title: "Untitled"

output: pdf_document

###############Final HW

######Task 1

########Part a variance calculation,score and boxplots

digits=pendigits$class load("pendigits.Rdata") pendigits$class<-as.factor(pendigits$class) pendigits_predictors<-pendigits[,-17] pendigits_predictors_scaled <- scale(pendigits_predictors) pendigits_prcomp <- prcomp(pendigits_predictors_scaled,scale= FALSE,center=FALSE) percentage_var <- (pendigits_prcomp$sdev)^2 percentage_var_exp <- (percentage_var/(sum(percentage_var)))*100 plot(cumsum(percentage_var_exp), xlab = "Principal Components",ylab = "Cumulative Proportion of Variance Explained",type = "b") abline(h=80.09,col='red',v=5) abline(h=92.297,col='red',v=8) barplot(percentage_var_exp, main = "PC variance explained", ylab = "% variation explained", xlab = "PCs",ylim=c(min(percentage_var_exp),max(percentage_var_exp)))

plot(pendigits_prcomp$rotation[,1], pendigits_prcomp$rotation[,2], xlab = "PC1 scores", ylab = "PC2 scores")

col <- rep("black", length(pendigits[,1])) pendigits_response<-pendigits$class c0 <- which(pendigits_response == 0) c1 <- which(pendigits_response == 1) c2 <- which(pendigits_response == 2) c3 <- which(pendigits_response == 3) c4 <- which(pendigits_response == 4) c5 <- which(pendigits_response == 5) c6 <- which(pendigits_response == 6) c7 <- which(pendigits_response == 7) c8 <- which(pendigits_response == 8) c9 <- which(pendigits_response == 9) col[c0] <- "black" col[c1] <- "green" col[c2] <- "pink" col[c3] <- "red" col[c4] <- "yellow" col[c5] <- "blue" col[c6] <- "orange" col[c7] <- "skyblue" col[c8] <- "purple" col[c9] <- "brown" par(mfrow=c(2,2)) biplot(pendigits_prcomp, choices = 1:2, scale=FALSE) biplot(pendigits_prcomp, choices = 2:3, scale=FALSE) biplot(pendigits_prcomp, choices = 3:4, scale=FALSE) biplot(pendigits_prcomp, choices = 4:5, scale=FALSE) par(mfrow=c(2,2)) biplot(pendigits_prcomp, choices = c(1,3), scale=FALSE) biplot(pendigits_prcomp, choices = c(2,5), scale=FALSE) biplot(pendigits_prcomp, choices = c(1,6), scale=FALSE) biplot(pendigits_prcomp, choices = c(5,8), scale=FALSE) library(plotly) plot_ly(x=pendigits_prcomp$x[,1], y=pendigits_prcomp$x[,2], z=pendigits_prcomp$x[,3], type="scatter3d", mode="markers", color=pendigits_response)

############Task1 Part B FItting KNN with principal components and without principal components

set.seed(2) require(class) train = sample(1:nrow(pendigits), nrow(pendigits)*.80) test = -train pendigits_train <- pendigits[train,-17] pendigits_test <- pendigits[test,-17] pendigits_train_class = pendigits[train,17] pendigits_test_class = pendigits[test,17] predicted = knn(pendigits_train, pendigits_test, pendigits_train_class, k = 1) table(predicted, pendigits_test_class) mean(predicted == pendigits_test_class) library(gmodels)

knn_acc <- list() for (i in 1:20) { knn_pred <- knn(train = pendigits_train, test = pendigits_test, cl = pendigits_train_class, k = i) knn_acc[as.character(i)] = mean(knn_pred == pendigits_test_class) } knn_acc <- unlist(knn_acc) data_frame(knn_acc = knn_acc) %>% mutate(k = row_number()) %>% ggplot(aes(k, knn_acc)) + geom_col(aes(fill = k == which.max(knn_acc))) + labs(x = 'K', y = 'Accuracy without Principal Components', title = 'KNN Accuracy for different values of K') + scale_x_continuous(breaks = 1:20) + scale_y_continuous(breaks = round(c(seq(0.90, 0.94, 0.01), max(knn_acc)), digits = 3)) + geom_hline(yintercept = max(knn_acc), lty = 2) + coord_cartesian(ylim = c(min(knn_acc), max(knn_acc))) + guides(fill = FALSE)

