Project 3-Group 7

Xinrou Li, Xiuruo Yan

March 18th, 2018

In your final repo, there should be an R markdown file that organizes **all computational steps** for evaluating your proposed image classification framework.

This file is currently a template for running evaluation experiments of image analysis (or any predictive modeling). You should update it according to your codes but following precisely the same structure.

Step 0: specify directories

Set the working directory to the image 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.

Step 1: set up controls for evaluation experiments.

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

- (T/F) process features for training set
- (T/F) run evaluation on an independent test set
- (T/F) run the training process
- (T/F) use the new dataset (or just original sift data)
- (T/F) use the baseline model
- (number) number of trees adaboost use

```
run.cv = F # run cross-validation on the training set
K <- 5 # number of CV folds
run.feature.train = T # process features for training set
run.test = TRUE # run evaluation on an independent test set
run.feature.test = TRUE # process features for test set
model.train = T
```

Boolean variables for Feature Extraction procedures

```
run.pca = FALSE
run.hogs = FALSE
run.sift = FALSE
run.sift_pca = FALSE
run.color = TRUE
```

Boolean variables indicating which model to run

```
run.gbm = TRUE
run.svm.linear = FALSE
run.svm.rbf = FALSE
run.rf = FALSE
run.xgboost = FALSE
run.adaboost = FALSE
run.lg = FALSE

# for GBM
model_values<-list(ntree = seq(1, 500, 1))
# for SVM
svm_lin_values <- list(cost = c(0.1, 1, 10, 50, 100, 150))
# for SVM
svm_rbf_values <- list(cost = c(0.1, 1, 10, 50, 100, 150),
                       gamma = c(0.01, 0.5, 1, 5, 10))
# for XGboost
xgboost_values <- seq(0.01,0.25,0.05)
# for Logistic Regression
lg_values <- list(maxit = seq(10,30,1))
# for Adaboost
adaboost_values <- list(ntree = c(510, 20, 30))
# for Random Forest
rf_par <- list(ntree = c(50, 100, 150, 200, 250, 300, 350))
```

Step 2: import training images class labels.

For the example of zip code digits, we code digit 9 as “1” and digit 7 as “0” for binary classification.

```
label_train <- read.csv("../data/image/label_train.csv")[, 3]
```

Step 3: construct visual feature

For this simple example, we use the row averages of raw pixel values as the visual features. Note that this strategy **only** works for images with the same number of rows. For some other image datasets, the feature function should be able to handle heterogeneous input images. Save the constructed features to the output subfolder.

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features that are required by all the models you are going to evaluate later.

```
source("../lib/feature.R")

tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(img_train_dir,
                                                       run.pca = run.pca,
                                                       run.hogs = run.hogs,
                                                       run.sift = run.sift,
                                                       run.sift_pca = run.sift_pca,
                                                       run.color = run.color,
                                                       export=TRUE))
}

tm_feature_test <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <-   feature(img_test_dir,
                                                       run.pca = run.pca,
                                                       run.hogs = run.hogs,
                                                       run.sift = run.sift,
                                                       run.sift_pca = run.sift_pca,
                                                       run.color = run.color,
                                                       export=TRUE))
}

save(dat_train, file="../output/feature_train.RData")
save(dat_test, file="../output/feature_test.RData")

if(run.sift){
  #dat_test <- read.csv(img_test_dir, header=T)
  baseline_dat_train <- read.csv("../data/image/sift_train.csv", header=T)
  dat_test <- baseline_dat_train[,-1]
}
```

Step 4: Train a classification model with training images

Call the train model and test model from library.

```
source("../lib/train_final.R")
source("../lib/test_final.R")

load("../output/feature_train.RData")
```

