.seed(04182018)
for(i in 1:nrow(xgb.tune)){
        t1 = Sys.time()
        print(paste('Starting iteration', i, 'of', nrow(xgb.tune), ':'))

        param_list <- list(max_depth=xgb.tune$max_depth[i],
                           eta=xgb.tune$eta[i],
                           gamma = xgb.tune$gamma[i],
                           silent=1,
                           nthread=2,
                           objective='multi:softmax')

        model <- xgb.cv(data = as.matrix(df.features),
                        nrounds = 50,
                        nfold = 5,
                        metrics = list("merror"),
                        label = df.target,
                        params = param_list,
                        num_class = 4)

        # Takes mean of 10 training rounds with highest classification rate
        xgb.tune$accuracy[i] <- 1-(mean(sort(model$evaluation_log$test_merror_mean)[1:10]))
        t2 = Sys.time()
        print(paste('Iteration', i, 'took :', (t2-t1), 'seconds'))
}


# Show best parameters
best_gb_params <- xgb.tune[which.max(xgb.tune$accuracy),]
best_gb_params
} else{
    load("../app/model_xgb.rda")
}
```

```r
#model <- xgboost(data = as.matrix(df.features), label = as.numeric(df.target)-1,
#                  nrounds = 100, objective = "multi:softmax", num_class = 2)
test.data <- apply(test.data, 2, function(x) as.numeric(x))
test_Dmat <- xgb.DMatrix(as.matrix(test.data[,-ncol(test.data)]))
time3 <- system.time(pred_xgb <- predict(model,test_Dmat))
err_xgb <- mean(pred_xgb!=test.data[,ncol(test.data)])
cat("error:",err_xgb,"\n")
```

```
## error: 0.1223333
```

```r
cat("time:",time3)
```

```
## time: 0.31 0 0.06 NA NA
```

Model 4: Logistic Regression

```r
# Establish common downstream variables
classColumn <- "Y"
```

```r
badIndicator <- 1
goodIndicator <- 0
montonicConstraint <- "No"

# Training data
dataTrainingSample <- read.csv("../data/train.csv")
dataTrainingSample <- dataTrainingSample[1:5999,]
dataTrainingSample <- as.data.frame(dataTrainingSample)

# Testing data
dataTestingSample <- read.csv("../data/test.csv")
dataTestingSample <- as.data.frame(dataTestingSample)

# Get totals for downstream
numberOfObservations <- nrow(dataTrainingSample)
numberOfBads <- nrow(dataTrainingSample[dataTrainingSample[,classColumn]==badIndicator,])
numberOfGoods <- numberOfObservations-numberOfBads

# Create tons of model combinations
Variables <- c("X1","X2","X3","X4","X5","X6","X7","X8","X9","X10","X11","X12",
           "X13","X14","X15","X16","X17","X18","X19","X20","X21","X22","X23")
logisticSummary <- matrix(0, nrow = 1, ncol = 14)
colnames(logisticSummary) <- c("Variables", "Formula", "Var1", "Var2", "Var3", "Var4", "Var5", "Var6",
logisticSummary <- as.data.frame(logisticSummary)

logisticModels <- NULL

for (i in (1:length(Variables)))
{
  Vars <- 1
  temp <- logisticSummary

  FORMULA <- paste0(classColumn, " ~ ", Variables[i])
  MODEL <- glm(as.formula(FORMULA), data=dataTrainingSample, family=binomial(link="logit"))
  PREDICTED <- plogis(predict(MODEL, dataTestingSample))

  temp[1,"AUC"] <- auc(dataTestingSample[,classColumn],PREDICTED)
  temp[1,"P < 0.05"] <- length(which(as.numeric(coef(summary(MODEL))[,4]) < 0.05))-(Vars+1)

  if (temp[1,"P < 0.05"]==0)
  {
    temp[1,"Variables"] <- 1
    temp[1,"Var1"] <- Variables[i]
    temp[1, "Formula"] <- FORMULA

    logisticModels <- rbind(logisticModels, temp)

    variablesSublist1 <- Variables[which(Variables%in%c(Variables[i])==FALSE)]

    for (j in (1:length(variablesSublist1)))
    {
      temp <- logisticSummary
      Vars <- 2
```

