Bioinformatics HW5

Ryan Gallagher

2023-11-17

## Use the same data set ‘Golub\_Merge’ as from HW 4.

### a) Perform gene selection by selecting the top 50 genes with the largest t-statistics comparing ALL and AML in *all 72 samples.* With the 50 selected genes, use the first 34 samples as the training set to construct a prediction model and test on the remaining 38 samples. What is the error rate (confusion matrix) for LDA, KNN, random forest, and SVM methods? Which is better?

data(Golub\_Merge)  
dat = Golub\_Merge  
  
pheno\_data = pData(Golub\_Merge)  
exprs\_data = exprs(Golub\_Merge)  
class = pheno\_data$ALL.AML  
  
t\_stats = apply(exprs\_data, 1, function(gene\_expression) {  
 all\_group = gene\_expression[class == "ALL"]  
 aml\_group = gene\_expression[class == "AML"]  
   
 mean\_diff = mean(all\_group) - mean(aml\_group)  
 pooled\_sd = sqrt(((var(all\_group) / length(all\_group)) + (var(aml\_group) / length(aml\_group))))  
 t\_stat = mean\_diff / pooled\_sd  
 return(t\_stat)  
})  
  
top\_genes\_indices = order(abs(t\_stats), decreasing = TRUE)[1:50]  
selected\_genes = rownames(exprs\_data)[top\_genes\_indices]

# Split data  
train\_data = exprs\_data[selected\_genes, 1:34]  
train\_data = t(train\_data)  
train\_labels = class[1:34]  
  
test\_data = exprs\_data[selected\_genes, 35:72]  
test\_data = t(test\_data)  
test\_labels = class[35:72]

#LDA  
lda\_model = lda(train\_data, train\_labels)

## Warning in lda.default(x, grouping, ...): variables are collinear

lda\_pred = predict(lda\_model, test\_data)$class

#KNN  
k\_values = 1:20  
cv\_errors = numeric(length(k\_values))  
  
for (i in seq\_along(k\_values)) {  
 cv\_results = knn.cv(train = train\_data, cl = train\_labels, k = k\_values[i])  
 cv\_errors[i] = sum(cv\_results != train\_labels) / length(train\_labels)  
}  
  
best\_k = k\_values[which.min(cv\_errors)]  
  
# using the best k value / using k = 3  
knn\_pred\_cv = knn\_pred = knn(train = train\_data, test = test\_data, cl = train\_labels, k = best\_k)  
knn\_pred = knn(train = train\_data, test = test\_data, cl = train\_labels, k = 3)

# Random Forest  
rf\_model = randomForest(train\_data, as.factor(train\_labels), ntree = 100)  
rf\_pred = predict(rf\_model, test\_data)

# SVM  
svm\_model = svm(train\_data, train\_labels, type = 'C-classification', kernel = 'linear')  
svm\_pred = predict(svm\_model, test\_data)

# Confusion matrices and error rates  
conf\_matrix\_lda = table(Predicted = lda\_pred, Actual = test\_labels)  
conf\_matrix\_knn = table(Predicted = knn\_pred, Actual = test\_labels)  
conf\_matrix\_knn\_cv = table(Predicted = knn\_pred\_cv, Actual = test\_labels)  
conf\_matrix\_rf = table(Predicted = rf\_pred, Actual = test\_labels)  
conf\_matrix\_svm = table(Predicted = svm\_pred, Actual = test\_labels)  
  
error\_rate\_lda = 1 - sum(diag(conf\_matrix\_lda)) / sum(conf\_matrix\_lda)  
error\_rate\_knn = 1 - sum(diag(conf\_matrix\_knn)) / sum(conf\_matrix\_knn)  
error\_rate\_knn\_cv = 1 - sum(diag(conf\_matrix\_knn\_cv)) / sum(conf\_matrix\_knn\_cv)  
error\_rate\_rf = 1 - sum(diag(conf\_matrix\_rf)) / sum(conf\_matrix\_rf)  
error\_rate\_svm = 1 - sum(diag(conf\_matrix\_svm)) / sum(conf\_matrix\_svm)  
  
# Compare error rates  
error\_rates = c(LDA = error\_rate\_lda, KNN = error\_rate\_knn, KNN\_cv = error\_rate\_knn\_cv, RF = error\_rate\_rf, SVM = error\_rate\_svm)  
error\_rates

## LDA KNN KNN\_cv RF SVM   
## 0.21052632 0.10526316 0.00000000 0.00000000 0.02631579

It appears that the method with the lowest test error rate is Random Forest / KNN with cross-validation.

