R Code

```
## Machine Learning Project
#
#                  Author: N. Murphy
#
## Version: Masters-Coursework-ML-ML-Project-NMURPHY.R
#
## Version Control:
# ~/R/ML/Assignents/Project/
#
# Problem Specification: This script implements Q1 of assignment 2
# Data Specification: None
# Configuration Control: userpath/MATLAB/Master/Coursework/ML
# Version Control: No version control
# References: None

## 1. Data Description
#
# None

## 2. Clear workspace
rm(list=ls()) # clear environment

## 3. Paths
# 3.1. setwd("<location of your dataset>")
rootp <- getwd()
setwd("..") # move up one level in the directory tree
# setwd(path.expand('~'))
filentrain <- "/Train_Digits_20171108.csv"
filentest <- "/Test_Digits_20171108.csv"
fpath <- "~/R/ML/Assignments/Project"
ffilentrain <- paste(fpath,filentrain,sep="")
ffilentest <- paste(fpath,filentest,sep="")

## 4. Load Train Data
traindata <- read.csv(ffilentrain)
testdata <- read.csv(ffilentest)

## 5. Load libraries
library(e1071)
library(randomForest)
library(neuralnet)
library(nnet)
library(rpart)
library(rattle)
library(gbm)
library(tree)
library(glm)


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

```
## Pre-processing ##
####################

# true output
ytemp <- traindata[,1]
# Even/odd
is.even <- function(x) x%%2 == 0
is.odd <- function(x) x%%2 == 1
eveninds <- which(is.even(ytemp), arr.ind = TRUE)
oddinds <- which(is.odd(ytemp), arr.ind = FALSE)

# Convert y to +-1 for odd (-1) and even (+1)
y <- matrix(NA,nrow=length(ytemp),ncol=1)
y[eveninds,1] <- 1
y[oddinds,1] <- 0#-1

## Inputs
X <- traindata[,2:785]

## Split into training and validation (80:20 split)
# N <- dim(X)[1]
# Xtrain <- X[1:(N*0.8),]
# Xval <- tail(X,N*0.2)
# ytrain <- y[1:(N*0.8)]
# yval <- tail(y,N*0.2)
N <- dim(X)[1]
# Xtrain <- X
ytrain <- y

## Reduced training set using Principal Component Analysis
Xtrain_cov <- var(X)
PC_Xtrain <- prcomp(Xtrain_cov)

## find number of principal components to use- use number of principal components that explain at least
eigs <- PC_Xtrain$sdev^2
fv <- rbind(Cumulative = cumsum(eigs)/sum(eigs))
numPC <- which(fv[1,]>0.99)[1]

## create reduced training set
Xtrain_PCA <- as.matrix(X)%*%PC_Xtrain$rotation[,1:numPC]  #extract first numpc princ. comps
Xtrain <- Xtrain_PCA


#######################################
## View some images from training set ##
# Create a 28*28 matrix with pixel colour values
m = matrix(unlist(traindata[10,-1]), nrow = 28, byrow = TRUE)
# Plot that matrix
image(m,col=grey.colors(255))
# reverses (rotates the matrix)
rotate <- function(x) t(apply(x, 2, rev))
# Plot the first 6 images
par(mfrow=c(2,3))
```

```r
lapply(1:6,
       function(x) image(
         rotate(matrix(unlist(traindata[x,-1]),nrow = 28, byrow = FALSE)),
         col = grey.colors(256),
         xlab = traindata[x,1]
       )
)



## Implement various classification methods

########################
# 1. Pocket Algorithm ##
########################
# Vectorized version to take in NxP matrix X
iter <- 1200
E_in_w_hat <- matrix(NA,iter,1)
E_in_w <- matrix(NA,iter,1)

PLA = function(X,y,w0)
{
  X = cbind(1,X)
  w_hat = matrix(w0,dim(X)[2],1)
  misclass = (sign(as.matrix(X)%*%w_hat)!=y)
  for (i in 1:iter)
  {
    pick = sample(which(misclass==TRUE),1)
    w = w_hat +X[pick,]*y[pick]
    #evaluate errors for updates w and w_hat which has given lower E_in so far
    misclass_w = (sign(as.matrix(X)%*%w)!=y)
    misclass_w_hat = (sign(as.matrix(X)%*%w_hat)!=y)
    E_in_w[i,1] <- sum(misclass_w)
    E_in_w_hat[i,1] <- sum(misclass_w_hat)
    if (E_in_w_hat[i,1]>E_in_w[i,1] ){
       w_hat <- w
       misclass <- misclass_w
    } else {
       w_hat <- w_hat
       misclass <- misclass_w_hat
    }

