Assignment_1_6659_tianni2

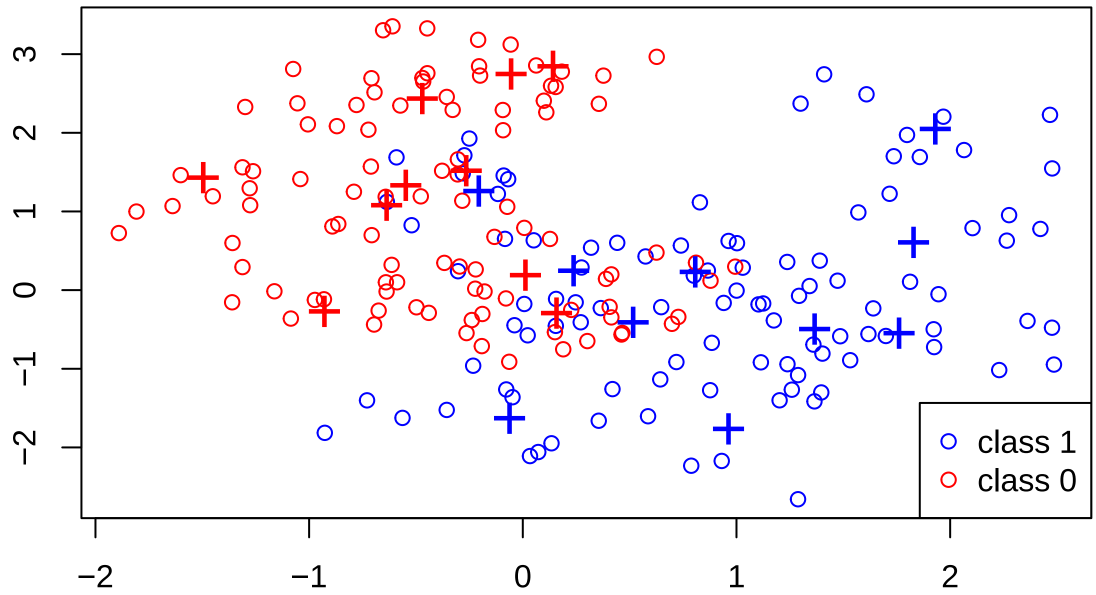
Tian Ni

9/9/2021

Construct the Data Generating Function

```
set.seed(6659)
p=2
csize=10
sigma=1
m1=matrix(rnorm(csize*p),csize,p)*sigma+cbind(rep(1,csize),rep(0,csize))
m0=matrix(rnorm(csize*p),csize,p)*sigma+cbind(rep(0,csize),rep(1,csize))

sim_params = list(
 csize = 10,      # number of centers
 p = 2,           # dimension
 s = sqrt(1/5),   # standard deviation for generating data
 n = 100,         # training size per class
 N = 5000,        # test size per class
 m0 = m0,         # 10 centers for class 0
 m1 = m1         # 10 centers for class 1
)
generate_sim_data = function(sim_params){
  p = sim_params$p
  s = sim_params$s
  n = sim_params$n
  N = sim_params$N
  m1 = sim_params$m1
  m0 = sim_params$m0
  csize = sim_params$csize

  id1 = sample(1:csize, n, replace = TRUE);
  id0 = sample(1:csize, n, replace = TRUE);
  traindata = matrix(rnorm(2*n*p), 2*n, p)*s + rbind(m1[id1,], m0[id0,])
  Ytrain = factor(c(rep(1,n), rep(0,n)))
  shuffle_row_id = sample(1:n)
  id1 = sample(1:csize, N, replace=TRUE);
  id0 = sample(1:csize, N, replace=TRUE);
  testdata = matrix(rnorm(2*N*p), 2*N, p)*s + rbind(m1[id1,], m0[id0,])
  Ytest = factor(c(rep(1,N), rep(0,N)))

  # Return the training/test data along with labels
  list(
  traindata = traindata,
  Ytrain = Ytrain,
  testdata = testdata,
  Ytest = Ytest
```

```
  )
}

mydata = generate_sim_data(sim_params)
traindata = mydata$train
Ytrain = mydata$Ytrain
testdata = mydata$testdata
Ytest = mydata$Ytest
n = nrow(traindata)

mycol = rep("blue", n)
mycol[Ytrain==0] = "red"
plot(traindata[, 1], traindata[, 2], type = "n", xlab = "", ylab = "")
points(traindata[, 1], traindata[, 2], col = mycol);
points(m1[, 1], m1[, 2], pch = "+", cex = 2, col = "blue");
points(m0[, 1], m0[, 2], pch = "+", cex = 2, col = "red");
legend("bottomright", pch = c(1,1), col = c("blue", "red"),
       legend = c("class 1", "class 0"))
```



1.

