CSE455/CSE552 – Machine Learning (Spring 2016)

Homework #1 Report

Murat ALTUNTAŞ (111044043)

Part 1:

Code:

```
#########################################################################
############################### Functions ###############################
#########################################################################
normalize <- function(x) { return ((x - min(x)) / (max(x) - min(x))) }
euc_dist <- function(x1, x2) { return (sqrt(sum((x1 - x2) ^ 2))) }
manh_dist <- function(p,q){ sum(abs(p-q)) }
Mode <- function(x) {
  ls <- unique(x)
  ls[which.max(tabulate(match(x, ls)))]
}
CreateTable <- function(x1,x2) {
        total <- matrix(0,36,36)
        for(i in 1:34) {
                total[x1[i],x2[i]] <- total[x1[i],x2[i]] + 1
        }
        return(total)
}
myKnnEuc <- function(train,test,cl,k) {
        train_row <- nrow(train) # train dataset row sayısı
        test_row <- nrow(test)   # test dataset row sayısı
        eucArr <- 1:train_row    # eucledian hesaplarının row sayısı
        labelArr <- 1:k          # labeller
        result <- 1:test_row     # sonuc arrayi
        for(i in 1:test_row) {
                for(j in 1:train_row) {
                        eucArr[j] <- euc_dist(train[j,],test[i,])
                }
```

```
                for(l in 1:k) {
                        labelArr[l] <- cl[match(l,rank(eucArr))]
                }
                result[i] <- Mode(labelArr)
        }
        return(result)
}
myKnnManh <- function(train,test,cl,k) {
        train_row <- nrow(train) # train dataset row sayısı
        test_row <- nrow(test)   # test dataset row sayısı
        eucArr <- 1:train_row    # eucledian hesaplarının row sayısı
        labelArr <- 1:k          # labeller
        result <- 1:test_row     # sonuc arrayi
        for(i in 1:test_row) {
                for(j in 1:train_row) {
                        eucArr[j] <- manh_dist(train[j,],test[i,])
                }
                for(l in 1:k) {
                        labelArr[l] <- cl[match(l,rank(eucArr))]
                }
                result[i] <- Mode(labelArr)
        }
        return(result)
}

#########################################################################
############################### iris (Eucledian) ########################
#########################################################################
#Randomly shuffle the data
iris<-iris[sample(nrow(iris)),]

#Create 10 equally size folds
folds <- cut(seq(1,nrow(iris)),breaks=10,labels=FALSE)
gp <- runif(nrow(iris))   # random siralama
iris <- iris[order(gp),]
```

```
totalMAtEuc <- matrix(0,3,3)
#Perform 10 fold cross validation
for(i in 1:10){
        iris_n <- as.data.frame(lapply(iris[,c(1,2,3,4)],normalize))
  #Segement your data by fold using the which() function
  testIndexes <- which(folds==i,arr.ind=TRUE)
  iris_test <- iris_n[testIndexes, ]
  iris_train <- iris_n[-testIndexes, ]
  #Use the test and train data partitions however you desire...
        #-- train ve test olarak ayırma --#
        iris_test_target <- iris[testIndexes, 5]   # 5 => kolon numarası
        iris_train_target <- iris[-testIndexes, 5]
        test_Euc <- myKnnEuc(iris_train, iris_test, iris_train_target,5)   # part 1
        #-- Cros Table --#
        totalMAtEuc <- as.matrix(table(iris_test_target, test_Euc)) + totalMAtEuc
}
print(totalMAtEuc)
CrossTable(totalMAtEuc, prop.chisq = FALSE)
cat("%", (sum(diag(totalMAtEuc)) / sum(totalMAtEuc) * 100))

#########################################################################
########################### leaf (Eucledian) ############################
#########################################################################

leaf <- read.csv("C:\\Users\\Murat\\Desktop\\ss\\ML\\HW\\HW1\\leaf.csv", header = FALSE)
leaf<-leaf[sample(nrow(leaf)),]

#Create 10 equally size folds
folds <- cut(seq(1,nrow(leaf)),breaks=10,labels=FALSE)
gp <- runif(nrow(leaf))   # random siralama
leaf <- leaf[order(gp),]
totalMAtEuc <- matrix(0,36,36)

#Perform 10 fold cross validation
for(i in 1:10){
```

```
        leaf_n <- as.data.frame(lapply(leaf[,2:16],normalize))


        #Segement your data by fold using the which() function
  testIndexes <- which(folds==i,arr.ind=TRUE)
  leaf_test <- leaf_n[testIndexes, ]
  leaf_train <- leaf_n[-testIndexes, ]


        #-- train ve test olarak ayırma --#
        leaf_train_target <- leaf[-testIndexes, 1]   # 1 => kolon numarası
        leaf_test_target <- leaf[testIndexes, 1]
        test_Euc <- myKnnEuc(leaf_train, leaf_test, leaf_train_target,5)   # part 1
        totalMAtEuc <- as.matrix(CreateTable(leaf_test_target, test_Euc)) + totalMAtEuc


}


cat("%", (sum(diag(totalMAtEuc)) / sum(totalMAtEuc) * 100))
write.table(totalMAtEuc, file="mydataEuc.txt", sep="\t")
```