```
      FORMULA <- paste0(classColumn, " ~ ", Variables[i], " + ", variablesSublist1[j])
      MODEL <- glm(as.formula(FORMULA), data=dataTrainingSample, family=binomial(link="logit"))
      PREDICTED <- plogis(predict(MODEL, dataTestingSample))

      temp[1,"AUC"] <- auc(dataTestingSample[,classColumn],PREDICTED)
      temp[1,"P < 0.05"] <- length(which(as.numeric(coef(summary(MODEL))[,4]) < 0.05))-(Vars+1)

      if (temp[1,"P < 0.05"]==0)
      {
        temp[1,"Variables"] <- 2
        temp[1,"Var1"] <- Variables[i]
        temp[1,"Var2"] <- variablesSublist1[j]
        temp[1, "Formula"] <- FORMULA

        logisticModels <- rbind(logisticModels, temp)

        variablesSublist2 <- Variables[which(Variables%in%c(Variables[i], variablesSublist1[j])==FALSE)]

        for (k in (1:length(variablesSublist2)))
        {
          temp <- logisticSummary
          Vars <- 3

          FORMULA <- paste0(classColumn, " ~ ", Variables[i], " + ", variablesSublist1[j], " + ", varia
          MODEL <- glm(as.formula(FORMULA), data=dataTrainingSample, family=binomial(link="logit"))
          PREDICTED <- plogis(predict(MODEL, dataTestingSample))

          temp[1,"AUC"] <- auc(dataTestingSample[,classColumn],PREDICTED)
          temp[1,"P < 0.05"] <- length(which(as.numeric(coef(summary(MODEL))[,4]) < 0.05))-(Vars+1)

          if (temp[1,"P < 0.05"]==0)
          {
            temp[1,"Variables"] <- 3
            temp[1,"Var1"] <- Variables[i]
            temp[1,"Var2"] <- variablesSublist1[j]
            temp[1,"Var3"] <- variablesSublist2[k]
            temp[1, "Formula"] <- FORMULA

            logisticModels <- rbind(logisticModels, temp)

          }

        }

      }

    }
  }
}

# Order from least to most predictive
logisticModels <- logisticModels[order(logisticModels$AUC),]
```

```r
tail(logisticModels)
```

```
##      Variables          Formula Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
## 25           3 Y ~ X1 + X6 + X9   X1   X6   X9    0    0    0    0    0
## 69           3 Y ~ X1 + X9 + X6   X1   X9   X6    0    0    0    0    0
## 241          3 Y ~ X6 + X1 + X9   X6   X1   X9    0    0    0    0    0
## 305          3 Y ~ X6 + X9 + X1   X6   X9   X1    0    0    0    0    0
## 1081         3 Y ~ X9 + X1 + X6   X9   X1   X6    0    0    0    0    0
## 1109         3 Y ~ X9 + X6 + X1   X9   X6   X1    0    0    0    0    0
##      Var9 Var10       AUC P < 0.05
## 25      0     0 0.7289305        0
## 69      0     0 0.7289305        0
## 241     0     0 0.7289305        0
## 305     0     0 0.7289305        0
## 1081    0     0 0.7289305        0
## 1109    0     0 0.7289305        0
```

```r
fit <- glm(default.payment.next.month~LIMIT_BAL+PAY_0+PAY_5,data = train.data,family = binomial("logit")
time4 <- system.time(pred_log <- predict(fit,newdata = as.data.frame(test.data[,-ncol(test.data)]),type
pred_log <- ifelse(pred_log>0.5,1,0)
err_log <- mean(pred_log!=test.data[,ncol(test.data)])
cat("error:",err_log,"\n")
```

```
## error: 0.1918333
```

```r
cat("time:",time4)
```

```
## time: 0.03 0 0.04 NA NA
```

Model 5: Neural Network

```r
options(warn = -1)
# This is required for the nnet package
dataTestingSample$Y <- as.factor(dataTestingSample$Y)

# Plug in everything to a neural network
FORMULA <- logisticModels[nrow(logisticModels), "Formula"]
MODEL <- nnet(as.formula(FORMULA), data=dataTrainingSample, size = 2, rang = 0.1,decay = 5e-4, maxit = 
```

```
## # weights:  11
## initial  value 1388.647330
## iter  10 value 1046.756065
## final  value 1046.750403
## converged
```

```r
time5 <- system.time(dataTestingSample$PREDICTION <- predict(MODEL, dataTestingSample[,-which(colnames(

dataTestingSample$PREDICTION[dataTestingSample$PREDICTION>0.5]<-1
dataTestingSample$PREDICTION[dataTestingSample$PREDICTION<=0.5]<-0
#dataTestingSample$PREDICTION == dataTestingSample$Y
err_nn <- sum(dataTestingSample$PREDICTION != dataTestingSample$Y)/nrow(dataTestingSample)
cat("error:",err_nn,"\n")
```

```
## error: 0.2216667
```

```r
cat("time:",time5)
```

```
## time: 0.04 0 0.04 NA NA
```

Reference: https://gist.githubusercontent.com/Peque/41a9e20d6687f2f3108d/raw/85e14f3a292e126f1454864427e3a189c2fe33f3/nnet_plot_update.r