Constructing my knn function

```
set.seed(6659)
myknn=function(xtrain,xtest,ytrain,k){
  m=nrow(xtrain)
```

```
  n=nrow(xtest)
  ytrain=as.numeric(ytrain)-rep(1,length(ytrain))
  ytest=c()
  for(i in 1:n){
    distance=c()
    for(j in 1:m){
    distance[j]=sqrt((xtest[i,1]-xtrain[j,1])^2+(xtest[i,2]-xtrain[j,2])^2)
    }
    dis_y=cbind(distance,ytrain)
    o=order(distance,ytrain)
    dis_y_o=dis_y[o,]
    for(l in 1:k){
      if(l>1){
        if(dis_y_o[l,1]!=dis_y_o[l-1,1]){
          x=1
          sum_y=dis_y_o[l,2]
          for(p in 1:x){
            if(dis_y_o[l,1]==dis_y_o[l+x,1]){
            sum_y=sum_y+dis_y_o[l+x,2]
            x=x+1
        }
        dis_y_o[l:(l+x-1),2]=mean(sum_y)

      }
      }
      }
    }
    mean_y=mean(dis_y_o[1:k,2])
    if(mean_y > 0.5)
      ytest[i]=1
    if(mean_y < 0.5)
      ytest[i]=0
    if(mean_y ==0.5)
      ytest[i]=dis_y_o[1,2]
  }
  return(ytest)
}
```

In my knn function, when I have distance ties, I would take the mean probability of those realizations which have same distance; when I have voting ties(not exactly same because the k is odd in our case, what I mean is the mean probability = 0.5), I would directly let y be the same value as the closest one's y.

Test the knn function

When k is 1

```
set.seed(6659)
library(class)
test.pred=myknn(traindata,testdata,Ytrain,1)
table(Ytest,test.pred)
```

```
##       test.pred
## Ytest    0    1
##     0 3966 1034
##     1 1009 3991
```

```r
test.pred.r=knn(traindata,testdata,Ytrain,1,prob = TRUE)
table(Ytest,test.pred.r)
```

```
##      test.pred.r
## Ytest    0    1
##     0 3966 1034
##     1 1009 3991
```

Everything looks right when k=1.

When k is 3

```r
test.pred3=myknn(traindata,testdata,Ytrain,3)
table(Ytest,test.pred3)
```

```
##      test.pred3
## Ytest    0    1
##     0 4139  861
##     1 1072 3928
```

```r
test.pred.r=knn(traindata,testdata,Ytrain,3,prob = TRUE)
table(Ytest,test.pred.r)
```

```
##      test.pred.r
## Ytest    0    1
##     0 4138  862
##     1 1071 3929
```

```r
which(test.pred3 != test.pred.r)
```

```
## [1]  178 9816
```

```r
attr(test.pred.r,"prob")[178]
```

```
## [1] 0.5
```

```r
attr(test.pred.r,"prob")[9816]
```

```
## [1] 0.5
```

When k is 3, elements [178] and [9816] are mismatched, and we can see that their predict probabilities are both 0.5.

When k is 5

```r
test.pred5=myknn(traindata,testdata,Ytrain,5)
table(Ytest,test.pred5)
```

```
##      test.pred5
## Ytest    0    1
##     0 4242  758
##     1 1022 3978
```

```r
test.pred.r=knn(traindata,testdata,Ytrain,5,prob = TRUE)
table(Ytest,test.pred.r)
```

```
##      test.pred.r
## Ytest    0    1
##     0 4242  758
##     1 1023 3977
```

```
which(test.pred5 != test.pred.r)
```

```
## [1] 4802
```

```
attr(test.pred.r,"prob")[4802]
```

```
## [1] 0.5
```

When k is 5, element [4802] is mismatched, and we can see that its predict probability is 0.5.

2.

