kNN vs Linear Regression

Example I

Data Generation

Set the model parameters like dimension and standard error, and store the two centers (each is a two-dim vector) in m1 and m2.

```
p = 2;
s = 1;      # sd for generating x
m1 = c (1,0);
m0 = c(0,1);
```

Generate n samples from each normal component. First, we generate a (2n)-by-p matrix with entries being iid samples form a normal dist with mean 0 and variance s-square. Then we form a (2n)-by-p matrix with the first n rows being the two-dimensional vector m1 (the center of the first normal component) and the next n rows being m2. We use command `rep` to generate repeated rows and use command `rbind` to stack two matrices vertically together.

```
n = 100;  # generate the train data; n for each class
traindata = matrix(rnorm(2*n*p), 2*n, p)*s +
  rbind(matrix(rep(m1, n), nrow = n, byrow=TRUE),
        matrix(rep(m0, n), nrow = n, byrow=TRUE))

dim(traindata)
Ytrain = factor(c(rep(1,n), rep(0,n)))
```

Generate 2N test samples similarly.

```
N = 5000;
testdata = matrix(rnorm(2*N*p), 2*N, p)*s +
  rbind(matrix(rep(m1, N), nrow = N, byrow=TRUE),
        matrix(rep(m0, N), nrow = N, byrow=TRUE))
Ytest = factor(c(rep(1,N), rep(0,N)))
```

Visulization

Let’s take a look of the data. In the plot generated by the code below, points from two groups are colored in red and blue, respectively; the two centers are plotted as +, and a legend is added to explain the association of each color.

The option `type=n` tells R not to print anything, just to set the plotting area, then we add points group by group.

```
plot(traindata[,1], traindata[,2], type="n", xlab="", ylab="")

# add points from class 1
points(traindata[1:n,1], traindata[1:n,2], col="blue");
# add points from class 0
points(traindata[(n+1):(2*n),1], traindata[(n+1):(2*n),2],
       col="red");

# add the 10 centers for class 1
points(m1[1], m1[2], pch = "+", cex = 2, col = "blue");
# add the 10 centers for class 0
points(m0[1], m0[2], pch = "+", cex = 2, col = "red");

legend("bottomright", pch = c(1,1), col = c("red", "blue"),
       legend = c("class 1", "class 0"))
```

Visulization with ggplot2

(Feel free to skip this part if you've known how to use ggplot2.)

Previously I plot the data in the old "Artist Palette" way in the sense that I start with a convas (plotting region), and then add items one by one, first points, then color, then legend. Here is the drawback of this approach: if I want to add a point outside the current plotting region, then I have to tear off the convas and restart everything by starting a larger plotting region (to reserve the space for points I'll add later.)

A better graphical package for R is `ggplot2` ; gg stands for grammar for graphics http://ggplot2.org/ (http://ggplot2.org/)

Two major commands: `qplot` and `ggplot` . You can find many examples and tutorialsonline.

Run the following code (each line gives your a slightly different plot) to know how to set various arguments in `qplot` .

```r
library(ggplot2)

## ggplot only accepts dataframe
mytraindata = data.frame(X1 = traindata[,1],
                         X2 = traindata[,2],
                         Y = Ytrain)

## plot the data and color by the Y label
qplot(X1, X2, data = mytraindata, colour = Y)

## change the shape and size of the points
qplot(X1, X2, data = mytraindata, colour = Y,
      shape = I(1), size = I(3))

## change x,y labels and add titles
qplot(X1, X2, data = mytraindata, colour = Y,
      shape = I(1), size = I(3),
      main = "Scatter Plot (Training Data)",
      xlab="", ylab="")

## want to remove the gray background? Change theme
qplot(X1, X2, data = mytraindata, colour = Y,
      shape = I(1), size = I(3)) + theme_bw()

## more theme options
## A single command would be a little lengthy. We can save
## the output of qplot into an object "myplot"

myplot = qplot(X1, X2, data = mytraindata, colour = Y,
               shape = I(1), size = I(3))
myplot = myplot + theme_bw();
myplot + theme(legend.position="left")
?theme

## How to adding the two centers on
## existing plot, we need to use the command "ggplot"
?ggplot
myplot = ggplot(mytraindata, aes(X1, X2))

## use user-specified color
myplot + geom_point(aes(color = Y), shape = 1, size = 3) + scale_color_manual(values = c("red",
"blue"))

## add the two centers;
## change shape and size;
## use user-sepecified color
myplot = ggplot(mytraindata,aes(X1, X2)) +
  geom_point(aes(colour = Y), shape = 3, size = 2) +
  scale_color_manual(values = c("red", "blue"))

myplot +
  geom_point(data = data.frame(X1 = m1[1], X2 = m1[2]),
             aes(X1, X2), colour = "blue", shape = 2, size = 5) +
```

```
  geom_point(data = data.frame(X1 = m0[1], X2 = m0[2]),
             aes(X1, X2), colour = "red", shape = 2, size = 5)
```