   # err<- (sign(as.matrix(X)%*%t(w))-y)^2
  }
  return(list(E_in_w_hat,E_in_w,w_hat,w))
}

## 5-fold Cross-validation
R <- 5
folds <- cut(seq(1,nrow(Xtrain)),breaks=10,labels=FALSE)
Ein_pocket <- cbind(rep(NA,R))
E_val_pock <- Ein_pocket
# ValData <- list()
```

```r
# TrainData <- list()

## Split data into 10 folds
for(j in 1:R){
  #Segement your data by fold using the which() function
  Inds <- which(folds==j,arr.ind=TRUE) # indices of current validation fold
  ValData <- Xtrain[Inds,]
  TrainData <- Xtrain[-Inds,]
  ydatatrain <- ytrain[-Inds,]
  ydataval <- ytrain[Inds,]

  res = PLA(TrainData,ydatatrain,matrix(0,dim(TrainData)[2]+1,1))

  #Ein_percept[j] <- tail(res[[2]]/2500,1)
  Ein_pocket[j] <- tail(res[[1]]/dim(TrainData)[1],1)

  ## Validation error for pocket and perceptron
  E_val_pock[j] <- sum(sign(as.matrix(cbind(1,ValData))%*%res[[3]])!=ydataval)/250
  #E_val_perc[j] <- sum(sign(as.matrix(cbind(1,ValData))%*%res[[4]])!=ydataval)/500

}

## Cross-val. error
E_CV_pock <- mean(E_val_pock)
##vector of in sample errors
Ein_pocket

## Plot insample error vs iter
plot(1:iter,res[[2]]/2500,type = "l") #Ein percept
plot(1:iter,res[[1]]/2500,type = "l") #Ein pocket


###########################
# 2. Logistic Regression ##
###########################

## 5-fold Cross-validation
R <- 5
folds <- cut(seq(1,nrow(Xtrain)),breaks=10,labels=FALSE)
Ein_pocket <- cbind(rep(NA,R))
E_val_pock <- Ein_pocket
# ValData <- list()
# TrainData <- list()

## Split data into 10 folds
for(j in 1:R){
  #Segement your data by fold using the which() function
  Inds <- which(folds==j,arr.ind=TRUE) # indices of current validation fold
  ValData <- Xtrain[Inds,]
  TrainData <- Xtrain[-Inds,]
  ydatatrain <- ytrain[-Inds,]
  ydataval <- ytrain[Inds,]
```

```r
  # Implement the glm model for logistic regression
  GLM <- glm(as.formula(paste('ydatatrain  ~ ',paste(paste('Pixel_',1:784,sep=''), collapse='+'), sep='
  GLM <- glm(factor(ydatatrain)~.,data=TrainData ,family="binomial")

  ## Predict on validation data
  pred_glm <- predict.glm(GLM,newdata = ValData, type="response")
#
# ## Confusion matrix
# table(`Actual Class` = yval, `Predicted Class` = pred_glm)

  # Error and accuracy
  E_val_glm <- sum(ydataval != pred_glm)/nrow(ydataval)
  Acc_SVM = 1 - E_val_glm


}

# Fit a neural network 1 hidden layers
Xtrainglm <- X[1:(N*0.8),]
ytrainglm <- y[1:(N*0.8)]
Xval <- tail(X,N*0.2)
yval <- tail(ytrain,N*0.2)
GLM <- train(Xtrainglm ,factor(ytrainglm ),method = "glm" ,family = "binomial")
pred_glm <- predict(GLM,newdata = Xval)
E_val_glm <- sum(yval!= pred_glm)/length(yval)

## Test data prediciton
pred_glmtest <- predict(GLM,newdata = testdata[,2:785])
pred_glmtest <- ifelse(pred_glmtest,1,0)

##############################
# 3. Support Vector Machine ##
##############################

