main

Shiqi Duan

2017/10/31

In this project, we carried out model evaluation and selection for predictive analytics on the image data. We created a classification engine for images of poodles, fried chickens, and muffins. For feature extraction method, we used HOG because it is easy to understand and much faster than SIFT and GIST. For classification methods, we tried Xgboost, SVM with linear and RBF kernels, Logistic Regression, Neural Networks and Random Forest. And in the end, we chose HOG and the majority vote of svm_rbf, logistic regression, and xgboost as our final (advanced) 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","OpenImageR","xgboost","ggplot2","gbm","tidyr", "nnet")
packages.needed=setdiff(packages.used, intersect(installed.packages()[,1], packages.used))
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}
library(nnet)
library(OpenImageR)
library(EBImage)
library(gbm)
library(e1071)
library(xgboost)
library(ggplot2)
library(tidyr)
```

Specify the path for the given 3000 pitures and the independent test data:

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

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

Please Note that: When doing the project, we split the data into training and test sets, 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: choose T for baseline model; choose F for our advanced 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 HOG features to build the model * (T/F) use HOG 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=F         # 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 mo
del

new.feature.train =T    #process features for given training set
new.feature.test=F      # 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.csv(paste(experiment_dir,"label_train.csv",sep = ""), header=TRUE,
as.is = TRUE)[,-1]
y <- as.factor(y)
n <- length(y)
```

Step 3: Preparation for Training the model:

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

Step 3.1:Extract features:

```
if(new.feature.train){
  X <- read.csv(paste(experiment_dir,"features_HOG.csv",sep = ""), header=TRU
E, as.is = TRUE)[,-1]  #X is 3000*54
}else{
  X <- read.csv(paste(experiment_dir,"sift_train.csv",sep = ""), header=TRUE,
 as.is = TRUE)[,-1]  #X is 3000*5000
}



if(run.test){
if(new.feature.test){
  tm_feature_test <- system.time(new.X <-
                                   feature(paste(newdata_dir,"images/",sep="
"),export=T))
```

```
  save(new.X, file="../output/test_HOG.RData")
} else {
  new.X<-read.csv(paste(newdata_dir,"sift_features.csv",sep = ""), header=TRU
E, as.is = TRUE) [,-1]
  }
}

#dim(X)
#dim(new.X)
```

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

```
if(set_seed){
  set.seed(0)
  idx<-sample(n,round(train_proportion*n,1),replace = F)
} else{
  idx<-sample(n,round(train_proportion*n,1),replace = F)
}
#n is the No. of all provided data
dat <- cbind(y, X)
dat_train <- dat[idx, ]
dat_test <- dat[-idx, ]
```