K-NN method

```
library(class)

## Choice of the neighhood size.
## Here I just use the values from the textbook
myk = c(151, 101, 69, 45, 31, 21, 11, 7, 5, 3, 1)
m = length(myk);

train.err.knn = rep(0,m);
test.err.knn = rep(0, m);

for( j in 1:m){
  Ytrain.pred = knn(traindata, traindata, Ytrain, k = myk[j])
  train.err.knn[j] = sum(Ytrain != Ytrain.pred)/(2*n)
  Ytest.pred = knn(traindata, testdata, Ytrain,k = myk[j])
  test.err.knn[j] = sum(Ytest != Ytest.pred)/(2*N)
}
```

Least Sqaure Method

```
RegModel = lm(as.numeric(Ytrain) - 1 ~ traindata)
Ytrain_pred_LS = as.numeric(RegModel$fitted > 0.5)
Ytest_pred_LS = RegModel$coef[1] + RegModel$coef[2] * testdata[,1] +
  RegModel$coef[3] * testdata[,2]
Ytest_pred_LS = as.numeric(Ytest_pred_LS > 0.5)

## cross tab for training data and training error
table(Ytrain, Ytrain_pred_LS);
train.err.LS = sum(Ytrain !=  Ytrain_pred_LS) / (2*n);

## cross tab for test data and test error
table(Ytest, Ytest_pred_LS);
test.err.LS = sum(Ytest !=  Ytest_pred_LS) / (2*N);
```

Bayes Error

```
Ytest_pred_Bayes = as.numeric(2*testdata %*% matrix(m1-m0, nrow=2) > (sum(m1^2)-sum(m0^2)));

test.err.Bayes = sum(Ytest !=  Ytest_pred_Bayes) / (2*N)
```

Plot the Performance

Test errors are in red and training errors are in blue. The upper x-coordinate indicates the K values, and the lower x-coordinate indicates the degree-of-freedom of the KNN procedures so the labels are reciprocally related to K.

The training and test errors for linear regression are plotted at df=3, since the linear model has 3 parameters, i.e., 3 dfs.

```
plot(c(0.5,m), range(test.err.LS, train.err.LS, test.err.knn, train.err.knn),
     type="n", xlab="Degree of Freedom", ylab="Error", xaxt="n")

df = round((2*n)/myk)
axis(1, at = 1:m, labels = df)
axis(3, at = 1:m, labels = myk)

points(1:m, test.err.knn, col="red", pch=1);
lines(1:m, test.err.knn, col="red", lty=1);
points(1:m, train.err.knn, col="blue", pch=1);
lines(1:m, train.err.knn, col="blue", lty=2);

points(3, train.err.LS, pch=2, cex=2, col="blue")
points(3, test.err.LS, pch=2, cex=2, col="red")

abline(test.err.Bayes, 0, col="purple")
```

Example II

Data Generation

Generate the 20 centers, 10 for each group.

```
csize = 10;       # number of centers
p = 2;
sigma = 1;      # sd for generating the centers
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));
```

Generate training data.

```
n=100;
# Randomly allocate the n samples for class 1  to the 10 clusters
id1 = sample(1:csize, n, replace = TRUE);
# Randomly allocate the n samples for class 1 to the 10 clusters
id0 = sample(1:csize, n, replace = TRUE);

s= sqrt(1/5);                              # sd for generating x.

traindata = matrix(rnorm(2*n*p), 2*n, p)*s + rbind(m1[id1,], m0[id0,])
Ytrain = factor(c(rep(1,n), rep(0,n)))
```

Generate test data.

```
N = 5000;
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)))
```

Visulization

```
plot(traindata[, 1], traindata[, 2], type = "n", xlab = "", ylab = "")

points(traindata[1:n, 1], traindata[1:n, 2], col = "blue");
points(traindata[(n+1):(2*n), 1], traindata[(n+1):(2*n), 2], col="red");

points(m1[1:csize, 1], m1[1:csize, 2], pch="+", cex=1.5, col="blue");
points(m0[1:csize, 1], m0[1:csize, 2], pch="+", cex=1.5, col="red");

legend("bottomleft", pch = c(1,1), col = c("red", "blue"),
       legend = c("class 1", "class 0"))
```

Bayes Error

The calculation for the Bayes error for Example II is different from the one for Example I since the underlying data generating processes are different.

Since we need to repeatly evaluate a mixture of normal with 10 components on each test point, I wrote a function first `mixnorm` , and then use the command `apply` to apply the same function on each row of the test data.

```
mixnorm=function(x){
  ## return the density ratio for a point x, where each
  ## density is a mixture of normal with 10 components
  sum(exp(-apply((t(m1)-x)^2, 2, sum)*5/2))/sum(exp(-apply((t(m0)-x)^2, 2, sum)*5/2))
}

Ytest_pred_Bayes = apply(testdata, 1, mixnorm)
Ytest_pred_Bayes = as.numeric(Ytest_pred_Bayes > 1);
table(Ytest, Ytest_pred_Bayes);
test.err.Bayes = sum(Ytest !=  Ytest_pred_Bayes) / (2*N)
```