Results:

İris Dataset

```
> print(totalMAtEuc)
                 test_Euc
iris_test_target  1  2  3
      setosa     50  0  0
      versicolor  0 48  2
      virginica   0  3 46
```

```
> CrossTable(totalMAtEuc, prop.chisq = FALSE)

   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  149
```

	test_Euc			
iris_test_target	1	2	3	Row Total
setosa	50	0	0	50
	1.000	0.000	0.000	0.336
	1.000	0.000	0.000	
	0.336	0.000	0.000	
versicolor	0	48	2	50
	0.000	0.960	0.040	0.336
	0.000	0.941	0.042	
	0.000	0.322	0.013	
virginica	0	3	46	49
	0.000	0.061	0.939	0.329
	0.000	0.059	0.958	
	0.000	0.020	0.309	
Column Total	50	51	48	149
	0.336	0.342	0.322	

```
> cat("%", (sum(diag(totalMAtEuc)) / sum(totalMAtEuc) * 100))
% 96.6443
```

Leaf Dataset

```
> cat("%", (sum(diag(totalMAtEuc)) / sum(totalMAtEuc) * 100))
% 57.35294
> write.table(totalMAtEuc, file="mydataEuc.txt", sep="\t")
```

(leaf datasının tablosu çok büyük olduğu için çıktıyı bir txt ye kaydedip ek te gönderdim.)

Comments:

Bu kısımda öklid uzaklığı ile knn algoritmasını implement ederek, iris ve leaf verilerini kullanarak test ettim. Cross validation yapmak için verileri 10'a böldüm. Çıkan sonuçları bir matriste toplayarak, doğru sonuçları tüm sonuçlara oranladım ve % 'lik başarıyı elde ettim.

Part 2:

Code:

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

############################## Functions ###############################

#########################################################################

normalize <- function(x) { return ((x - min(x)) / (max(x) - min(x))) }

euc_dist <- function(x1, x2) { return (sqrt(sum((x1 - x2) ^ 2))) }

manh_dist <- function(p,q){ sum(abs(p-q)) }

Mode <- function(x) {

 ls <- unique(x)

 ls[which.max(tabulate(match(x, ls)))]

}
```

```
CreateTable <- function(x1,x2) {
	total <- matrix(0,36,36)
	for(i in 1:34) {
		total[x1[i],x2[i]] <- total[x1[i],x2[i]] + 1
	}
	return(total)
}

myKnnEuc <- function(train,test,cl,k) {
	train_row <- nrow(train) # train dataset row sayısı
	test_row <- nrow(test)   # test dataset row sayısı
	eucArr <- 1:train_row    # eucledian hesaplarının row sayısı
	labelArr <- 1:k          # labeller
	result <- 1:test_row     # sonuc arrayi
	for(i in 1:test_row) {
		for(j in 1:train_row) {
			eucArr[j] <- euc_dist(train[j,],test[i,])
		}
		for(l in 1:k) {
			labelArr[l] <- cl[match(l,rank(eucArr))]
		}
		result[i] <- Mode(labelArr)
	}
	return(result)
}
myKnnManh <- function(train,test,cl,k) {
	train_row <- nrow(train) # train dataset row sayısı
	test_row <- nrow(test)   # test dataset row sayısı
	eucArr <- 1:train_row    # eucledian hesaplarının row sayısı
	labelArr <- 1:k          # labeller
	result <- 1:test_row     # sonuc arrayi
	for(i in 1:test_row) {
		for(j in 1:train_row) {
			eucArr[j] <- manh_dist(train[j,],test[i,])
		}
```

```
                for(l in 1:k) {
                        labelArr[l] <- cl[match(l,rank(eucArr))]
                }
                result[i] <- Mode(labelArr)
        }
        return(result)
}

#########################################################################
############################## iris (Manhattan) ##########################
#########################################################################
#Randomly shuffle the data
iris<-iris[sample(nrow(iris)),]

#Create 10 equally size folds
folds <- cut(seq(1,nrow(iris)),breaks=10,labels=FALSE)
gp <- runif(nrow(iris))   # random siralama
iris <- iris[order(gp),]
totalMAtManh <- matrix(0,3,3)
#Perform 10 fold cross validation
for(i in 1:10){
        iris_n <- as.data.frame(lapply(iris[,c(1,2,3,4)],normalize))
   #Segement your data by fold using the which() function
   testIndexes <- which(folds==i,arr.ind=TRUE)
   iris_test <- iris_n[testIndexes, ]
   iris_train <- iris_n[-testIndexes, ]
   #Use the test and train data partitions however you desire...