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters, that is, the interaction depth for GBM in this example.

```
source("../lib/cross_validation.R")

# SIFT feature default to be true for baseline model

if( run.sift ){
  baseline_dat_train <- read.csv("../data/image/SIFT_train.csv", header=T)
  baseline_dat_train <- baseline_dat_train[,-1]
  dat_train <- baseline_dat_train
}
```

```
# Boolean variables to indicate which model to perform cross-validation.
cv.gbm = T
cv.svm.lin = F
cv.svm.rbf = F
cv.rf = F
cv.xgboost = F
cv.lg = F
cv.adaboost = F

if(run.cv) {
  if( cv.gbm ){
    err_cv <- array(dim=c(length(model_values), 2))
      for(k in 1:length(model_values)){
        cat("k=", k, "\n")
        err_cv[k,] <- cv.function( as.data.frame(dat_train), label_train, model_value
s[k], K, cv.gbm = T)
      }
  }
  if( cv.svm.lin ){
    err_cv <- array(dim=c(length(svm_lin_values), 2))
      for(k in 1:length(svm_lin_values)){
        cat("k=", k, "\n")
        err_cv[k,] <- cv.function( dat_train, label_train, d = svm_lin_values[k], K =
K, cv.svm.lin = T)
    }
  }
  if( cv.svm.rbf ){
    err_cv <- array(dim=c(length(svm_rbf_values), 2))
      for(k in 1:length(svm_rbf_labels)){
        cat("k=", k, "\n")
        err_cv[k,] <- cv.function( dat_train, label_train, d = svm_rbf_labels[k], K =
K, cv.svm.rbf = T)
    }
  }
  if( cv.rf ){
    err_cv <- array(dim=c(nrow(rf_par), 2))
      model_values = rf_par
      for(k in 1:nrow(rf_par)){
        cat("k=", k, "\n")
```

```
        err_cv[k,] <- cv.function( dat_train, label_train, rf_par[k,], K, cv.rf = T)
    }
  }

  if( cv.xgboost ){
    err_cv <- array(dim=c(length(xgboost_values), 2))
      for(k in 1:length(xgboost_values)){
        cat("k=", k, "\n")
        err_cv[k,] <- cv.function( as.data.frame(dat_train), label_train, xgboost_val
ues[k], K,
                                   cv.xgboost = T)
      }
    }
  if( cv.lg ){
    err_cv <- array(dim=c(length(lg_values), 2))
      for(k in 1:length(lg_values)){
        cat("k=", k, "\n")
        err_cv[k,] <- cv.function( as.data.frame(dat_train), label_train, lg_values[k
], K, cv.lg = T)
      }
    }
  if( cv.adaboost ){
    err_cv <- array(dim=c(length(adaboost_values), 2))
      for(k in 1:length(adaboost_values)){
        cat("k=", k, "\n")
        err_cv[k,] <- cv.function( as.data.frame(dat_train), label_train, adaboost_va
lues[k], K,
                                   cv.adaboost = T)
      }
  }

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

Visualize cross-validation results.

```
if(run.cv){
  if(cv.gbm){
    load("../output/err_cv.RData")
    #pdf("../fig/cv_results.pdf", width=7, height=5)
    plot(model_values, err_cv[,1], xlab="Interaction Depth", ylab="CV Error",
    main="Cross Validation Error", type="n", ylim=c(0, 1))
    points(model_values, err_cv[,1], col="blue", pch=16)
    lines(model_values, err_cv[,1], col="blue")
    arrows(model_values, err_cv[,1]-err_cv[,2], model_values, err_cv[,1]+err_cv[,2],
    length=0.1, angle=90, code=3)
  }
  if(cv.svm.lin){
    load("../output/err_cv.RData")
    #pdf("../fig/cv_results.pdf", width=7, height=5)
```

```
    plot(svm_lin_values, err_cv[,1], xlab="Cost & Gamma", ylab="CV Error",
    main="Cross Validation Error", type="n", ylim=c(0, 1))
    points(svm_lin_values, err_cv[,1], col="blue", pch=16)
    lines(svm_lin_values, err_cv[,1], col="blue")
    arrows(svm_lin_values, err_cv[,1]-err_cv[,2], svm_lin_values, err_cv[,1]+err_cv[,
2],
    length=0.1, angle=90, code=3)
  }
  if(svm_rbf_values){
    load("../output/err_cv.RData")
    #pdf("../fig/cv_results.pdf", width=7, height=5)
    plot(svm_rbf_values, err_cv[,1], xlab="Cost & Gamma", ylab="CV Error",
    main="Cross Validation Error", type="n", ylim=c(0, 1))
    points(svm_rbf_values, err_cv[,1], col="blue", pch=16)
    lines(svm_rbf_values, err_cv[,1], col="blue")
    arrows(svm_rbf_values, err_cv[,1]-err_cv[,2], svm_rbf_values, err_cv[,1]+err_cv[,
2],
    length=0.1, angle=90, code=3)
  }

  if(cv.rf){
    load('../output/err_cv.RData')
    models <- cbind(rf_par, err_cv)
    colnames(models) <- c('ntree','err','sd')
    models <- models[,-3]
    ggplot(data=models, aes(x=as.factor(ntree), y=err, colour = as.factor(ntree))) +
    geom_line() +
    geom_point( size=4, shape=21, fill="white")
  }

  if(cv.lg){
    load("../output/err_cv.RData")
    #pdf("../fig/cv_results.pdf", width=7, height=5)
    plot(model_values, err_cv[,1], xlab="Maximum iteraction", ylab="CV Error",
    main="Cross Validation Error", type="n", ylim=c(0, 1))
    points(model_values, err_cv[,1], col="blue", pch=16)
    lines(model_values, err_cv[,1], col="blue")
    arrows(model_values, err_cv[,1]-err_cv[,2], model_values, err_cv[,1]+err_cv[,2],
    length=0.1, angle=90, code=3)
  }
  if(cv.xgboost){
    load("../output/err_cv.RData")
    #pdf("../fig/cv_results.pdf", width=7, height=5)
    plot(model_values, err_cv[,1], xlab="XGboost_Paramters", ylab="CV Error",
    main="Cross Validation Error", type="n", ylim=c(0, 1))
    points(model_values, err_cv[,1], col="blue", pch=16)
    lines(model_values, err_cv[,1], col="blue")
    arrows(model_values, err_cv[,1]-err_cv[,2], model_values, err_cv[,1]+err_cv[,2],
    length=0.1, angle=90, code=3)
  }
```