#

set.seed(2) require(class) train = sample(1:nrow(pendigits), nrow(pendigits)*.80) test = -train pendigits_train <- pendigits[train,-17] pendigits_test <- pendigits[test,-17] pendigits_train_class = pendigits[train,17] pendigits_test_class = pendigits[test,17] rotate<-pendigits_prcomp$rotation[,1:5] train.prcomp <- pendigits_prcomp$x[train,1:5] test.prcomp <-predict(pendigits_prcomp,pendigits[test,]) test.prcomp<-test.prcomp[,1:5] pendigits_trainFinal = as.matrix(pendigits_train)%*%(rotate) pendigits_test_Final = as.matrix(pendigits_test) %*% (rotate) predicted = knn(train.prcomp, test.prcomp, pendigits_train_class, k = 1) predicted = knn(pendigits_trainFinal, pendigits_test_Final, pendigits_train_class, k = 1) table(predicted, pendigits_test_class) mean(predicted == pendigits_test_class) head(pendigits_trainFinal) knn_acc <- list() for (i in 1:20) { knn_pred <- knn(train = train.prcomp, test = test.prcomp, cl = pendigits_train_class, k = i) knn_acc[as.character(i)] = mean(knn_pred == pendigits_test_class) }

knn_acc <- unlist(knn_acc) data_frame(knn_acc = knn_acc) %>% mutate(k = row_number()) %>% ggplot(aes(k, knn_acc)) + geom_col(aes(fill = k == which.max(knn_acc))) + labs(x = 'K', y = 'Accuracy with Principal Components', title = 'KNN Accuracy for different values of K') + scale_x_continuous(breaks = 1:20) + scale_y_continuous(breaks = round(c(seq(0.90, 0.94, 0.01), max(knn_acc)), digits = 3)) + geom_hline(yintercept = max(knn_acc), lty = 2) + coord_cartesian(ylim = c(min(knn_acc), max(knn_acc))) + guides(fill = FALSE) knn_acc <- list() for (i in 1:20) { knn_pred <- knn(train =pendigits_trainFinal, test =pendigits_test_Final, cl = pendigits_train_class, k = i) knn_acc[as.character(i)] = mean(knn_pred == pendigits_test_class) }

knn_acc <- unlist(knn_acc) data_frame(knn_acc = knn_acc) %>% mutate(k = row_number()) %>% ggplot(aes(k, knn_acc)) + geom_col(aes(fill = k == which.max(knn_acc))) + labs(x = 'K', y = 'Accuracy with Principal Components', title = 'KNN Accuracy for different values of K') + scale_x_continuous(breaks = 1:20) + scale_y_continuous(breaks = round(c(seq(0.90, 0.94, 0.01), max(knn_acc)), digits = 3)) + geom_hline(yintercept = max(knn_acc), lty = 2) + coord_cartesian(ylim = c(min(knn_acc), max(knn_acc))) + guides(fill = FALSE)

#################Task1 Part C fiiting Random Forest without Principal Components

set.seed(2) train = sample(1:nrow(pendigits), nrow(pendigits)*.80) test = -train pendigits_train <- pendigits[train,] pendigits_test <- pendigits[test,] pendigits_train_class = pendigits_train[,17] pendigits_test_class = pendigits_test[,17] rf.fit <- randomForest(pendigits_train_class~., data = pendigits_train, ntree = 1000) y_hat <- predict(rf.fit, newdata = pendigits_test, type = "response") y_true<-pendigits_test_class x<- which(y_hat!=y_true)

miss_rf<-length(x)/length(y_true) plot(rf.fit)