Step 4: Train classification models with training images. Step 4.1 : Model selection with cross-validation: Do model selection by choosing among different values of training model parameters.

```
if (run.cv){
  if (!Baseline){  ##We do CV for candidates in the Advanced Model

    ## parameters for xgboost
    max_depth = c(12, 14, 16)
    eta = c(0.15, 0.17, 0.19)
    gamma = c(0.4, 0.5, 0.6)
    min_child_weight = c(5.5, 6.5, 7.5)
    subsample = c(0.5, 0.6, 0.7)

    niter = length(max_depth)*length(eta)*length(gamma)*length(min_child_weig
ht)*length(subsample)
    cv_res = data.frame(max_depth = rep(NA,niter),
                        eta = rep(NA,niter),
                        gamma = rep(NA,niter),
                        min_child_weight = rep(NA,niter),
                        subsample = rep(NA, niter),
                        err = rep(NA,niter))
    count = 1
    for(i1 in max_depth){
      for(i2 in eta){
        for(i3 in gamma){
```

```
          for(i4 in min_child_weight){
            for(i5 in subsample){
              print(paste0("max_depth=",i1,
                           " eta=",i2," gamma=",i3,
                           " min_child_weight=",i4," subsample=",i5,
                           " count=", count))
              cv_res[count,1:5] <- c(i1,i2,i3,i4,i5)
              par.list = list(max_depth = i1, eta = i2,
                              gamma = i3, min_child_weight = i4, subsample =
i5)
              xgb <- cv.xgb(dat_train, K, par.list)
              cv_res[count,6] <- xgb$cv_error
              count <- count+1
            }
          }
        }
      }
    }
    save(cv_res, file = "../output/xgboost_err_cv.Rdata")
    # best parameters for Xgboost
    best <- which.min(cv_res[,6])
    cv_err.xgb <- min(cv_res[,6])
    xgb.para <- list(max_depth = cv_res[best,1], eta = cv_res[best,2],
                     gamma = cv_res[best,3], min_child_weight = cv_res[best,
4],
                     subsample = cv_res[best,5])

    # Print the Cross-Validation Error for the chosen Model:
    cat("The Cross Validation Error for Xgboost =", cv_err.xgb,"\n")
    cat("The \'best\' max_depth for Xgboost =",cv_res[best,1],"\n")
    cat("The \'best\' gamma for Xgboost =",cv_res[best,3],"\n" )
    cat("The \'best\' eta for Xgboost =",cv_res[best,2],"\n")
    cat("The \'best\' min_child_weight for Xgboost =",cv_res[best,4],"\n" )
    cat("The \'best\' subsample for Xgboost =",cv_res[best,5],"\n" )


    ## parameters for svm_rbf
    par.list = list(cost = c(0.01, 0.1, 1, 5, 10),
                    gamma = c(0.01, 0.1, 1, 10, 100))
    res_rbf <- cv.svm_rbf(dat_train, par.list, K)
    svm_rbf.para <- res_rbf$best.par

    save(res_rbf, file = "../output/svmRBF_err_cv.Rdata")
    # Print the Cross-Validation Error for the chosen Model:
    cat("The Cross Validation Error for svm_RBF =", res_rbf$smallest.cv_erro
r,"\n")
    cat("The \'best\' cost for svm_RBF =",svm_rbf.para$cost,"\n")
    cat("The \'best\' gamma for svm_RBF =",svm_rbf.para$gamma,"\n")
```

```
    ## parameters for Logistic
    iter_values <- seq(10,220,30)
    err_cv <- array(dim=c(length(iter_values), 2))
    for(k in 1:length(iter_values)){
      cat("k=", k, "\n")
      err_cv[k,] <- cv.lg(dat_train, par=list(maxit=iter_values[k]), K)
    }
    save(err_cv, file = "../output/logistic_err_cv.Rdata")
    # best parameters for Logistic
    min_cv_err <- min(err_cv[,1])
    iter_best <- iter_values[which.min(err_cv[,1] + err_cv[,2])]
    lg.para <- list(maxit=iter_best)

    # Print the Cross-Validation Error for the chosen Model:
    cat("The Cross Validation Error for Logistic =", min_cv_err,"\n")
    cat("The \'best\' iteration for Logistic =",iter_best,"\n")

    ### find cv-error for our advanced model
    n1 <- nrow(dat_train)
    n.fold <- floor(n1/K)
    s1 <- sample(rep(1:K, c(rep(n.fold, K-1), n1-(K-1)*n.fold)))
    cv_err.adv <- rep(NA, K)
    for (i in 1:K){
      train.data = dat_train[s1 != i,]
      data.test = dat_train[s1 == i, -1]
      y.test = dat_train[s1 == i, 1]

      fit.xgb = train.xgb(train.data, xgb.para)
      pred.xgb = test.xgb(fit.xgb, data.test)

      fit.rbf = train.svm_rbf(train.data, svm_rbf.para)
      pred.rbf = test.svm_rbf(fit.rbf, data.test)

      fit.lg = train.lg(train.data, lg.para)
      pred.lg = test.lg(fit.lg, data.test)

      pred = cbind(pred.xgb,as.numeric(pred.rbf)-1,as.numeric(pred.lg)-1)
      pred.adv = as.numeric(sapply(apply(pred,1,table),function(x){names(whic
h.max(x))}))
      cv_err.adv[i] = mean(pred.adv!=y.test)
    }
    # Print the Cross-Validation Error for the Advanced Model:
    cat("The Cross Validation Error for advanced model =", min(cv_err.adv),"\
n")
  }
  else{
    ##We do CV for the Baseline Model:
```

```
    ## parameters for baseline
    ntrees = c(100,300,500)
    interactiondepths = c(1,3,5)
    shrinkages = c(0.01,0.05,0.1)

    niter = length(ntrees)*length(interactiondepths)*length(shrinkages)
    cv_base = data.frame(ntree = rep(NA,niter),
                         interactiondepth = rep(NA,niter),
                         shrinkage = rep(NA,niter),
                         err = rep(NA,niter),
                         sd = rep(NA, niter))
    counter = 1
    for (i in ntrees) {
      for (j in interactiondepths) {
        for (k in shrinkages) {
          print(paste0("ntrees=",i," interactiondepths=",j," shrinkages=",k,"
 counter=",counter))
          cv_base$ntree[counter] = i
          cv_base$interactiondepth[counter] = j
          cv_base$shrinkage[counter] = k
          par.base = list(n_trees = i, interaction_depth = j, shrinkage = k)
          res_gbm = cv.gbm(dat_train, par.base, K)
          cv_base$err[counter] = res_gbm[1]
          cv_base$sd[counter] = res_gbm[2]
          counter = counter+1
        }
      }
    }
    save(cv_base, file = "../output/gbm_err_cv.Rdata")
    # best parameters for GBM
    best <- which.min(cv_base$err)
    cv_err.gbm <- min(cv_base$err)
    gbm.para <- list(n_trees = cv_base[best,]$ntree,
                     interaction_depth = cv_base[best,]$interactiondepth,
                     shrinkage = cv_base[best,]$shrinkage)

    # Print the Cross-Validation Error for the chosen Model:
    cat("The Cross Validation Error for GBM =", cv_err.gbm,"\n")
    cat("The \'best\' interaction_depth for GBM =",cv_base[best,]$interaction
depth,"\n")
    cat("The \'best\' n_trees for GBM =",cv_base[best,]$ntree,"\n" )
    cat("The \'best\' shrinkage for GBM =",cv_base[best,]$shrinkage,"\n")
  }
}
```