### a) Perform gene selection by selecting the top 50 genes with the largest t-statistics comparing ALL and AML in *the first 34 samples.* With the 50 selected genes, use the first 34 samples as the training set to construct a prediction model and test on the remaining 38 samples. What is the error rate (confusion matrix) for LDA, KNN, random forest, and SVM methods? Which is better?

data(Golub\_Merge)  
dat = Golub\_Merge  
  
pheno\_data = pData(Golub\_Merge)[1:34, ]  
exprs\_data = exprs(Golub\_Merge)[, 1:34]  
class = pheno\_data$ALL.AML  
  
t\_stats = apply(exprs\_data, 1, function(gene\_expression) {  
 all\_group = gene\_expression[class == "ALL"]  
 aml\_group = gene\_expression[class == "AML"]  
   
 mean\_diff = mean(all\_group) - mean(aml\_group)  
 pooled\_sd = sqrt(((var(all\_group) / length(all\_group)) + (var(aml\_group) / length(aml\_group))))  
 t\_stat = mean\_diff / pooled\_sd  
 return(t\_stat)  
})  
  
  
top\_genes\_indices = order(abs(t\_stats), decreasing = TRUE)[1:50]  
selected\_genes = rownames(exprs\_data)[top\_genes\_indices]

pheno\_data = pData(Golub\_Merge)  
exprs\_data = exprs(Golub\_Merge)  
class = pheno\_data$ALL.AML  
  
# Split data  
train\_data = exprs\_data[selected\_genes, 1:34]  
train\_data = t(train\_data)  
train\_labels = class[1:34]  
  
test\_data = exprs\_data[selected\_genes, 35:72]  
test\_data = t(test\_data)  
test\_labels = class[35:72]

#LDA  
lda\_model = lda(train\_data, train\_labels)

## Warning in lda.default(x, grouping, ...): variables are collinear

lda\_pred = predict(lda\_model, test\_data)$class

#KNN  
k\_values = 1:20  
cv\_errors = numeric(length(k\_values))  
  
for (i in seq\_along(k\_values)) {  
 cv\_results = knn.cv(train = train\_data, cl = train\_labels, k = k\_values[i])  
 cv\_errors[i] = sum(cv\_results != train\_labels) / length(train\_labels)  
}  
  
best\_k = k\_values[which.min(cv\_errors)]  
  
# using the best k value / k=3  
knn\_pred\_cv = knn\_pred = knn(train = train\_data, test = test\_data, cl = train\_labels, k = best\_k)  
knn\_pred = knn(train = train\_data, test = test\_data, cl = train\_labels, k = 3)

# Random Forest  
rf\_model = randomForest(train\_data, as.factor(train\_labels), ntree = 100)  
rf\_pred = predict(rf\_model, test\_data)

# SVM  
svm\_model = svm(train\_data, train\_labels, type = 'C-classification', kernel = 'linear')  
svm\_pred = predict(svm\_model, test\_data)

# Confusion matrices and error rates  
conf\_matrix\_lda = table(Predicted = lda\_pred, Actual = test\_labels)  
conf\_matrix\_knn = table(Predicted = knn\_pred, Actual = test\_labels)  
conf\_matrix\_knn\_cv = table(Predicted = knn\_pred\_cv, Actual = test\_labels)  
conf\_matrix\_rf = table(Predicted = rf\_pred, Actual = test\_labels)  
conf\_matrix\_svm = table(Predicted = svm\_pred, Actual = test\_labels)  
  
error\_rate\_lda = 1 - sum(diag(conf\_matrix\_lda)) / sum(conf\_matrix\_lda)  
error\_rate\_knn = 1 - sum(diag(conf\_matrix\_knn)) / sum(conf\_matrix\_knn)  
error\_rate\_knn\_cv = 1 - sum(diag(conf\_matrix\_knn\_cv)) / sum(conf\_matrix\_knn\_cv)  
error\_rate\_rf = 1 - sum(diag(conf\_matrix\_rf)) / sum(conf\_matrix\_rf)  
error\_rate\_svm = 1 - sum(diag(conf\_matrix\_svm)) / sum(conf\_matrix\_svm)  
  