Linear Regression

```
set.seed(6659)
fit_lr_model_matrix = function(sim_data) {

  # change Y from factor to numeric
  sim_data$Ytrain = as.numeric(sim_data$Ytrain) - 1
  sim_data$Ytest = as.numeric(sim_data$Ytest) - 1

  train_matrix = sim_data$traindata
  test_matrix =  sim_data$testdata

  # obtain quadratic regression coefs
  coefs = lm(sim_data$Ytrain ~ train_matrix)$coef
  train_yhat = coefs[1] + train_matrix %*% coefs[-1]
  test_yhat = coefs[1] + test_matrix %*% coefs[-1]

  decision_thresh = 0.5

  train_pred = as.numeric(train_yhat > decision_thresh)
  test_pred = as.numeric(test_yhat > decision_thresh)

  # return the mean classification errors on training/test sets
  list(
    train_error = sum(sim_data$Ytrain  != train_pred) / length(sim_data$Ytrain),
    test_error = sum(sim_data$Ytest  != test_pred) /
      length(sim_data$Ytest)
  )
}
lr=matrix(rep(0.000000,100),50,2)
for(i in 1:50){
  mydata = generate_sim_data(sim_params)
  lr[i,1]=fit_lr_model_matrix(mydata)$train_error
  lr[i,2]=fit_lr_model_matrix(mydata)$test_error
}
```

Quadratic Regression

```
set.seed(6659)
fit_qr_model_matrix = function(sim_data) {

  # change Y from factor to numeric
```

```
  sim_data$Ytrain = as.numeric(sim_data$Ytrain) - 1
  sim_data$Ytest = as.numeric(sim_data$Ytest) - 1

  train_matrix = cbind(sim_data$traindata, sim_data$traindata^2, sim_data$traindata[,1] * sim_data$train
  test_matrix = cbind(sim_data$testdata, sim_data$testdata^2, sim_data$testdata[,1] * sim_data$testdata

  # obtain quadratic regression coefs
  coefs = lm(sim_data$Ytrain ~ train_matrix)$coef
  train_yhat = coefs[1] + train_matrix %*% coefs[-1]
  test_yhat = coefs[1] + test_matrix %*% coefs[-1]

  decision_thresh = 0.5

  train_pred = as.numeric(train_yhat > decision_thresh)
  test_pred = as.numeric(test_yhat > decision_thresh)

  # return the mean classification errors on training/test sets
  list(
    train_error = sum(sim_data$Ytrain  != train_pred) / length(sim_data$Ytrain),
    test_error = sum(sim_data$Ytest  != test_pred) /
      length(sim_data$Ytest)
  )
}

qr=matrix(rep(0.000000,100),50,2)
for(i in 1:50){
  mydata = generate_sim_data(sim_params)
  qr[i,1]=fit_qr_model_matrix(mydata)$train_error
  qr[i,2]=fit_qr_model_matrix(mydata)$test_error
}
```

CV-KNN

```
cvKNNAveErrorRate=function(K,traindata,Ytrain,foldNum){
  n = nrow(traindata)
  foldSize = floor(n/foldNum)
  error = 0
  myIndex = sample(1 : n)
  for(runId in 1:foldNum){
  testSetIndex = ((runId-1)*foldSize + 1):(ifelse(runId == foldNum, n, runId*foldSize))
  testSetIndex = myIndex[testSetIndex]
  trainX = traindata[-testSetIndex, ]
  trainY = Ytrain[-testSetIndex]
  testX = traindata[testSetIndex, ]
  testY = Ytrain[testSetIndex]
  predictY = knn(trainX, testX, trainY, K)
  error = error + sum(predictY != testY)
}
error = error / n
error
}

cvKNN = function(traindata, Ytrain, foldNum) {
```

