**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\_type\_P  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 | 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. |