Step 4.2: train the model:

```
##We Train the Advanced Model
if (!Baseline){
  if (run.cv){
```

```
    ##Train the model using the best parameters chosen by cross validation:
    tm_train_xgb <- system.time(fit_train_xgb <- train.xgb(dat_train,xgb.par
a))
    tm_train_rbf <- system.time(fit_train_rbf <- train.svm_rbf(dat_train, svm
_rbf.para))
    tm_train_lg  <- system.time(fit_train_lg <- train.lg(dat_train, lg.para))

  } else {
    #use pre-specify parameters:
    xgb.para1 <- list(max_depth = 18,
                      eta = 0.18,
                      gamma = 0.45,
                      min_child_weight = 6.6,
                      subsample = 0.6)
    svm_rbf.para1 <- list(cost=10, gamma=100)
    lg.para1 <- list(maxit=190)

    ##Train the model:
    tm_train_xgb <- system.time(fit_train_xgb <- train.xgb(dat_train,xgb.para
1))
    tm_train_rbf <- system.time(fit_train_rbf <- train.svm_rbf(dat_train, svm
_rbf.para1))
    tm_train_lg  <- system.time(fit_train_lg <- train.lg(dat_train, lg.para
1))

    cat("We don't have the CV error based on the user's selection")
  }

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

```
## [1]  train-merror:0.216444
## Will train until train_merror hasn't improved in 100 rounds.
##
## [2]  train-merror:0.179111
## [3]  train-merror:0.160889
## [4]  train-merror:0.140444
## [5]  train-merror:0.132000
## [6]  train-merror:0.129778
## [7]  train-merror:0.118667
## [8]  train-merror:0.110667
## [9]  train-merror:0.103556
## [10] train-merror:0.100444
## [11] train-merror:0.091111
## [12] train-merror:0.088000
## [13] train-merror:0.081778
## [14] train-merror:0.075556
```

