Nicholas Orriols

Statistical Learning

Homework 3

1. (i) The subjects’ weight and width are continuous predictors while the values for spine and color are qualitative. The qualitative predictors are included in the model as binary values for each subject (effectively, 3 dummy variables for color and 2 for spine). Including spine and color as nominal qualitative predictors is independent of the choice of model: including them as continuous covariates would assume an (incorrect) ordinal relation between their different levels.

(ii) After reading in the data and setting color and spine as categorical predictors, the following code finds the best model for a given value of gamma in a radial SVM (for 5 values of gamma). It then outputs each model’s gamma, cost, training error, and testing error.

> set.seed(1.618)

> t=sort(sample(1:nrow(d), size=floor(.5\*nrow(d))))

> train=d[t,]

> valid=d[-t,]

>

> G=10^(-2:2)

> C=c(0.0001,0.001,0.01,0.1,1,2,3,5,10,25,50,100)

>

> cost=c()

> gamma=c()

> trainError=c()

> testError=c()

>

> for(i in 1:length(G)) {

+

+ tune = tune(svm, y~., data=train, kernel="radial",

+ ranges=list(cost=C, gamma=c(G[i])))

+ cost[i]=tune$best.model$cost

+ gamma[i]=tune$best.model$gamma

+ trainError[i]=tune$best.performance

+ testError[i]=mean(predict(

+ tune$best.model, newdata = valid[,-1]) != valid[,1])

+

+ }

>

> ii = data.frame(Gamma=gamma, Cost=cost,

+ Train=round(trainError,3),

+ Test=round(testError,3))

> ii

Gamma Cost Train Test

1 1e-02 25.0000 0.269 0.333

2 1e-01 2.0000 0.253 0.322

3 1e+00 0.0001 0.322 0.391

4 1e+01 0.0001 0.321 0.391

5 1e+02 0.0001 0.324 0.391

There is slight evidence of overfitting for each model in the form of testing error rates which are notably greater than the training error rates. For the smaller values of gamma (where the evidence is stronger), this is expected since the corresponding kernel function is more localized (i.e. observations which are closer together have a larger kernel function) for smaller values of gamma. In general, radial SVMs with small gamma values are more prone to over-fitting for this reason. This is also likely why the cost value for the smallest gamma model was much larger than the other cost values. Note that, since the validation data has 87 observations, the testing error rates indicate each model has roughly 30 misclassifications for the testing data.

(iii) The following code finds the best polynomial SVM of a given degree and outputs each SVM’s training and testing error.

> set.seed(1.618)

> cost=c()

> degree=c()

> trainError=c()

> testError=c()

> for(i in 1:5){

+

+ tune=tune(svm, y~., data=train, kernel="polynomial",

+ ranges=list(cost=C, degree=c(i)))

+ cost[i]=tune$best.model$cost

+ degree[i]=tune$best.model$degree

+ trainError[i]=tune$best.performance

+ testError[i]=mean(predict(

+ tune$best.model, newdata = valid[,-1]) != valid[,1])

+

+ }

>

> iii=cbind(degree, cost,

+ Train=round(trainError,3),

+ Test=round(testError,3))

> iii

degree cost Train Test

[1,] 1 3 0.279 0.322

[2,] 2 50 0.242 0.333

[3,] 3 5 0.243 0.333

[4,] 4 10 0.253 0.345

[5,] 5 10 0.275 0.345

Again, there is slight but not egregious evidence (i.e. testing error rates which are notably greater than training error rates) of overfitting. Notably, the testing error rates change very little for different degree values.

(iv) The following code over 10 attempts finds the best value of *k* using 10-fold cross-validation, and outputs how many times each best-value *k* was chosen.