# Compare error rates  
error\_rates = c(LDA = error\_rate\_lda, KNN = error\_rate\_knn, KNN\_cv = error\_rate\_knn\_cv, RF = error\_rate\_rf, SVM = error\_rate\_svm)  
error\_rates

## LDA KNN KNN\_cv RF SVM   
## 0.07894737 0.07894737 0.07894737 0.00000000 0.07894737

Now, Random Forest performed the best and all test MSE are decreased from part (a).

### (c) Compare the results in (a) and (b), which is more appropriate? Why?

We find in part (a) that KNN is our best classifier, with a distinct difference from the other methods. In part (b), however, we find that Random Forest is our best method, though all methods are relatively reliable. This leads me to think that the method from part (b) is the most appropriate. Perhaps this method protects against overfitting, and so our models are more effictive in the test-cases here.

### (d) Instead of pre-specified 34 training and 38 test samples, repeat (b) but use 10-fold cross validation for all 72 samples instead.

data(Golub\_Merge)  
dat = Golub\_Merge  
  
pheno\_data = pData(Golub\_Merge)  
exprs\_data = exprs(Golub\_Merge)  
class = pheno\_data$ALL.AML  
  
t\_stats = apply(exprs\_data, 1, function(gene\_expression) {  
 all\_group = gene\_expression[class == "ALL"]  
 aml\_group = gene\_expression[class == "AML"]  
   
 mean\_diff = mean(all\_group) - mean(aml\_group)  
 pooled\_sd = sqrt(((var(all\_group) / length(all\_group)) + (var(aml\_group) / length(aml\_group))))  
 t\_stat = mean\_diff / pooled\_sd  
 return(t\_stat)  
})  
  
top\_genes\_indices = order(abs(t\_stats), decreasing = TRUE)[1:50]  
selected\_genes = rownames(exprs\_data)[top\_genes\_indices]

# Split data  
library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

dat = t(exprs\_data[selected\_genes,])  
control = trainControl(method = "cv", number = 10)  
labels = class

lda\_model\_cv <- train(x = t(exprs\_data[selected\_genes, ]),  
 y = class,  
 method = "lda",  
 trControl = control)  
  
knn\_model\_cv <- train(x = t(exprs\_data[selected\_genes, ]),  
 y = class,  
 method = "knn",  
 tuneGrid = expand.grid(k = 3),  
 trControl = control)  
  
rf\_model\_cv <- train(x = t(exprs\_data[selected\_genes, ]),  
 y = as.factor(class),  
 method = "rf",  
 trControl = control)  
  
svm\_model\_cv <- train(x = t(exprs\_data[selected\_genes, ]),  
 y = class,  
 method = "svmLinear",  
 trControl = control)

error\_rate\_lda\_cv = lda\_model\_cv$results$Accuracy  
error\_rate\_knn\_cv = knn\_model\_cv$results$Accuracy  
error\_rate\_rf\_cv = rf\_model\_cv$results$Accuracy  
error\_rate\_svm\_cv = svm\_model\_cv$results$Accuracy  
  
  
error\_rate\_lda\_cv = 1 - error\_rate\_lda\_cv  
error\_rate\_knn\_cv = 1 - error\_rate\_knn\_cv  
error\_rate\_rf\_cv = 1 - error\_rate\_rf\_cv  
error\_rate\_svm\_cv = 1 - error\_rate\_svm\_cv  
  
# Compare error rates  
error\_rates\_cv = c(LDA = error\_rate\_lda\_cv, KNN = error\_rate\_knn\_cv, RF = error\_rate\_rf\_cv, SVM = error\_rate\_svm\_cv)  
error\_rates\_cv

## LDA KNN RF1 RF2 RF3 SVM   
## 0.16785714 0.08214286 0.02916667 0.04166667 0.05416667 0.00000000

This time, we find that SVM is our best classifier.