```
## [15] train-merror:0.068444
## [16] train-merror:0.061778
## [17] train-merror:0.058222
## [18] train-merror:0.051556
## [19] train-merror:0.048889
## [20] train-merror:0.046222
## [21] train-merror:0.042222
## [22] train-merror:0.039556
## [23] train-merror:0.036000
## [24] train-merror:0.032889
## [25] train-merror:0.029333
## [26] train-merror:0.025778
## [27] train-merror:0.024000
## [28] train-merror:0.022222
## [29] train-merror:0.020000
## [30] train-merror:0.019111
## [31] train-merror:0.016889
## [32] train-merror:0.016444
## [33] train-merror:0.015111
## [34] train-merror:0.016000
## [35] train-merror:0.014222
## [36] train-merror:0.012444
## [37] train-merror:0.011111
## [38] train-merror:0.011556
## [39] train-merror:0.010667
## [40] train-merror:0.009333
## [41] train-merror:0.008444
## [42] train-merror:0.008444
## [43] train-merror:0.008889
## [44] train-merror:0.007556
## [45] train-merror:0.008444
## [46] train-merror:0.007556
## [47] train-merror:0.006667
## [48] train-merror:0.006222
## [49] train-merror:0.004000
## [50] train-merror:0.004000
## [51] train-merror:0.003111
## [52] train-merror:0.003556
## [53] train-merror:0.003556
## [54] train-merror:0.003556
## [55] train-merror:0.003111
## [56] train-merror:0.002667
## [57] train-merror:0.002222
## [58] train-merror:0.002222
## [59] train-merror:0.002222
## [60] train-merror:0.002222
## [61] train-merror:0.001778
## [62] train-merror:0.002222
## [63] train-merror:0.002222
## [64] train-merror:0.001778
```

```
## [65] train-merror:0.001778
## [66] train-merror:0.001778
## [67] train-merror:0.001333
## [68] train-merror:0.001333
## [69] train-merror:0.001778
## [70] train-merror:0.001333
## [71] train-merror:0.001333
## [72] train-merror:0.001333
## [73] train-merror:0.002222
## [74] train-merror:0.001333
## [75] train-merror:0.001333
## [76] train-merror:0.001333
## [77] train-merror:0.001333
## [78] train-merror:0.000889
## [79] train-merror:0.000889
## [80] train-merror:0.001333
## [81] train-merror:0.000889
## [82] train-merror:0.001333
## [83] train-merror:0.000889
## [84] train-merror:0.000889
## [85] train-merror:0.001333
## [86] train-merror:0.000444
## [87] train-merror:0.000444
## [88] train-merror:0.000444
## [89] train-merror:0.000444
## [90] train-merror:0.000444
## [91] train-merror:0.000444
## [92] train-merror:0.000444
## [93] train-merror:0.000444
## [94] train-merror:0.000444
## [95] train-merror:0.000444
## [96] train-merror:0.000000
## [97] train-merror:0.000444
## [98] train-merror:0.000444
## [99] train-merror:0.000000
## [100]    train-merror:0.000444
## # weights:  168 (110 variable)
## initial  value 2471.877650
## iter  10 value 1843.364950
## iter  20 value 1610.639314
## iter  30 value 1457.011895
## iter  40 value 1358.783001
## iter  50 value 1260.170937
## iter  60 value 1209.622381
## iter  70 value 1181.664669
## iter  80 value 1155.334413
## iter  90 value 1140.324081
## iter 100 value 1125.070606
## iter 110 value 1111.821991
## iter 120 value 1099.964128
```