> set.seed(1.618)

> indices=list()

> I = 1:nrow(d)

>

> # Get 10 random samples

> for (i in 1:10){

+

+ indices[[i]] = sort(sample(I, size=17+as.numeric(i >= 8)))

+ I = I[!is.element(I, indices[[i]])]

+

+ }

>

> metaK = c()

> for (m in 1:100) {

+ # For each k, get average error over 10 samples

+ meanErrors=c()

+ for(k in 1:17) {

+ errorsOfk=c()

+ for(i in 1:length(indices)) {

+ knn=knn(train=d[-indices[[i]], -1],

+ test=d[indices[[i]], -1],

+ cl=d[-indices[[i]], 1], k = k)

+ errorsOfk[i] = mean(knn != d[indices[[i]], 1])

+ }

+ # Average error for k over 10 samples

+ meanErrors[k] = mean(errorsOfk)

+ }

+ # Get which k had min average error over 10 samples

+ metaK[m] = which.min(meanErrors)

+ }

> # Over 100 attempts, get frequencies of which k had min average error

> table(metaK)

metaK

6 9 10 17

1 89 6 4

The output indicates that *k=9* was found to be the best value 89 out of 100 times, so it seems *k=9* would be the best value for KNN.

(v) The following code fits a logistic regression model with linear predictors, an LDA and QDA model, a KNN model with *k = 9* (result from cross-validation in (iv)), and 4 radial and polynomial SVMs for different values of gamma (denoted *Y*) and degree (denoted *d*) (the cost of each model is chosen with cross-validation). It then outputs each of the error rates and produces the following box plots.

> G=c(.01,.1,1,10)

> C=c(.01,.1,1,2,5,10,50,100)

> D=1:4

> E=matrix(NA, 100, 12)

>

> for (i in 1:100){

+

+ # Had QDA rank deficiencies with set.seed(1.618 \* i)

+ # Runtime is about 10 mins

+

+ set.seed(sqrt(3) \* i)

+ s=sort(sample(1:nrow(d), size=25))

+

+ for(j in 1:4){

+

+ rad.svm = tune(svm, y~., data=d[-s,], kernel="radial",

+ ranges=list(cost=C, gamma=G[j]))$best.model

+

+ pol.svm = tune(svm, y~., data=d[-s,], kernel="polynomial",

+ ranges=list(cost=C, degree=D[j]))$best.model

+

+ E[i,j] = mean(predict(rad.svm, newdata=d[s,-1]) != d[s,1])

+ E[i,j+4] = mean(predict(pol.svm, newdata=d[s,-1]) != d[s,1])

+

+ }

+

+ glm=glm(y~., data=d[-s,], family=binomial)

+ lda=lda(y~., data=d[-s,])

+ qda=qda(y~., data=d[-s,])

+ knn = knn(train=d[-s,-1], test=d[s,-1], cl=d[-s,1], k=9)

+

+ glm.p = as.numeric(predict(glm, newdata=d[s,-1], type="response") >= .5)

+

+ E[i,9]=mean(glm.p != d[s,1])

+ E[i,10]=mean(predict(lda, newdata=d[s,-1])$class != d[s,1])

+ E[i,11]=mean(predict(qda, newdata=d[s,-1])$class != d[s,1])

+ E[i,12]=mean(knn != d[s,1])

+

+ }

> # Radial SVMs (Y=.01,.1,1,10)

> apply(E[,1:4], 2, mean)

[1] 0.3196 0.2976 0.3056 0.3684

>

> # Polynomial SVMs (d=1,2,3,4)

> apply(E[,5:8], 2, mean)

[1] 0.3408 0.2972 0.3076 0.3180

>

> # GLM, LDA, QDA, KNN

> apply(E[,9:12], 2, mean)

[1] 0.3096 0.3060 0.3060 0.3040

> boxplot(E[,1:4], names=c("Y=.01","Y=.1","Y=1","Y=10"),

+ main="Rad SVM Mean Error Rates")