### Test different kernals using CV:
#different kernals
kern = c('polynomial','linear','radial')
#C = 50
## R-fold Cross-validation
R <- 5
folds <- cut(seq(1,nrow(Xtrain)),breaks=R,labels=FALSE)
E_val_svm <- cbind(rep(NA,R))
E_CV_svmkern <- cbind(rep(NA,length(kern)))


### CV error for different cost parameters
C <- c(0.01,1,5,30,80)
## R-fold Cross-validation
R <- 5
folds <- cut(seq(1,nrow(Xtrain)),breaks=R,labels=FALSE)
E_val_svm <- cbind(rep(NA,R))
E_CV_svm <- matrix(NA,length(kern),length(C))
```

```r
for (k in 1:length(kern)){
  for (i in 1:length(C)){
  ## Split data into R folds
  for(j in 1:R){
    #Segement your data by fold using the which() function
    Inds <- which(folds==j,arr.ind=TRUE) # indices of current validation fold
    ValData <- Xtrain[Inds,]
    TrainData <- Xtrain[-Inds,]
    ydatatrain <- ytrain[-Inds,]
    ydataval <- ytrain[Inds,]

    ## Using R's in-built function 'svm'
    SVM_fn <- svm(factor(ydatatrain) ~ ., data=TrainData, type='C-classification', kernel=kern[k], cost=

    ## Predict on validation data
    predSVM <- predict(SVM_fn,newdata = ValData, type = "class")

    ## Confusion matrix
    table(`Actual Class` = ydataval, `Predicted Class` = predSVM)

    # Error and accuracy
    E_val_svm[j] <- sum(ydataval != predSVM)/length(ydataval)
    #Acc_SVM = 1 - E_SVM
   }

  ## Cross-validation error SVM
  E_CV_svm[k,i] <- mean(E_val_svm)
  }
}
## find cost and kernal which gave lowest CV error
indssvm <- which(E_CV_svm==min(E_CV_svm))
# number of trees
print(kern[indssvm[1]])
#cost
print(C[indssvm[1]])

## data frame of results for svm CV
df_SVM <- data.frame(100*E_CV_svm)
rownames(df_SVM) <- kern
colnames(df_SVM) <- C
library(knitr)
kable(df_SVM)

## Predict on test data
SVM_fntest <- svm(factor(ytrain) ~ ., data=X, type='C-classification', kernel=kern[indssvm[1]], cost=C[

pred_svmtest <- as.matrix(predict(SVM_fntest,newdata = testdata[,2:785]))


######################
# 4. Neural Network ##
######################
```

```
# Fit a neural network 1 hidden layers
XtrainNN <- X[1:(N*0.8),]
ytrainNN <- y[1:(N*0.8)]
Xval <- tail(X,N*0.2)
yval <- tail(y,N*0.2)
 nnet_model <- nnet(XtrainNN ,ytrainNN, size = 40,  MaxNWts=1000000 )
predresults <- predict(nnet_model,newdata=Xval)
results <- table(round(predresults),yval)
# ### gives sum(diag(results))/length(y) = 15.8% error

pred_NNtest <- predict(nnet_model,newdata=testdata[,2:785])
pred_NNtest <- round(pred_NNtest)
# #sum(round(Predicted)!=yval)
#results <- data.frame(actual = yval, predprob = predNN$net.result, prediction =  round(predNN$net.resu



#
# ptm <- proc.time()
# NN <- neuralnet(as.formula(paste('ytrainNN ~ ',paste(paste('Pixel_',1:784,sep=''), collapse='+'), sep
# proc.time() - ptm
#
# ## Predictions on validation set
# predNN <- compute(NN,Xval)
#
# ## Confusion matrix
# CM_NN <- table(round(predNN$net.result),yval)
#
# ## Accuracy/Validation error
# (sum(diag(CM_NN)))/sum(CM_NN)
#
# ## predictions on test data
# NN <- neuralnet(as.formula(paste('ytrain ~ ',paste(paste('Pixel_',1:784,sep=''), collapse='+'), sep='
# pred_NNtest <- compute(NN,testdata[,2:785])


