Main File for Group 1

Group 1

In this project, we carried out model evaluation and selection for predictive analytics on the image data. We created a classification engine for grayscale images of poodles versus images of fried chickens. For feature extraction method, we used GIST because it is easy to understand and faster than SIFT. For classification methods, we tried xgboost, linear SVM, non-linear SVM and Random Forest. And in the end, we chose **GIST and xgboost** as our final model.

Step 0: Install packages and specify directories:

In this step, we check whether the needed packages are correctly installed and then set the path for training and testing data.

```
packages.used=c("e1071","EBImage","xgboost","ggplot2","gbm","tidyr")
packages.needed=setdiff(packages.used, intersect(installed.packages()[,1], packages.
used))
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}
library(EBImage)
library(e1071)
library(xgboost)
library(ggplot2)
library(tidyr)
```

Specify the path for the given 2000 pitures and the independent test data:

```
##here is where all the given 2000 pictures are put
experiment_dir <- "../data/training_data/"

##here is where all the independent testing set are put (it will be given in class)
extradata_dir <- "../data/extra_data/"
```

Please Note that: When doing the project, we split the data into training and testing set, instead of putting them separately in different files, we generate random numbers to split the data. To ensure the results are reproducible, set.seed() is used whenever randomization is needed. However, you can change the option if you like.

Step 1: set up controls for evaluation experiments.

In this chunk,we have a set of controls for the evaluation experiments. * (T/F) Use baseline or the Advance Model

Please Note: If you want recall the Train and Test function for the baseline model (which are called Train_gbm.r and Test_gbm.r in the lib folder),please choose T; if you want recall the Train

and Test function for the advanced model((which are called Train_xgboost.r and Test_xgboost in the lib folder)),please choose F.

- (T/F) use Baseline Model or the Advance Model.
- (T/F) use set.seed before randomization to get reproducible results.
- (T/F) cross-validation on the training set
- (number) K, the number of CV folds
- (number) train_proportion, the proportion of training data, preferable to be between 0.6-0.8
- (T/F) use our created new features to build the model
- (T/F) use our created new features on the independent test set
- (T/F) run evaluation on an independent test set

```
Baseline=F       #Use baseline or the Advance Model
set_seed=T        #use set.seed() whenever randomization needed
run.cv=T         # run cross-validation on the training set
K = 5                 # number of CV folds
train_proportion=0.75  # Porportion of the data that used for training the model

new.feature.train =T    #process features for gievn training set
new.feature.test=T      # process features for independent testing set
run.test=F     # run evaluation on an independent test set
```

Step 2: import training images class labels:

```
y<- read.table("../data/training_data/labels.csv", header=T)
if (Baseline){
  names(y) = c("labels")
  n<-nrow(y)
} else {
  n<-nrow(y)
  y<-y[1:n,]
}
```

Step 3: Preparation for Training the model:

Step 3.1: Load new features:

```
if(new.feature.train){
  X<-read.csv(paste(experiment_dir,"gist_features.csv",sep = ""),header=F)  #X is 2
000*512
  } else {
  X <-t(read.csv(paste(experiment_dir,"sift_features/sift_features.csv",sep = ""),h
eader=T))
    #X is 2000*5000
  }

if(new.feature.test){
  Independent_X<-read.csv(paste(extradata_dir,"gist_features.csv",sep = ""),header=
```

```
F)

  save(Independent_X, file="../output/Newfeature_test.RData")

} else {
  Independent_X<-t(read.csv(paste(extradata_dir,"sift_features/sift_features.csv",s
ep = ""),header=T))
}

#dim(X)
#dim(Independent_X)
```

Step 3.2: Random split the data to training and testing set:

```
if(set_seed){
  set.seed(0)
  Index<-sample(n,round(train_proportion*n,1),replace = F)
} else{
  Index<-sample(n,round(train_proportion*n,1),replace = F)
}
#n is the No. of all provided data
if (Baseline){
  df <- cbind(y, X)
  df.train <- df[Index, ]
  df.test <- df[-Index, ]
}else{
  Train.x<- data.matrix(X[Index,])
  Train.y<-y[Index]
  Test.x<-data.matrix(X[-Index,])
  Test.y<-y[-Index]

  Train.x<- data.matrix(Train.x,rownames.force = NA)
  Train.D <- xgb.DMatrix(data=Train.x,label=Train.y,missing = NaN)

  Test.x<- data.matrix(Test.x,rownames.force = NA)
  Test.D <- xgb.DMatrix(data=Test.x,label=Test.y,missing = NaN)
}
```

Step 4: Train a classification model with training images.

```
source("../lib/train.R")
source("../lib/test.R")
```

Step 4.1 : Model selection with cross-validation:

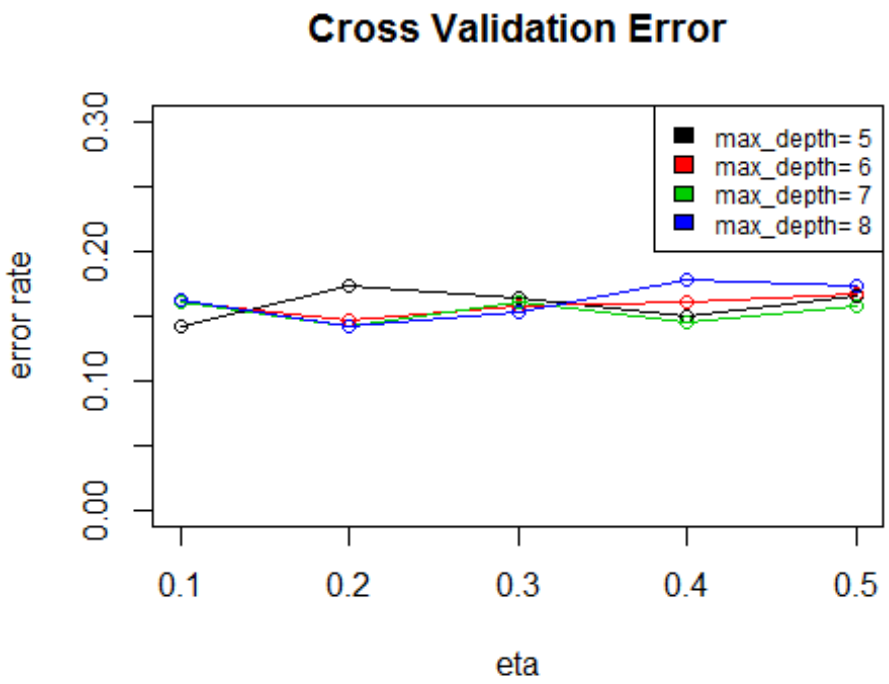
Do model selection by choosing among different values of training model parameters.

```r
if (run.cv){
  if (!Baseline){  ##We do CV for the Advanced Model
    #Set candidates:
    depth.list<- c(5,6,7,8)
    eta.list<- seq(0.1,0.5,0.1)

    #Initilize:
    error<-matrix(NA,nrow = length(eta.list),ncol = length(depth.list))
    iteration<-matrix(NA,nrow = length(eta.list),ncol = length(depth.list))

    for (i in 1:length(depth.list)) {
      for (j in 1:length(eta.list) ) {
       parameters <- list ( objective        = "binary:logistic",
                            #booser            = "gbtree",
                            eta               = eta.list[j],
                            max_depth         = depth.list[i],
                            subsample         = 0.5,
                            gamma = 0)

        crossvalid <- xgb.cv( params          = parameters,
                              data            = Train.D,
                              nrounds         = 100,
                              verbose         = 1,
                              maximize        = FALSE,
                              nfold           = 5,
                              early_stopping_rounds    = 8,
                              print_every_n       = 1)

        iteration[j,i]<-crossvalid$best_iteration
        error[j,i]<-
          as.numeric(crossvalid$evaluation_log[crossvalid$best_iteration,4])
      }
    save(error, file="../output/xgboost_err_cv.RData")
    }
  } else { ##We do CV for the Baseline Model:

    source("../lib/cross_validation_gbm.R")

    shrinkage_values <- c(0.1, 0.01, 0.001)
    model_labels = paste("GBM with shrinkage parameters =", shrinkage_values)
    nb_trees <- 15000

     err_cv_shr <- data.frame(matrix(nrow=nb_trees,ncol=3))
     for(k in 1:length(shrinkage_values)){
       cat("k=", k, "\n")
       err_cv_shr[,k] <- cv.function.shr(df.train, shrinkage_values[k], K)
       }
       err_cv_shr$nb.trees <- seq_len(nb_trees)
       err_cv_shr <- tidyr::gather(err_cv_shr, shrinkage, error, X1:X3,factor_key=F
```

```
ALSE)

      save(err_cv_shr, file="../output/gbm_err_cv_shr.RData")
  }
}
```

Visualize cross-validation results:

```
if(!Baseline){
  if (run.cv){
    plot(eta.list,error[,1],type= "o",col = 1,ylim = c(0,0.3),xlab = "eta",ylab = "
error rate",main = "Cross Validation Error")
    for (i in 2: length(depth.list)){points(eta.list,error[,i],type = "o",col = i)}
    legend("topright", fill = 1:length(depth.list), legend =
             paste("max_depth=",depth.list),cex = 0.75)
  }
}
```




```
if(Baseline){
  if(run.cv){
  load("../output/err_cv_shr.RData")
  #pdf("../fig/cv_results.pdf", width=7, height=5)
  g <- ggplot(err_cv_shr, aes(nb.trees, error, colour=shrinkage)) +
  geom_line() +
  ylab("CV Error Rate") +
```

```
  xlab("Number of trees") +
  labs(title="Baseline - GBM - Depth of 1") +
  scale_color_discrete(name="Shrinkage", labels=c("0.1","0.01","0.001"))
  g
  #dev.off()
}
}
```