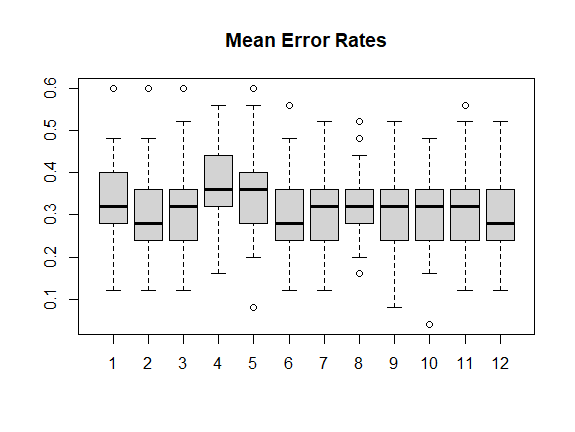
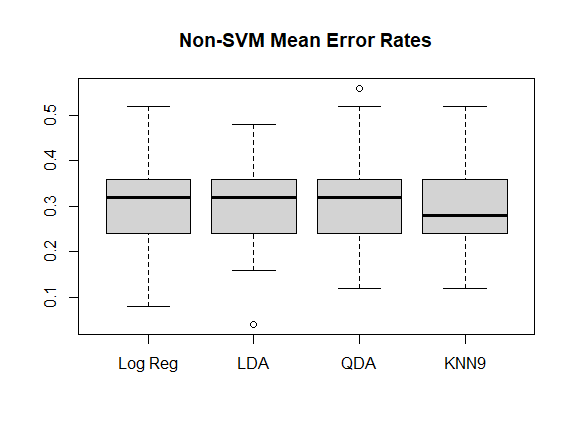
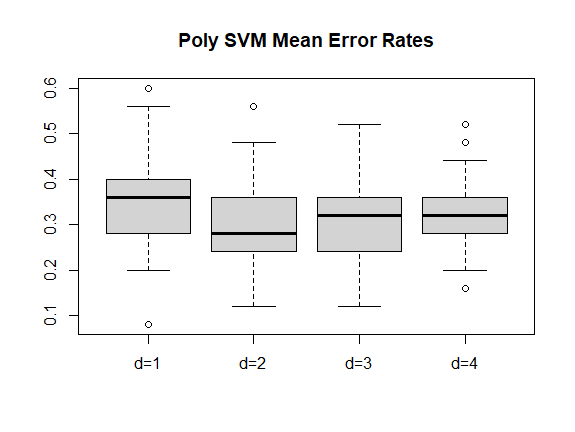
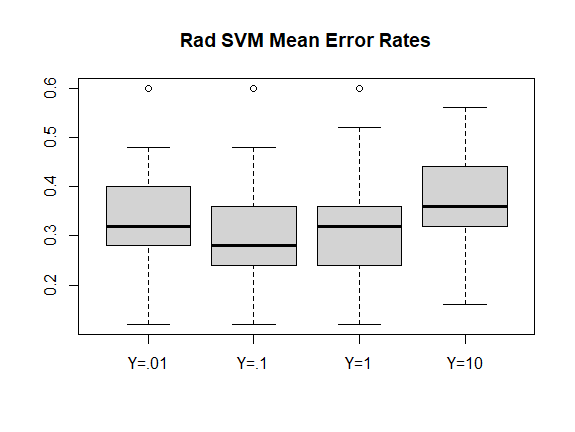
> boxplot(E[,5:8], names=c("d=1","d=2","d=3","d=4"),

+ main="Poly SVM Mean Error Rates")

> boxplot(E[,9:12], names=c("Log Reg", "LDA", "QDA", "KNN9"),

+ main="Non-SVM Mean Error Rates")

> boxplot(E, main="Mean Error Rates")



1. On average, the radial SVM with *Y = .1* and the polynomial SVM with *d = 2* had the best performances.
2. The polynomial SVM with *d = 4* had the least variation among error rates, whereas the GLM had the most variation. Most of the remaining models had similar variations among error rates. Of the 11 outliers, 2 were exceptionally low error rates (LDA had an error rate of .04 and the polynomial SVM with *d = 1* had an error rate of .08); examining over 1200 models, there is a good chance of having a few very low error rates.
3. (i) In this case, the 80th quantile is *X = 8* since integrating the density function for these *Xi* from 0 to 8 gives a value of .8.

(ii) A good estimator would be *X80*, that is the sample estimate of the 80th quantile.