        #-- train ve test olarak ayırma --#
        iris_test_target <- iris[testIndexes, 5]   # 5 => kolon numarası
        iris_train_target <- iris[-testIndexes, 5]
        test_Manh <- myKnnManh(iris_train, iris_test, iris_train_target,5) # part 2
        #-- Cros Table --#
        totalMAtManh <- as.matrix(table(iris_test_target, test_Manh)) + totalMAtManh
}
```

```
print(totalMAtEuc)
CrossTable(totalMAtManh, prop.chisq = FALSE)
cat("%", (sum(diag(totalMAtManh)) / sum(totalMAtManh) * 100))

#########################################################################
############################## leaf (Manhattan) ##########################
#########################################################################

leaf <- read.csv("C:\\Users\\Murat\\Desktop\\ss\\ML\\HW\\HW1\\leaf.csv", header = FALSE)
leaf<-leaf[sample(nrow(leaf)),]

#Create 10 equally size folds
folds <- cut(seq(1,nrow(leaf)),breaks=10,labels=FALSE)
gp <- runif(nrow(leaf))   # random siralama
leaf <- leaf[order(gp),]
totalMAtManh <- matrix(0,36,36)

#Perform 10 fold cross validation
for(i in 1:10){
        leaf_n <- as.data.frame(lapply(leaf[,2:16],normalize))

        #Segement your data by fold using the which() function
  testIndexes <- which(folds==i,arr.ind=TRUE)
  leaf_test <- leaf_n[testIndexes, ]
  leaf_train <- leaf_n[-testIndexes, ]

        #-- train ve test olarak ayırma --#
        leaf_train_target <- leaf[-testIndexes, 1]   # 1 => kolon numarası
        leaf_test_target <- leaf[testIndexes, 1]
        test_Manh <- myKnnManh(leaf_train, leaf_test, leaf_train_target,5) # part 2

        totalMAtManh <- as.matrix(CreateTable(leaf_test_target, test_Manh)) + totalMAtManh
}
cat("%", (sum(diag(totalMAtManh)) / sum(totalMAtManh) * 100))
write.table(totalMAtEuc, "mydataManh.txt", sep="\t")
```

Results:

İris Dataset

```
> print(totalMAtManh)
                 test_Manh
iris_test_target  1  2  3
      setosa     50  0  0
      versicolor  0 46  4
      virginica   0  4 46

> CrossTable(totalMAtManh, prop.chisq = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  150


                  | test_Manh
iris_test_target |         1 |         2 |         3 | Row Total |
-----------------|-----------|-----------|-----------|-----------|
          setosa |        50 |         0 |         0 |        50 |
                 |     1.000 |     0.000 |     0.000 |     0.333 |
                 |     1.000 |     0.000 |     0.000 |           |
                 |     0.333 |     0.000 |     0.000 |           |
-----------------|-----------|-----------|-----------|-----------|
      versicolor |         0 |        46 |         4 |        50 |
                 |     0.000 |     0.920 |     0.080 |     0.333 |
                 |     0.000 |     0.920 |     0.080 |           |
                 |     0.000 |     0.307 |     0.027 |           |
-----------------|-----------|-----------|-----------|-----------|
       virginica |         0 |         4 |        46 |        50 |
                 |     0.000 |     0.080 |     0.920 |     0.333 |
                 |     0.000 |     0.080 |     0.920 |           |
                 |     0.000 |     0.027 |     0.307 |           |
-----------------|-----------|-----------|-----------|-----------|
    Column Total |        50 |        50 |        50 |       150 |
                 |     0.333 |     0.333 |     0.333 |           |
-----------------|-----------|-----------|-----------|-----------|


> cat("%", (sum(diag(totalMAtManh)) / sum(totalMAtManh) * 100))
% 94.66667
```

Leaf Dataset

```
> cat("%", (sum(diag(totalMAtManh)) / sum(totalMAtManh) * 100))
% 62.35294
>
> write.table(totalMAtEuc, "mydataManh.txt", sep="\t")
```

(leaf datasının tablosu çok büyük olduğu için çıktıyı bir txt ye kaydedip ek te gönderdim.)

Comments:

Bu kısımda Manhattan uzaklığı ile knn algoritmasını implement ederek, iris ve leaf verilerini kullanarak test ettim. Cross validation yapmak için verileri 10'a böldüm. Çıkan sonuçları bir matriste toplayarak, doğru sonuçları tüm sonuçlara oranladım ve % 'lik başarıyı elde ettim. İris verisi üzerinde Manhattan algoritması, Euclidian algoritmasına göre %2 düşük çıkarken, leaf verisi üzerinde Manhattan algoritması, Euclidian algoritmasına göre %5 daha başarılı sonuç verdi.

Part 3:

Code:

```
#########################################################################
############################### Library ################################
#########################################################################

library(kernlab)
library(pROC)
library(ROCR)
library(class)

#########################################################################
############################### Functions ###############################
#########################################################################
CreateTable <- function(x1,x2) {
	total <- matrix(0,36,36)
	for(i in 1:34) {
		total[x1[i],x2[i]] <- total[x1[i],x2[i]] + 1
	}
	return(total)
}

#########################################################################
############################### iris(linear svm) ########################
#########################################################################

#Randomly shuffle the data
iris<-iris[sample(nrow(iris)),]
```

```
#Create 10 equally size folds
folds <- cut(seq(1,nrow(iris)),breaks=10,labels=FALSE)
gp <- runif(nrow(iris))   # random siralama
iris <- iris[order(gp),]
totalMatLinear <- matrix(0,3,3)
all_predict_L <- c()
all_iris_test_target <- c()
for(i in 1:10){
        #-- train ve test olarak ayırma --#
        #Segement your data by fold using the which() function
        testIndexes <- which(folds==i,arr.ind=TRUE)
        iris_test <- iris[testIndexes, ]
        iris_train <- iris[-testIndexes, ]
        iris_train_target <- iris[-testIndexes, 5]   # 5 => kolon numarası
        iris_test_target <- iris[testIndexes, 5]
        all_iris_test_target <- c(all_iris_test_target, iris_test_target)
        filter_Linear <- ksvm(Species~.,data=iris_train, kernel="vanilladot", prob.model = TRUE)
        iris_type_L <- predict(filter_Linear,iris_test, type = "prob")
        all_predict_L <- rbind(all_predict_L, iris_type_L)
        iris_type_L <- predict(filter_Linear,iris_test, type = "response")
        table(iris_test_target, iris_type_L)
        totalMatLinear <- as.matrix(table(iris_test_target, iris_type_L)) + totalMatLinear
}

pred <- prediction( all_predict_L[,1], all_iris_test_target == 1)
perf <- performance( pred, "tpr", "fpr" )
xValues <- unlist(perf@x.values)
yValues <- unlist(perf@y.values)
aValues <- unlist(perf@alpha.values)
for(i in 2:length(levels(factor(iris[[5]])))){
  pred <- prediction( all_predict_L[,i], all_iris_test_target == i)
  perf <- performance( pred, "tpr", "fpr" )
  xValues <- xValues + unlist(perf@x.values)
  yValues <- yValues + unlist(perf@y.values)
```

```
  aValues <- aValues + unlist(perf@alpha.values)
}
perf@x.values <- list(xValues / length(levels(factor(iris[[5]]))))
perf@y.values <- list(yValues / length(levels(factor(iris[[5]]))))
perf@alpha.values <- list(aValues / length(levels(factor(iris[[5]]))))
plot( perf, col = "blue")
print(totalMatLinear)
cat("Linear SVM: %", (sum(diag(totalMatLinear)) / sum(totalMatLinear) * 100))

#########################################################################
########################## leaf(Linear svm) ##########################
#########################################################################

leaf <- read.csv("C:\\Users\\Murat\\Desktop\\ss\\ML\\HW\\HW1\\leaf.csv", header = FALSE)
#Randomly shuffle the data
leaf<-leaf[sample(nrow(leaf)),]

#set class as factor
leaf[[1]]=factor(leaf[[1]])
#Create 10 equally size folds
folds <- cut(seq(1,nrow(leaf)),breaks=10,labels=FALSE)
gp <- runif(nrow(leaf))   # random siralama
leaf <- leaf[order(gp),]
totalMatLinear <- matrix(0,36,36)
all_predict_L <- c()
all_leaf_test_target <- c()