Print the best parameters and train the model:

```
tm_train=NA

##We Train the Advanced Model
if (!Baseline){
  if (run.cv){
    ##Get the parameters with the best results:
    best.index<-which(error == min(error), arr.ind = TRUE)
    depth.choose<-depth.list[best.index[1,2]]
    era.choose<-eta.list[best.index[1,1]]
    iteration.choose<-iteration[best.index[1,1],best.index[1,2]]

    parameters <- list ( objective        = "binary:logistic",
                         #booser           = "gbtree",
                         eta               = era.choose,
                         max_depth         = depth.choose,
                         subsample         = 0.5,
                         gamma = 0)

    ##Print the Cross-Validation Error for the chosen Model:
    cat("The Cross Validation Error for Xgboost =", min(error), "\n")
    cat("The \'best\' eta for Xgboost =",era.choose,"\n")
    cat("The \'best\' itration for Xgboost =",iteration.choose,"\n" )
    cat("The \'best\' depth for Xgboost =",depth.choose )

    ##Train the model:
    tm_train <- system.time(fit_train_xg<-train_xgboost(Train.D,parameters,iteratio
n.choose))
  } else {
    #use pre-specify parameters:
    parameters <- list ( objective        = "binary:logistic",
                         #booser           = "gbtree",
                         eta               = 0.09,
                         max_depth         = 5,
                         subsample         = 0.5,
                         gamma = 0)

    ##Train the model:
    tm_train <- system.time(fit_train_xg<-train_xgboost(Train.D,parameters,100))
    cat("We don't have the CV error based on the user's selection")
  }
```

```
  ##Save the model:
  save(fit_train_xg, file="../output/fit_train_xg.RData")
}

## The Cross Validation Error for Xgboost = 0.142
## The 'best' eta for Xgboost = 0.2
## The 'best' itration for Xgboost = 38
## The 'best' depth for Xgboost = 7

if (Baseline){
  if(F){
    source("../lib/cross_validation_gbm.R")
    row <- which.min(err_cv_shr$error)
    model_best.nb_trees <- err_cv_shr$nb.trees[row]
    model_best.shrinkage <- err_cv_shr$shrinkage[row]
    par_best <- list(nb_trees=model_best.nb_trees, shrinkage=model_best.shrinkage)
    if(par_best$shrinkage == "X1"){
     par_best$shrinkage = 0.1
  }
    if(par_best$shrinkage == "X2"){
      par_best$shrinkage = 0.01
    }
   if(par_best$shrinkage == "X3"){
      par_best$shrinkage = 0.001
    }
  } else {
  par_best <- list(nb_trees=13000, shrinkage=0.001) # Optimal values fund by runnin
g the cv part (which is really long on the baseline)
  }
    err_cv <- cv.function(df.train, K, par_best)
    save(err_cv, file="../output/gbm_err_cv.RData")

  tm_train <- system.time(fit_train_gbm <- train_gbm(df.train, par_best))

  ##Save the model:
  #save(fit_train_gbm, file="../output/gbm_fit_train.RData")
}
```

Step 5: Model Evaluation:

Step 5.1: Get the training and testing error based on the given 2000 data:

```
#For the advanced model:
if (!Baseline){
  ##Get the test accuracy:
 pre_test<-test_xgboost(fit_train_xg,Test.x)
 cat("The Test Error for the advanced model based on the given 2000 images =",mean
```

```
(pre_test!=Test.y))
}

## The Test Error for the advanced model based on the given 2000 images = 0.128

#For the baseline model:
if (Baseline){
  pred_test <- test_gbm(fit_train_gbm, df.test)
   cat("The Test Error for the baseline model=",
       mean(pred_test!=df.test$labels))
}
```

Step 5.2: Make prediction for new testing set:

Here, we valuate the model with the completely holdout testing data.

```
tm_test=NA

if(run.test){
  ##Load features for the independent testing data:
  #load(file = ("../output/Newfeature_test.RData"))

  if (Baseline){
    load(file="../output/fit_train_bgm.RData")
    tm_test <- system.time(
           pred_newtest <-test_gbm(fit_train_gbm,Independent_X))

    save(pred_newtest, file="../output/pred_gbm.RData")
  } else {
    load(file="../output/fit_train_xg.RData")
    tm_test <- system.time(
           pred_newtest <-test_xgboost(fit_train_xg,Independent_X))

    save(pred_newtest, file="../output/pred_xgboost.RData")
  }

}

#y2<- read.table("../data/extra_data/labels.csv", header=T)
#y2<-y2[1:nrow(y2),]
#mean(pred_newtest!=y2)
```

Step 6: Summarize Running Time

```
cat("Time for training model=", tm_train[1], "s \n")

## Time for training model= 4.17 s

cat("Time for making prediction=", tm_test[1], "s \n")

## Time for making prediction= NA s
```