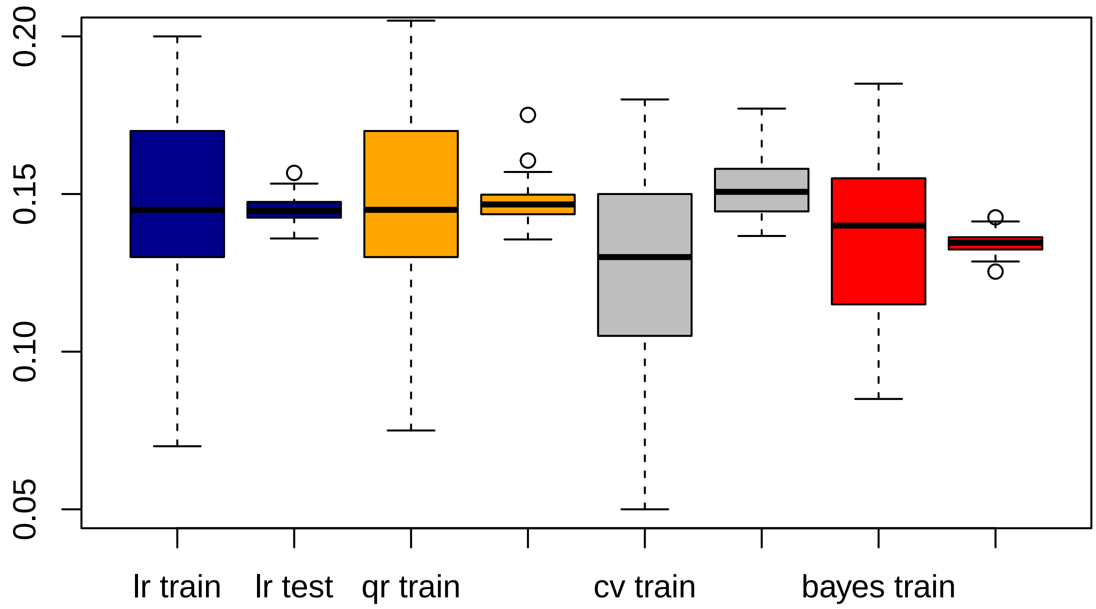
```
  n = nrow(traindata)
  foldSize = floor(n/foldNum)
  KVector = seq(1, (nrow(traindata) - foldSize), 1)
  cvErrorRates = sapply(KVector, cvKNNAveErrorRate, traindata, Ytrain,foldNum)
  result = list()
  result$bestK = max(KVector[cvErrorRates == min(cvErrorRates)])
  result$cvError = cvErrorRates[KVector == result$bestK]
  result
}

cv=matrix(rep(0.000000,200),50,2)
k_value=c()
for(i in 1:50){
  mydata = generate_sim_data(sim_params)
  k_value[i]=cvKNN(mydata$traindata,mydata$Ytrain,10)$bestK
  x1=mean(mydata$Ytrain != knn(mydata$traindata,mydata$traindata,mydata$Ytrain,k_value[i],prob = TRUE))
  x2=mean(mydata$Ytest != knn(mydata$traindata,mydata$testdata,mydata$Ytrain,k_value[i],prob = TRUE))
  cv[i,1]=x1
  cv[i,2]=x2
}
```

Bayes

```
set.seed(6659)
mixnorm = function(x, centers0, centers1, s){
  ## return the density ratio for a point x, where each
  ## density is a mixture of normal with multiple components
  d1 = sum(exp(-apply((t(centers1) - x)^2, 2, sum) / (2 * s^2)))
  d0 = sum(exp(-apply((t(centers0) - x)^2, 2, sum) / (2 * s^2)))

  return (d1 / d0)
}

# Construct the function to calculate the error rate
b_error=function(data,y,m0,m1){
  b_y_pred=c(rep(0,nrow(data)))
  for(i in 1:nrow(data)){
    p=mixnorm(data[i,],m0,m1,sqrt(1/5))
    b_y_pred[i]= ifelse(p>=1,1,0)
}

b_err=sum(b_y_pred != y)/nrow(data)
return(b_err)
}


bayes=matrix(rep(0.000000,100),50,2)
for(i in 1:50){
  sim=sim_params
  mydata = generate_sim_data(sim)
  bayes[i,1]=b_error(mydata$traindata,mydata$Ytrain,sim$m0,sim$m1)
  bayes[i,2]=b_error(mydata$testdata,mydata$Ytest,sim$m0,sim$m1)
}
```

Plot

```
boxplot(cbind(lr,qr,cv[,1:2],bayes),names=c("lr train","lr test","qr train","qr test","cv train","cv tes
```



Mean and sd of k-value

```
mean(k_value)
```

```
## [1] 27.82
```

```
sd(k_value)
```

```
## [1] 24.60437
```