for(i in 1:10){
        #-- train ve test olarak ayırma --#
        #Segement your data by fold using the which() function
        testIndexes <- which(folds==i,arr.ind=TRUE)
        leaf_test <- leaf[testIndexes, ]
        leaf_train <- leaf[-testIndexes, ]
        leaf_train_target <- leaf[-testIndexes, 1]   # 1 => kolon numarası
        leaf_test_target <- leaf[testIndexes, 1]
```

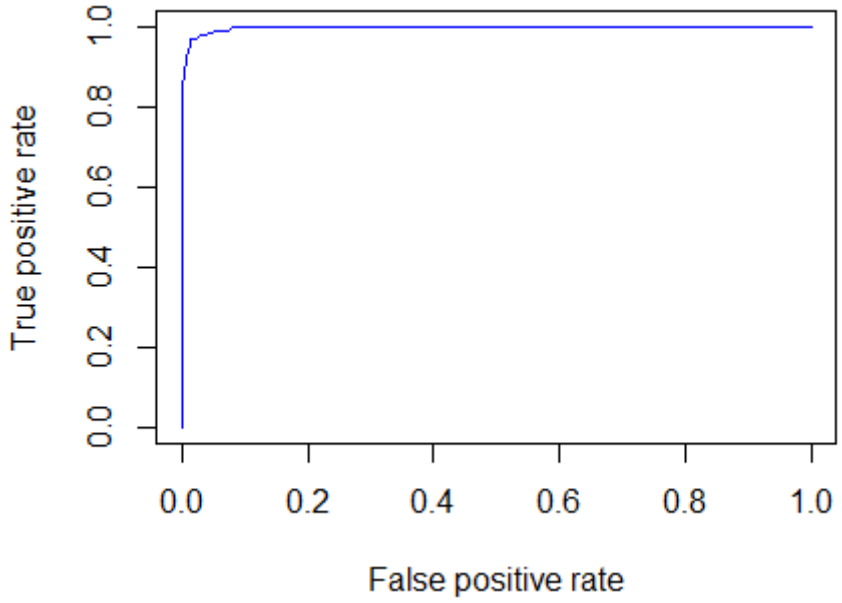
```
        all_leaf_test_target <- c(all_leaf_test_target, leaf_test_target)
        filter_Linear <- ksvm(V1~.,data=leaf_train,kernel="vanilladot", prob.model = TRUE)
        leaf_type_L <- predict(filter_Linear,leaf_test, type = "prob")
        all_predict_L <- rbind(all_predict_L, leaf_type_L)
        leaf_type_L <- predict(filter_Linear,leaf_test, type = "response")
        table(leaf_test_target, leaf_type_L)
        totalMatLinear <- as.matrix(CreateTable(leaf_test_target, leaf_type_L)) + totalMatLinear
}

pred <- prediction( all_predict_L[,1], all_leaf_test_target == 1)
perf <- performance( pred, "tpr", "fpr" )
xValues <- unlist(perf@x.values)
yValues <- unlist(perf@y.values)
aValues <- unlist(perf@alpha.values)
for(i in 2:length(levels(factor(leaf[[1]])))){
 pred <- prediction( all_predict_L[,i], all_leaf_test_target == i)
 perf <- performance( pred, "tpr", "fpr" )
 xValues <- xValues + unlist(perf@x.values)
 yValues <- yValues + unlist(perf@y.values)
 aValues <- aValues + unlist(perf@alpha.values)
}
perf@x.values <- list(xValues / length(levels(factor(leaf[[1]]))))
perf@y.values <- list(yValues / length(levels(factor(leaf[[1]]))))
perf@alpha.values <- list(aValues / length(levels(factor(leaf[[1]]))))
plot( perf, col = "red")
cat("Linear SVM: %", (sum(diag(totalMatLinear)) / sum(totalMatLinear) * 100))
write.table(totalMAtEuc, "Linear_SVM_Result.txt", sep="\t")
```

Results:

İris Dataset

```
> print(totalMatLinear)
                 iris_type_L
iris_test_target setosa versicolor virginica
      setosa         50          0         0
      versicolor      0         46         4
      virginica       0          1        49
> CrossTable(totalMatLinear, prop.chisq = FALSE)
```

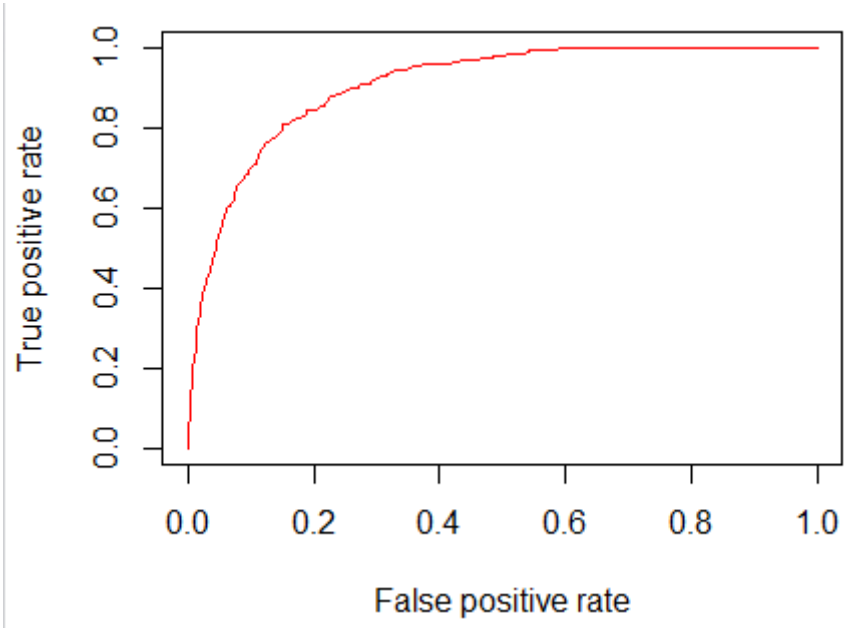
```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  150