######################
# 5. Random Forests ##
######################
## Cv to choose number of trees for random forest
numtreerf <- c(10,30,60,100,200,300) #nunber of tress to use for random forest

## R-fold Cross-validation
R <- 5
folds <- cut(seq(1,nrow(Xtrain)),breaks=R,labels=FALSE)
E_val_rf <- cbind(rep(NA,R))
E_in_rf <- cbind(rep(NA,R))
E_CV_rf <- cbind(rep(NA,length(numtreerf)))

for (i in 1:length(numtreerf)){
## Split data into 10 folds
for(j in 1:R){
  #Segement your data by fold using the which() function
```

```
  Inds <- which(folds==j,arr.ind=TRUE) # indices of current validation fold
  ValData <- Xtrain[Inds,]
  TrainData <- Xtrain[-Inds,]
  ydatatrain <- ytrain[-Inds,]
  ydataval <- ytrain[Inds,]

  ## Implement Random Forest using randomForest R package
  rf <- randomForest(factor(ydatatrain) ~ .,data=TrainData,type="class",ntree=numtreerf[i],importance=TI

  ## In sample error
  E_in_rf <- tail(rf$err.rate,1)[1]

  ## Predict on the validation data
  rfPredval = predict(rf, newdata=ValData)

  # Confusion matrix- where predicted values agree and disagree with target y
  CM_rf = table(rfPredval, ydataval)

  ## acuracy and error of the prediction on val. data
  accuracy_rf = (sum(diag(CM_rf)))/sum(CM_rf)
  E_val_rf[j] = 100*(1-accuracy_rf) #validation error
  }
 ## Cross-validation error RF
 E_CV_rf[i] <- mean(E_val_rf)
}
nooftreerf <- which(E_CV_rf==min(E_CV_rf))
print(numtreerf[nooftreerf])

## data frame of results for svm CV
df_rf <- data.frame(E_CV_rf)
rownames(df_rf) <- numtreerf
colnames(df_rf) <- c("Trees","Cross Validation Error")
library(knitr)
kable(df_rf)

## Train on all data using chosen number of trees
rffinal <- randomForest(factor(ytrain) ~ .,data=X,type="class",ntree=numtreerf[nooftreerf],importance=TI

## Predict on test data
Pred_rftest <- as.matrix(predict(rffinal, newdata=testdata[,2:785]))


## plot the mean square error of the forest object as function of no. of trees
plot(1:numtreerf[nooftreerf],rffinal$err.rate[,1],type="l",col="black",xlab = "No. of Trees",ylab="Erro
lines(1:numtreerf[nooftreerf],rffinal$err.rate[,2],type="l",lty="dashed",col="green") #for classificati
lines(1:numtreerf[nooftreerf],rffinal$err.rate[,3],type="l",lty="dashed",col="red")  #for classificatio
legend("top", cex =0.5,legend=c(0,"OOB",1), colnames("Even","Odd"),lty=c(2,1,2), col=c("green","black",


#############################################
# 6. Generalized Boosted Regression Models ##
#############################################
```

```
# ## Implement R GBM function
# fitControl <- trainControl(method = "repeatedcv", number = 10, repeats = 1)
# GBM = train(Xtrain,ytrain, method= 'gbm', trControl=fitControl)
#
# ## Predict on the validation data
# pred_gbm = predict(GBM, newdata=Xval)
#
# ## Confusion matrix
# CM_GBM = table(round(pred_gbm), yval)
#
# ## acuracy and error of the prediction on val. data
# accuracy_GBM = (sum(diag(CM_GBM)))/sum(CM_GBM)
# E_val_gbm = 100*(1-accuracy_GBM) #validation error