#################Task1 Part C Random Forest with Principal Components

pendigits_train <- pendigits[train,-17] pendigits_test <- pendigits[test,-17] pendigits_train_class = pendigits[train,17] pendigits_test_class = pendigits[test,17] rotate<-pendigits_prcomp$rotation[,1:5] pendigits_trainFinal = as.matrix(pendigits_train)%*%(rotate) pendigits_test_Final = as.matrix(pendigits_test) %*% (rotate) pendigits_trainFinal=as.data.frame(pendigits_trainFinal,pendigits_train_class) pendigits_test_Final=as.data.frame(pendigits_test_Final,pendigits_train_class) rf.fit <- randomForest(pendigits_train_class~., data = pendigits_trainFinal, n.tree = 1000) y_hat <- predict(rf.fit, newdata = pendigits_test_Final, type = "response") y_true<-pendigits_test_class x<- which(y_hat!=y_true)

miss_rf<-length(x)/length(y_true)

################Task 2 FItting cleveland data with Full Tree and Pruned Tree Model

```
 set.seed(2)

train = sample(1:nrow(cleveland_modified), nrow(cleveland_modified)*.80)
test = -train
cleveland_train <- cleveland_modified[train,-14]
cleveland_test <- cleveland_modified[test,-14]
cleveland_train_class <- cleveland_modified$diag1[train]
cleveland_test_class <- cleveland_modified$diag1[test]
model.controls <- rpart.control(minbucket = 2, minsplit = 4, xval = 10, cp = 0)
fit_cleveland <- rpart(cleveland_train_class~.,data = cleveland_train,method="class",control = model.controls)
#plot(fit_cleveland$cptable[,4], type = "o", lty = 1, col = "blue",main = "Cp for model selection", ylab = "cv error")
min_cp = which.min(fit_cleveland$cptable[,4])
pruned_fit_cleveland <- prune(fit_cleveland, cp = fit_cleveland$cptable[min_cp, 1])
graphics.off()
x11()
plot(fit_cleveland,  main = "Full_Tree")
text(fit_cleveland,use.n=TRUE,cex = .5)
plot(pruned_fit_cleveland,  main = "Pruned_Tree")
text(pruned_fit_cleveland,use.n=TRUE,cex = .5)

predict.full_tree <- predict(fit_cleveland, cleveland_test)
predict.pruned_tree <- predict(pruned_fit_cleveland, cleveland_test)
predict.full_tree_frame<-as.data.frame(predict.full_tree)
predict.pruned_tree_frame<-as.data.frame(predict.pruned_tree)
y_true_test <- cleveland_test_class

predicted<-predict.full_tree_frame %>%
  mutate(diag1 = case_when(
    predict.full_tree_frame$buff > predict.full_tree_frame$sick ~ "buff",
    predict.full_tree_frame$buff < predict.full_tree_frame$sick ~ "sick"
    ))
x<- which(predicted[,3]!=y_true_test)
length(x)
error<-length(x)/length(predicted[,3])
summary(predict.full_tree)

predict.pruned_tree_frame<-as.data.frame(predict.pruned_tree)
predicted_prune_tree<-predict.pruned_tree_frame %>%
  mutate(diag1 = case_when(
    predict.pruned_tree_frame$buff > predict.pruned_tree_frame$sick ~ "buff",
    predict.pruned_tree_frame$buff < predict.pruned_tree_frame$sick ~ "buff"
    ))
x<- which(predicted_prune_tree[,3]!=y_true_test)
error<-length(x)/length(predicted_prune_tree[,3])
```

######## Task 2 Fitting cleveland data with Random forest and fine tuning the model

load("cleveland.Rdata") cleveland_modified<-cleveland[,-15] set.seed(2)

train = sample(1:nrow(cleveland_modified), nrow(cleveland_modified)*.80) test = -train cleveland_train <- cleveland_modified[train,] cleveland_test <- cleveland_modified[test,] cleveland_train_class <- cleveland_train$diag1 cleveland_test_class <- cleveland_test$diag1 rf.fit <- randomForest(cleveland_train_class~., data = cleveland_train,importance=TRUE,ntree = 1000)

x11() varImpPlot(rf.fit)

importance(rf.fit)

y_hat <- predict(rf.fit, newdata = cleveland_test, type = "response") y_hat <- as.numeric(y_hat)-1 y_true <- as.numeric(cleveland_test_class)-1 misclass_rf <- sum(abs(y_true- y_hat))/length(y_hat)