                 | iris_type_L
iris_test_target |     setosa | versicolor |  virginica |  Row Total |
-----------------|------------|------------|------------|------------|
          setosa |         50 |          0 |          0 |         50 |
                 |      1.000 |      0.000 |      0.000 |      0.333 |
                 |      1.000 |      0.000 |      0.000 |            |
                 |      0.333 |      0.000 |      0.000 |            |
-----------------|------------|------------|------------|------------|
      versicolor |          0 |         46 |          4 |         50 |
                 |      0.000 |      0.920 |      0.080 |      0.333 |
                 |      0.000 |      0.979 |      0.075 |            |
                 |      0.000 |      0.307 |      0.027 |            |
-----------------|------------|------------|------------|------------|
       virginica |          0 |          1 |         49 |         50 |
                 |      0.000 |      0.020 |      0.980 |      0.333 |
                 |      0.000 |      0.021 |      0.925 |            |
                 |      0.000 |      0.007 |      0.327 |            |
-----------------|------------|------------|------------|------------|
    Column Total |         50 |         47 |         53 |        150 |
                 |      0.333 |      0.313 |      0.353 |            |
-----------------|------------|------------|------------|------------|


> cat("Linear SVM: %", (sum(diag(totalMatLinear)) / sum(totalMatLinear) *
100))
Linear SVM: % 96.66667
```



Leaf Dataset

```
> cat("Linear SVM: %", (sum(diag(totalMatLinear)) / sum(totalMatLinear) *
100))
Linear SVM: % 74.41176
> write.table(totalMatLinear, "Linear_SVM_Result.txt", sep="\t")
```

(leaf datasının tablosu çok büyük olduğu için çıktıyı bir txt ye kaydedip ek te gönderdim.)



Comments:

Bu kısımda kernlab kütüphanesindeki ksvm fonksiyonunu kullandım. İris ve leaf verilerinde kernel'ı "vanilladot" seçerek lineer svm'i kullandım. Cross validation yaparak veriyi 10'a böldüm ve sonuçları bir matriste topladım. Doğru sonuçları tüm sonuçlara oranladım ve % 'lik başarıyı elde ettim. Tahmin yüzdelerini ve gerçek labelleri bir listede tutarak ROC Curve çiziminde kullandım. Birden fazla sınıf

olduğu için her sınıfın ROC Curve'nün ortalamasını alarak ortalama bir ROC Curve çizdirdim.

Part 4:

Code:

```
#########################################################################
############################### Library ################################
#########################################################################

library(kernlab)
library(pROC)
library(ROCR)
library(class)

#########################################################################
############################### Functions ###############################
#########################################################################
CreateTable <- function(x1,x2) {
	total <- matrix(0,36,36)
	for(i in 1:34) {
		total[x1[i],x2[i]] <- total[x1[i],x2[i]] + 1
	}
	return(total)
}

#########################################################################
###################### iris(polynomial svm) #############################
#########################################################################

#Randomly shuffle the data
iris<-iris[sample(nrow(iris)),]

#Create 10 equally size folds
folds <- cut(seq(1,nrow(iris)),breaks=10,labels=FALSE)
gp <- runif(nrow(iris))   # random siralama
iris <- iris[order(gp),]
```

```
totalMatPoly <- matrix(0,3,3)
all_predict_P <- c()
all_iris_test_target <- c()
for(i in 1:10){
        #-- train ve test olarak ayırma --#
        #Segement your data by fold using the which() function
        testIndexes <- which(folds==i,arr.ind=TRUE)
        iris_test <- iris[testIndexes, ]
        iris_train <- iris[-testIndexes, ]
        iris_train_target <- iris[-testIndexes, 5]   # 5 => kolon numarası
        iris_test_target <- iris[testIndexes, 5]
        all_iris_test_target <- c(all_iris_test_target, iris_test_target)
        filter_Poly <- ksvm(Species~.,data=iris_train, kernel="polydot", prob.model = TRUE)
        iris_type_P <- predict(filter_Poly,iris_test, type = "prob")
        all_predict_P <- rbind(all_predict_P, iris_type_P)
        iris_type_P <- predict(filter_Poly,iris_test, type = "response")
        table(iris_test_target, iris_type_P)
        totalMatPoly <- as.matrix(table(iris_test_target, iris_type_P)) + totalMatPoly
}