```
## iter 130 value 1093.582529
## iter 140 value 1087.114270
## iter 150 value 1083.443183
## iter 160 value 1079.474013
## iter 170 value 1077.577682
## iter 180 value 1075.900532
## iter 190 value 1074.266093
## final  value 1074.266093
## stopped after 190 iterations
## We don't have the CV error based on the user's selection

if (Baseline){
  if(run.cv){
    tm_train <- system.time(fit_train <- train.gbm(dat_train, gbm.para))
  } else {
  gbm.para1 <- list(n_trees=100, interaction_depth = 1, shrinkage=0.05) # Opt
imal values found by running the cv part (which is really long on the baselin
e)

  tm_train <- system.time(fit_train <- train.gbm(dat_train, gbm.para1))
  }

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

Step 5: Model Evaluation:

Step 5.1: Get the training and test error based on the given 3000 data:

```
#For the advanced model:
if (!Baseline){
  ##Get the test accuracy:
 pre_test.xgb<-test.xgb(fit_train_xgb, dat_test[,-1])
 pre_test.rbf<-test.svm_rbf(fit_train_rbf, dat_test[,-1])
 pre_test.lg <- test.lg(fit_train_lg, dat_test[, -1])
 com <- cbind(pre_test.xgb, as.numeric(pre_test.rbf)-1, as.numeric(pre_test.l
g)-1)
 pre_test_adv = as.numeric(sapply(apply(com,1,table),function(x){names(which.
max(x))}))

 cat("The Test Error for the advanced model based on the given 3000 images =
",mean(pre_test_adv != dat_test[,1]))
}

## The Test Error for the advanced model based on the given 3000 images = 0.1
946667

#For the baseline model:
if (Baseline){
  pred_test <- test.gbm(fit_train, dat_test[,-1])
```

```
   cat("The Test Error for the baseline model=",
       mean(pred_test != dat_test[,1]))
}
```

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_gbm.RData")
    tm_test <- system.time(
           pred_new <-test.gbm(fit_train,new.X))

    save(pred_new, file="../output/pred_gbm.RData")
  } else {
    load(file="../output/fit_train_xgb.RData")
    load(file="../output/fit_train_rbf.RData")
    load(file="../output/fit_train_lg.RData")

    tm_test_xgb <- system.time(
           pred_new_xgb <-test.xgb(fit_train_xgb,new.X))
    tm_test_rbf <- system.time(
           pred_new_rbf <-test.svm_rbf(fit_train_rbf,new.X))
    tm_test_lg <- system.time(
           pred_new_lg <-test.lg(fit_train_lg,new.X))

    tcom = cbind(pred_new_cgb,as.numeric(pred_new_rbf)-1,as.numeric(pred_new_
lg)-1)
    pre_new_adv = as.numeric(sapply(apply(tcom,1,table),function(x){names(whi
ch.max(x))}))

    save(pre_new_adv, file="../output/pred_new_adv.RData")
  }

}
```

Step 6: Summarize Running Time

```
if(!Baseline){
  cat("time for training model on Xgboost = ", tm_train_xgb[1],"s \n")
  cat("time for training model on svm_RBF = ", tm_train_rbf[1],"s \n")
  cat("time for training model on logistic = ", tm_train_lg[1],"s \n")
  cat("total time for training advanced model is around = ",
      tm_train_xgb[1]+tm_train_lg[1]+tm_train_rbf[1],"s \n")
```

```
# cat("Time for making prediction on Xgboost = ", tm_test_xgb[1], "s \n")
# cat("Time for making prediction on svm_RBF = ", tm_test_rbf[1], "s \n")
# cat("Time for making prediction on logistic  = ", tm_test_lg[1], "s \n")
# cat("total time for making prediction on advanced model is around = ",
#      tm_test_xgb[1]+tm_test_lg[1]+tm_test_rbf[1],"s \n")
}else{
  cat("Time for training model=", tm_train[1], "s \n")
  cat("Time for making prediction=", tm_test[1], "s \n")

}
```

```
## time for training model on Xgboost =  3.351 s
## time for training model on svm_RBF =  0.843 s
## time for training model on logistic =  0.35 s
## total time for training advanced model is around =  4.544 s
```