(iii) The following code draws a random sample of 100 points from the uniform distribution and constructs a 95% confidence interval with the percentile method for *q.8* based on *B = 1000* estimates. It then outputs the confidence interval’s limits (which contain the estimated parameter).

> set.seed(1.618)

> X = sort(runif(100, min=0, max=10))

> theta1=c()

> for (i in 1:1000) {

+

+ s = sort(sample(X, size=100, replace = T))

+ theta1[i] = s[80]

+

+ }

>

> theta1=sort(theta1)

> CI = c(theta1[25], theta1[975]); CI

[1] 7.176185 8.394404

(iv) The following code creates 1000 random samples from the uniform distribution and then constructs each type of confidence interval for each of the 1000 samples. The confidence intervals are based on *B = 200* estimates of *q.8*. The code outputs the percentages of each type of interval that contained the true parameter (8).

> S=c(0,0)

> # Test 1000 intervals of each method

> # Runtime about 10 seconds

> for (i in 1:1000) {

+

+ set.seed(1.618\*i)

+ X = sort(runif(100, min=0, max=10))

+ theta2=c()

+

+ # Each interval based on B=200 estimates of q\_.8

+ for (j in 1:200) {

+

+ s = sort(sample(X, size=100, replace = T))

+ theta2[j] = s[80]

+

+ }

+

+ # Vector of q\_.8 estimates

+ theta2 = sort(theta2)

+

+ S[1] = S[1] + as.numeric(theta2[5]<8 & 8<theta2[195])

+

+ m = mean(theta2)

+ SE = sqrt(1/199 \* sum((theta2 - m)^2))

+ S[2] = S[2] + as.numeric(X[80]-1.96\*SE<8 & 8<X[80]+1.96\*SE)

+

+ }

> # Percentile method

> S[1]/1000

[1] 0.933

> # Standard error method

> S[2]/1000

[1] 0.929

(v) The following code is analogous to the code above except the confidence intervals are for *q.99* (the true value based on the distribution is 9.9).

> S=c(0,0)

> # Test 1000 intervals of each method

> # Runtime about 12 seconds

> for (i in 1:1000) {

+

+ set.seed(3.141\*i)

+ X = sort(runif(100, min=0, max=10))

+ theta3=c()

+

+

+ # Each interval based on B=200 estimates of q\_.99

+ for (j in 1:200) {

+

+ s = sort(sample(X, size=100, replace = T))

+ theta3[j] = s[99]

+

+ }

+

+ # Vector of q\_.99 estimates

+ theta3 = sort(theta3)

+

+ S[1] = S[1] + as.numeric(theta3[5]<9.9 & 9.9<theta3[195])

+

+ m = mean(theta3)

+ SE = sqrt(1/199 \* sum((theta3 - m)^2))

+ S[2] = S[2] + as.numeric(X[99]-1.96\*SE<9.9 & 9.9<X[99]+1.96\*SE)

+

+ }

> # Percentile method

> S[1]/1000

[1] 0.615

> # Standard error method

> S[2]/1000

[1] 0.913

The percentile method has a significantly lower than success rate than the percentile method in (iv). This is due to *q.99* being near the maximum of the distribution. When sampling a single point from the uniform distribution, there is a 1% chance that point is greater than *q.99 = 9.9*, so if we sample 100 points, on average only 1 would be greater than 9.9; in (iv), we would average 20 points greater than 8. It follows the likelihood our vector of *q.99* estimates would contain 5 values greater than 9.9 (a necessary condition for the percentile method with 95% confidence to be successful for *B = 200*) is much lower than the likelihood the vector of estimates in (iv) would contain 5 values greater than 8.

On the other hand, the standard error method can produce confidence intervals with values greater than the maximum of the distribution, so it is not restricted by its estimated parameter being near the maximum of the distribution. The produced confidence interval only depends on the sample estimate of the parameter and the amount of variation between the random sample estimates, that is not the values of the random sample estimates. Therefore, the success rate of the standard error intervals is relatively unaffected by the parameter they estimate.

1. 