pred <- prediction( all_predict_P[,1], all_iris_test_target == 1)
perf <- performance( pred, "tpr", "fpr" )
xValues <- unlist(perf@x.values)
yValues <- unlist(perf@y.values)
aValues <- unlist(perf@alpha.values)
for(i in 2:length(levels(factor(iris[[5]])))){
  pred <- prediction( all_predict_P[,i], all_iris_test_target == i)
  perf <- performance( pred, "tpr", "fpr" )
  xValues <- xValues + unlist(perf@x.values)
  yValues <- yValues + unlist(perf@y.values)
  aValues <- aValues + unlist(perf@alpha.values)
}
perf@x.values <- list(xValues / length(levels(factor(iris[[5]]))))
perf@y.values <- list(yValues / length(levels(factor(iris[[5]]))))
perf@alpha.values <- list(aValues / length(levels(factor(iris[[5]]))))
```

```
plot( perf, col = "blue")
print(totalMatPoly)
CrossTable(totalMatPoly, prop.chisq = FALSE)
cat("Polynomial SVM: %", (sum(diag(totalMatPoly)) / sum(totalMatPoly) * 100))

#########################################################################
########################## leaf(polynomial svm) ##########################
#########################################################################

leaf <- read.csv("C:\\Users\\Murat\\Desktop\\ss\\ML\\HW\\HW1\\leaf.csv", header = FALSE)
#Randomly shuffle the data
leaf<-leaf[sample(nrow(leaf)),]

#set class as factor
leaf[[1]]=factor(leaf[[1]])
#Create 10 equally size folds
folds <- cut(seq(1,nrow(leaf)),breaks=10,labels=FALSE)
gp <- runif(nrow(leaf))   # random siralama
leaf <- leaf[order(gp),]
totalMatPoly <- matrix(0,36,36)
all_predict_L <- c()
all_leaf_test_target <- c()

for(i in 1:10){
        #-- train ve test olarak ayırma --#
        #Segement your data by fold using the which() function
        testIndexes <- which(folds==i,arr.ind=TRUE)
        leaf_test <- leaf[testIndexes, ]
        leaf_train <- leaf[-testIndexes, ]
        leaf_train_target <- leaf[-testIndexes, 1]   # 1 => kolon numarası
        leaf_test_target <- leaf[testIndexes, 1]
        all_leaf_test_target <- c(all_leaf_test_target, leaf_test_target)
        filter_Linear <- ksvm(V1~.,data=leaf_train, kernel="polydot", prob.model = TRUE)
        leaf_type_L <- predict(filter_Linear,leaf_test, type = "prob")
        all_predict_L <- rbind(all_predict_L, leaf_type_L)
```

```
        leaf_type_L <- predict(filter_Linear,leaf_test, type = "response")
        table(leaf_test_target, leaf_type_L)
        totalMatPoly <- as.matrix(CreateTable(leaf_test_target, leaf_type_L)) + totalMatPoly
}
pred <- prediction( all_predict_L[,1], all_leaf_test_target == 1)
perf <- performance( pred, "tpr", "fpr" )
xValues <- unlist(perf@x.values)
yValues <- unlist(perf@y.values)
aValues <- unlist(perf@alpha.values)
for(i in 2:length(levels(factor(leaf[[1]])))){
 pred <- prediction( all_predict_L[,i], all_leaf_test_target == i)
 perf <- performance( pred, "tpr", "fpr" )
 xValues <- xValues + unlist(perf@x.values)
 yValues <- yValues + unlist(perf@y.values)
 aValues <- aValues + unlist(perf@alpha.values)
}
perf@x.values <- list(xValues / length(levels(factor(leaf[[1]]))))
perf@y.values <- list(yValues / length(levels(factor(leaf[[1]]))))
perf@alpha.values <- list(aValues / length(levels(factor(leaf[[1]]))))
plot( perf, col = "red")
cat("Polynomial SVM: %", (sum(diag(totalMatPoly)) / sum(totalMatPoly) * 100))
write.table(totalMatPoly, "Polynomial_SVM_Result.txt", sep="\t")
```

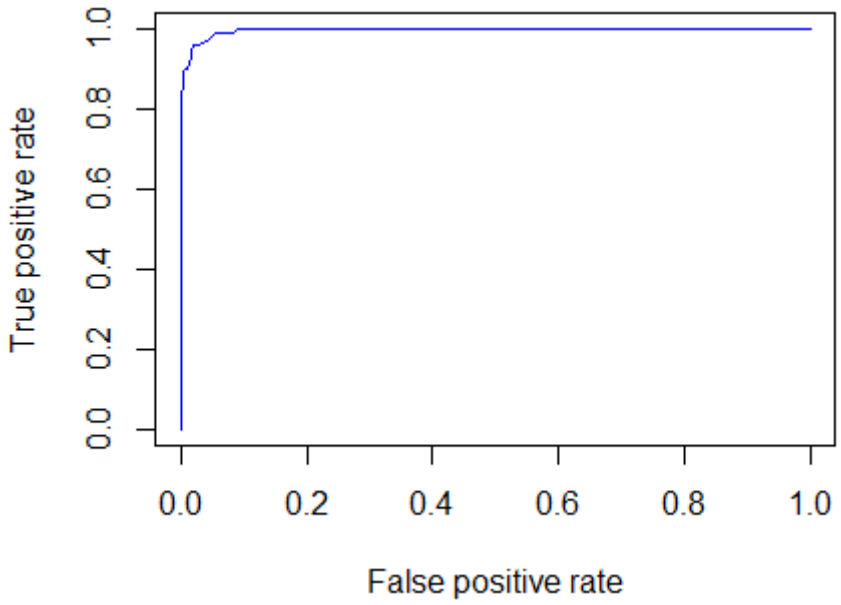
Results:

Iris Dataset