Manual tuning of random forest

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid") tunegrid <- expand.grid(.mtry=c(1,10)) modellist <- list() for (ntree in c(1000, 1500, 2000, 2500)) { set.seed(2) fit <- train(diag1~., data=cleveland_train, method="rf", metric='Accuracy', tuneGrid=tunegrid, trControl=control, ntree=ntree) key <- toString(ntree) modellist[[key]] <- fit }

customRF <- list(type = "Classification", library = "randomForest", loop = NULL) customRF$parameters <- data.frame(parameter = c("mtry", "ntree"), class =

rep("numeric", 2), label = c("mtry", "ntree")) customRF$grid <- function(x, y, len = NULL, search = "grid") {} customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) { randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...) } customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL) predict(modelFit, newdata) customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL) predict(modelFit, newdata, type = "prob") customRF$sort <- function(x) x[order(x[,1]),] customRF$levels <- function(x) x$classes control <- trainControl(method="repeatedcv", number=10, repeats=3) tunegrid <- expand.grid(.mtry=c(1:15), .ntree=c(1000, 1500, 2000, 2500)) set.seed(seed) custom <- train(Class~., data=dataset, method=customRF, metric=metric, tuneGrid=tunegrid, trControl=control) summary(custom) plot(custom)

#######Task 2 Fitting cleveland data with Neural Networks and fine tuning the model

#########As neural networks use activation functions between -1 and +1, itÃ¢â‚¬â„¢s important to scale down variables down

#

cleveland_modified$age <-(cleveland_modified$age-min(cleveland_modified$age)) / (max(cleveland_modified$age)-min(cleveland_modified$age)) cleveland_modified$trestbps <-(cleveland_modified$trestbps-min(cleveland_modified$trestbps)) / (max(cleveland_modified$trestbps)-min(cleveland_modified$trestbps)) cleveland_modified$chol <-(cleveland_modified$chol-min(cleveland_modified$chol)) / (max(cleveland_modified$chol)-min(cleveland_modified$chol)) cleveland_modified$thalach <-(cleveland_modified$thalach-min(cleveland_modified$thalach)) / (max(cleveland_modified$thalach)-min(cleveland_modified$thalach)) cleveland_modified$oldpeak <-(cleveland_modified$oldpeak-min(cleveland_modified$oldpeak)) / (max(cleveland_modified$oldpeak)-min(cleveland_modified$oldpeak)) cleveland_modified$ca <-(cleveland_modified$ca-min(cleveland_modified$ca)) / (max(cleveland_modified$ca)-min(cleveland_modified$ca))

##############In order to represent factor variables, we need to convert them into dummy variables.

A dummy variable takes the N distinct values and converts it into N-1 variables.

We use N-1 because the final value is represented by all dummy values set to zero

table(cleveland_modified$gender) head(model.matrix(~gender, data=cleveland_modified)) cleveland_modified_matrix<-model.matrix(~age+gender+cp+trestbps+chol+fbs+restecg+thalach+exang+oldpeak+slope+ca+thal+diag1, data=cleveland_modified)
colnames(cleveland_modified_matrix)
col_list <- paste(c(colnames(cleveland_modified_matrix[,-c(1,20)])),collapse="+")
col_list <- paste(c("diag1sick~",col_list),collapse="")
f <- formula(col_list)

library(neuralnet) set.seed(2)

train = sample(1:nrow(cleveland_modified_matrix), nrow(cleveland_modified_matrix)*.80) test = -train cleveland_train <- cleveland_modified_matrix[train,] cleveland_test <- cleveland_modified_matrix[test,] output<- compute(nmodel, cleveland_modified_matrix[,-c(1,20)],rep=1) diag1_train<-cleveland_train[,20]

nmodel <- neuralnet(f,data=cleveland_train,hidden=2,err.fct = "ce", linear.output=FALSE) nmodel$result.matrix pred <- predict(nmodel, newdata = cleveland_train) y_hat_train <- round(pred) train_err <- length(which(diag1 != y_hat_train))/length(y_hat_train) pred <- predict(nmodel, newdata = cleveland_test) y_hat_test <- round(pred) diag1_test<-cleveland_test[,20] diag1_train<-cleveland_train[,20] test_err <- length(which(diag1_test != y_hat_test))/length(y_hat_test)
table(diag1 #Neural Network