# ## Predict on test data
# Pred_gbm_test <- predict(GBM, newdata=testdata)

## USe CV to find number of trees
ptm <- proc.time()
numtrees_gbm <- c(100,400,800,1500)
shrinkparam = c(0.1,0.01,0.001)
## R-fold Cross-validation
R <- 5
folds <- cut(seq(1,nrow(Xtrain)),breaks=R,labels=FALSE)
E_val_gbm <- cbind(rep(NA,R))
E_CV_gbm <- matrix(NA,length(numtrees_gbm),length(shrinkparam))


for (k in 1:length(shrinkparam)){
 for (i in 1:length(numtrees_gbm)){
  ## Split data into 10 folds
  for(j in 1:R){
    #Segement your data by fold using the which() function
    Inds <- which(folds==j,arr.ind=TRUE) # indices of current validation fold
    ValData <- data.frame(Xtrain[Inds,])
    TrainData <- data.frame(Xtrain[-Inds,])
    ydatatrain <- ytrain[-Inds,]
    ydataval <- ytrain[Inds,]

    boost <- gbm(ydatatrain ~ ., data=TrainData, distribution="bernoulli", n.trees=numtrees_gbm[i],shrin
    proc.time() - ptm

    pred_GBM = predict(boost, newdata=ValData,n.trees = numtrees_gbm[i],type = "response")

    CM_GBM = table(round(pred_GBM),ydataval)

    ## acuracy and error of the prediction on val. data
    accuracy_GBM = (sum(diag(CM_GBM)))/length(ydataval)
    E_val_gbm[j] = 100*(1-accuracy_GBM) #validation error
  }
  ## Cross-validation error RF
  E_CV_gbm[i,k] <- mean(E_val_gbm)
 }
```

```r
}
proc.time() - ptm
## find lowest cv error for given number of trees
indsgbm <- which(E_CV_gbm==min(E_CV_gbm))
# number of trees
print(numtrees_gbm[indsgbm[2]])
#shrinkage
print(shrinkparam[indsgbm[1]])

## data frame of results for svm CV
df_gbm <- data.frame(E_CV_gbm)
rownames(df_gbm) <- numtrees_gbm
colnames(df_gbm) <- shrinkparam
library(knitr)
kable(df_gbm)

## Predict on test set using full training data with chosen number of trees from CV

boost <- gbm(as.formula(paste('ytrain ~ ',paste(paste('Pixel_',1:784,sep=''), collapse='+'), sep='')), d
pred_GBMtest = as.matrix(round(predict(boost, newdata=testdata[,2:785],n.trees =numtrees_gbm[indsgbm[2]]


############################
# 7. Classification trees ##
############################

## Stopping criteria
stopcrit <- rpart.control(minbucket = 5, minsplit = 10)

## cross validation
R <- 5
E_val_tree <- cbind(rep(NA,R))
E_CV_tree <- cbind(rep(NA,R))
## Split data into R folds
for(j in 1:R){
  #Segement your data by fold using the which() function
  Inds <- which(folds==j,arr.ind=TRUE) # indices of current validation fold
  ValData <- data.frame(Xtrain[Inds,])
  TrainData <- data.frame(Xtrain[-Inds,])
  ydatatrain <- ytrain[-Inds,]
  ydataval <- ytrain[Inds,]

  ## Implement 'rpart' function
  fulltree <- rpart(ydatatrain ~ ., method = "class", data = TrainData,control = stopcrit)

  ## Predict on the validation data
  pred_tree <- predict(fulltree, newdata = ValData, type = "class")
  E_val_tree[j] <- sum(ydataval != pred_tree)/length(ydataval)
}
E_CV_tree <- 100*mean(E_val_tree)


## Implement 'rpart' function on full data set
```

```r
stopcrit <- rpart.control(minbucket = 5, minsplit = 10)
fulltree <- rpart(ytrain ~ ., method = "class", data = X,control = stopcrit)
## Plot the tree
# plot(fulltree)
# text(fulltree, cex = 0.5)
library(rattle)
fancyRpartPlot(fulltree, tweak=1.5,main="Recursive Partitioning Tree")

## Predict on test data
pred_treetest <- as.matrix(predict(fulltree, newdata = testdata[,2:785], type = "class"))


################################
### Predictions on test data ##
Predictions <- cbind(pred_svmtest,Pred_rftest,pred_GBMtest,pred_treetest,pred_NNtest,as.matrix(pred_glm
Pred <- data.frame(Predictions)
colnames(Pred) <- c("SVM","Random Forest","Boosting","Tree","Neural Network","Logistic Regression")

## Write to .csv
write.csv(Pred, "~/R/ML/Assignments/Project/Predictions.csv")

#write.csv(pred_svmtest, "~/R/ML/Assignments/Project/Predictions.csv")

### EOF
```