```
> print(totalMatPoly)
                 iris_type_P
iris_test_target setosa versicolor virginica
      setosa         50          0         0
      versicolor      0         45         5
      virginica       0          1        49
> CrossTable(totalMatPoly, prop.chisq = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  150
```

iris_test_target	iris_type_P setosa	versicolor	virginica	Row Total
setosa	50	0	0	50
	1.000	0.000	0.000	0.333
	1.000	0.000	0.000	
	0.333	0.000	0.000	
versicolor	0	45	5	50
	0.000	0.900	0.100	0.333
	0.000	0.978	0.093	
	0.000	0.300	0.033	
virginica	0	1	49	50
	0.000	0.020	0.980	0.333
	0.000	0.022	0.907	
	0.000	0.007	0.327	
Column Total	50	46	54	150
	0.333	0.307	0.360	

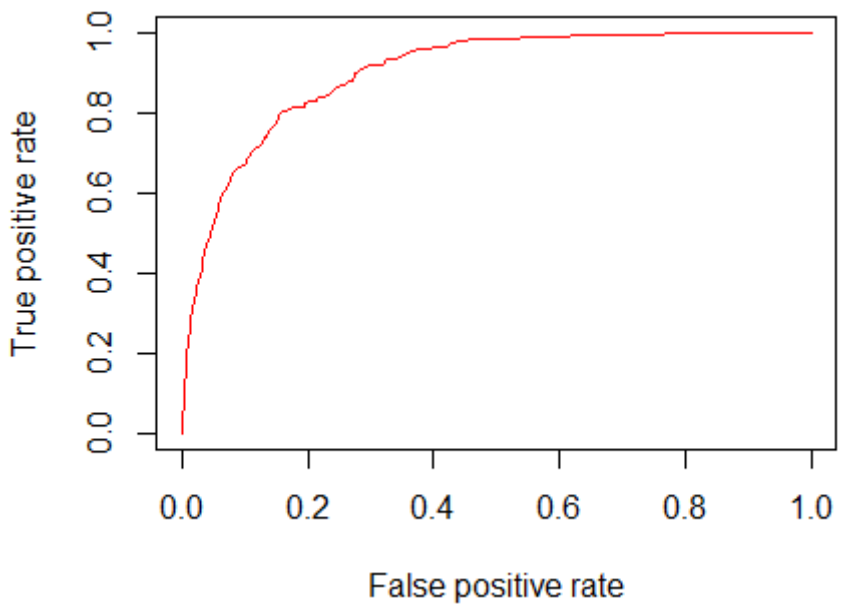
```
> cat("Polynomial SVM: %", (sum(diag(totalMatPoly)) / sum(totalMatPoly) *
100))
Polynomial SVM: % 96
```



Leaf Dataset

```
> cat("Polynomial SVM: %", (sum(diag(totalMatPoly)) / sum(totalMatPoly) *
100))
Polynomial SVM: % 74.41176
> write.table(totalMatPoly, "Polynomial_SVM_Result.txt", sep="\t")
```

(leaf datasının tablosu çok büyük olduğu için çıktıyı bir txt ye kaydedip ek te gönderdim.)



Comments:

Bu kısımda kernlab kütüphanesindeki ksvm fonksiyonunu kullandım. İris ve leaf verilerinde kernel'ı "polydot" seçerek polynomial svm'i kullandım. Cross validation yaparak veriyi 10'a böldüm ve sonuçları bir matriste topladım. Doğru sonuçları tüm sonuçlara oranladım ve % 'lik başarıyı elde ettim. Tahmin yüzdelerini ve gerçek labelleri bir listede tutarak ROC Curve çiziminde kullandım. Birden fazla sınıf olduğu için her sınıfın ROC Curve'nün ortalamasını alarak ortalama bir ROC Curve çizdirdim.