```
####adaboost
####takes so much time so we just choose the best ntree(10,20,30) with the minimum te
st error

}
```

- Choose the “best”" parameter value

```
if( run.cv ){
  if( !run.rf ){
    model_best=model_values[1]
  } else {
    model_best=model_values[1,]
  }
  # Best parameter for GBM
  if(run.gbm){
    model_best<- model_values[which.min(err_cv[,1])]
  }
  # Best parameter for SVM Linear
  if(run.svm.linear){
    model_best <- svm_lin_values[which.min(err_cv[,1])]
  }
  # Best parameter for SVM RBF
  if(run.svm.rbf){
    model_best <- svm_rbf_values[which.min(err_cv[,1])]
  }
  #Best parameter for Ranfom Forest
  if(run.rf){
    model_best <- rf_par[which.min(err_cv[,1]),]
  }
  # Best parameter for XGboost
  if(run.xgboost){
    model_best<- xgboost_values[which.min(err_cv[,1])]
  }
  # Best parameter for Adaboost
  if(run.adaboost){
    model_best <- adaboost_values[which.min(err_cv[,1])]
  }
  # Best parameter for Logistic Regression
  if(run.lg){
    model_best <- lg_values[which.min(err_cv[,1]),]
  }
}
```

- Train the model with the entire training set using the selected model (model parameter) via cross-validation.

```
if( !run.cv ){
  if( run.gbm){
```

```
    model_best <- list(n.trees = 441)
  }
  if( run.rf ){
    model_best <- list(ntree = 200)
  }
  if( run.svm.linear ){
    model_best <- list(cost=150)
  }
  if( run.svm.rbf){
    model_best <- list(cost=50, gamma=1)
  }
  if( run.xgboost ){
    model_best <- list(mtry = 20, ntree = 2000)
  }
  if( run.adaboost ){
    model_best <- list(ntree = 30)
  }
  if( run.lg ){
    model_best <- list(maxit = 16)
  }
}


if( model.train ){
  # Train GBM
  if( run.gbm ){
    tm_train_gbm <- system.time(model.gbm <- train( dat_train, label_train,
                                                    run.gbm = TRUE, model_best))
  }
  # Train SVM Linear
  if( run.svm.linear ){
    tm_train_svm_linear <- system.time(model.svm.linear <- train(dat_train, label_tra
in,
                                                                 run.svm.linear = TRUE
, model_best))
  }
  # Train SVM RBF
  if( run.svm.rbf ){
    tm_train_svm_rbf <- system.time(model.svm.rbf <- train(dat_train, label_train, mo
del_best,
                                                           run.svm.rbf = TRUE, model_best)
)
  }
  # Train Random Forest
  if( run.rf ){
    tm_train_rf <- system.time(model.rf <- train(dat_train, label_train, run.rf = TRU
E, model_best))
  }
    # Train XGboost
  if( run.xgboost ){
```

```
    tm_train_xgboost <- system.time(model.xgboost <- train(dat_train, label_train,
                                                           run.xgboost = TRUE, model_
best))
  }
  # Train Adaboost
  if( run.adaboost ){
    tm_train_adaboost <- system.time(model.adaboost <- train(dat_train, label_train,
model_best,
                                                    run.adaboost = TRUE, model_best))
  }
  # Train Logistics Regression
  if( run.lg ){
    tm_train_lg <- system.time(model.lg <- train( dat_train, label_train,
                                                  run.lg = TRUE, model_best))
  }
}
```

Step 5: Make prediction and Summarize Running Time

Feed the final training model with the completely holdout testing data.

```
tm_test=NA
if(run.test){
  load(file="./output/feature_test.RData")
  if(baseline){
    tm_test <- system.time(pred_test <- test_baseline(fit_train_gbm, dat_test,new.fea
tures))
    save(pred_test, file="../output/pred_test.RData")
  }else{
    tm_test <- system.time(pred_test <- test_ada(fit_train_base, dat_test,new.feature
s))
    save(pred_test, file="~/Desktop/[ADS]Advanced Data Science/Fall2017-project3-fall
2017-project3-grp9/output/pred_test.RData")
  }
}
```

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

```
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
cat("Time for training model=", tm_train[1], "s \n")
cat("Time for making prediction=", tm_test[1], "s \n")
```