plot(nmodel) train_err_store <- c() test_err_store <- c() for (i in 1:4){

```
 # fit neural network with "i" neurons
nmodel <- neuralnet(f, data = cleveland_train,
hidden = i, stepmax = 10^9, err.fct = "ce", linear.output = FALSE)

# calculate the train error
pred <- predict(nmodel, newdata = cleveland_train)
y_hat_train <- round(pred)
train_err <- length(which(diag1_train != y_hat_train))/length(y_hat_train)
train_err_store <- c(train_err_store, train_err) #store the error at each iteration

pred <- predict(nmodel, newdata = cleveland_test)
y_hat_test <- round(pred)
test_err <- length(which(diag1_test != y_hat_test))/length(y_hat_test)
test_err_store <- c(test_err_store, test_err) #store the error at each iteration
```

}

Test the resulting output

#######Creating confusion matrix

Confusion Matrix

roundedresults<-sapply(results,round,digits=0) roundedresultsdf=data.frame(roundedresults) attach(roundedresultsdf) table(actual,prediction)
mean(actual==prediction)

##################Task3 fitting the data on SVM with different kernels

```
 require(ISLR); require(tidyverse); require(ggthemes)
require(caret); require(e1071)
set.seed(1)

inTrain <- sample(nrow(OJ), 800, replace = FALSE)

training <- OJ[inTrain,]
testing <- OJ[-inTrain,]
```

svm_linear <- svm(Purchase ~ ., data = training, kernel = 'linear', cost = 0.01) summary(svm_linear)

> postResample(predict(svm_linear, training), training$Purchase) Accuracy Kappa 0.8250000 0.6313971 1-0.8250000 [1] 0.175 postResample(predict(svm_linear, testing), testing$Purchase) Accuracy Kappa 0.8222222 0.6082699 1-0.8222222 [1] 0.1777778

###Tuning SVM and calculating test and train error

> svm_linear_tune <- train(Purchase ~ ., data = training, method = 'svmLinear2', trControl = trainControl(method = 'cv', number = 10), preProcess = c('center', 'scale'), tuneGrid = expand.grid(cost = seq(0.01, 10, length.out = 20)))

postResample(predict(svm_linear_tune, training), training$Purchase) postResample(predict(svm_linear_tune, testing), testing$Purchase)

#########Training with a polynomial Kernel

svm_poly <- svm(Purchase ~ ., data = training, method = 'polynomial', degree = 2, cost = 0.01) summary(svm_poly)

postResample(predict(svm_poly, training), training$Purchase) postResample(predict(svm_poly, testing), testing$Purchase)

#########Tuning with SVM polynomial kernel

svm_poly_tune <- train(Purchase ~ ., data = training, method = 'svmPoly', trControl = trainControl(method = 'cv', number = 10), preProcess = c('center', 'scale'), tuneGrid = expand.grid(degree = 2, C = seq(0.01, 10, length.out = 20), scale = TRUE))

postResample(predict(svm_poly_tune, training), training$Purchase) postResample(predict(svm_poly_tune, testing), testing$Purchase)

#########Training with a radial Kernel

svm_radial <- svm(Purchase ~ ., data = training, method = 'radial', cost = 0.01)

> summary(svm_radial)

postResample(predict(svm_radial, training), training$Purchase) postResample(predict(svm_radial, testing), testing$Purchase)

###########Tuning SVM radial Kernel

svm_radial_tune <- train(Purchase ~ ., data = training, method = 'svmRadial', trControl = trainControl(method = 'cv', number = 10), preProcess = c('center', 'scale'), tuneGrid = expand.grid(C = seq(0.01, 10, length.out = 20), sigma = 0.05691)) postResample(predict(svm_radial_tune, training), training$Purchase) postResample(predict(svm_radial_tune, testing), testing$Purchase)