Lab 1: Bayes Classifier in 1D

```
library(tidyverse)
```

```
## -- Attaching packages -------- tidyverse 1.2.1 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.3.0

## -- Conflicts ----------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(class)
dir <- "/Users/richeym/Dropbox/COURSES/ADM/ADM_S20/CODE/Chap02"
file <- "Lab1D_Data.csv"
```

Comment

A very complete solution to this lab.

Introduction

Let's followup on the discussion of the Bayes Classifier. In this case, we'll work with data with only one predictor ($p = 1$). The goal is to show that you understand the basic ideas of the Bayes Classifier and can apply a KNN model to data.

Fill out this RMarkdown document with your work.

The Data

The data is located in file called "Lab1DData.csv". There are N=10000 observations and two fields.

- x coordinate
- probabilitity of class=="0"

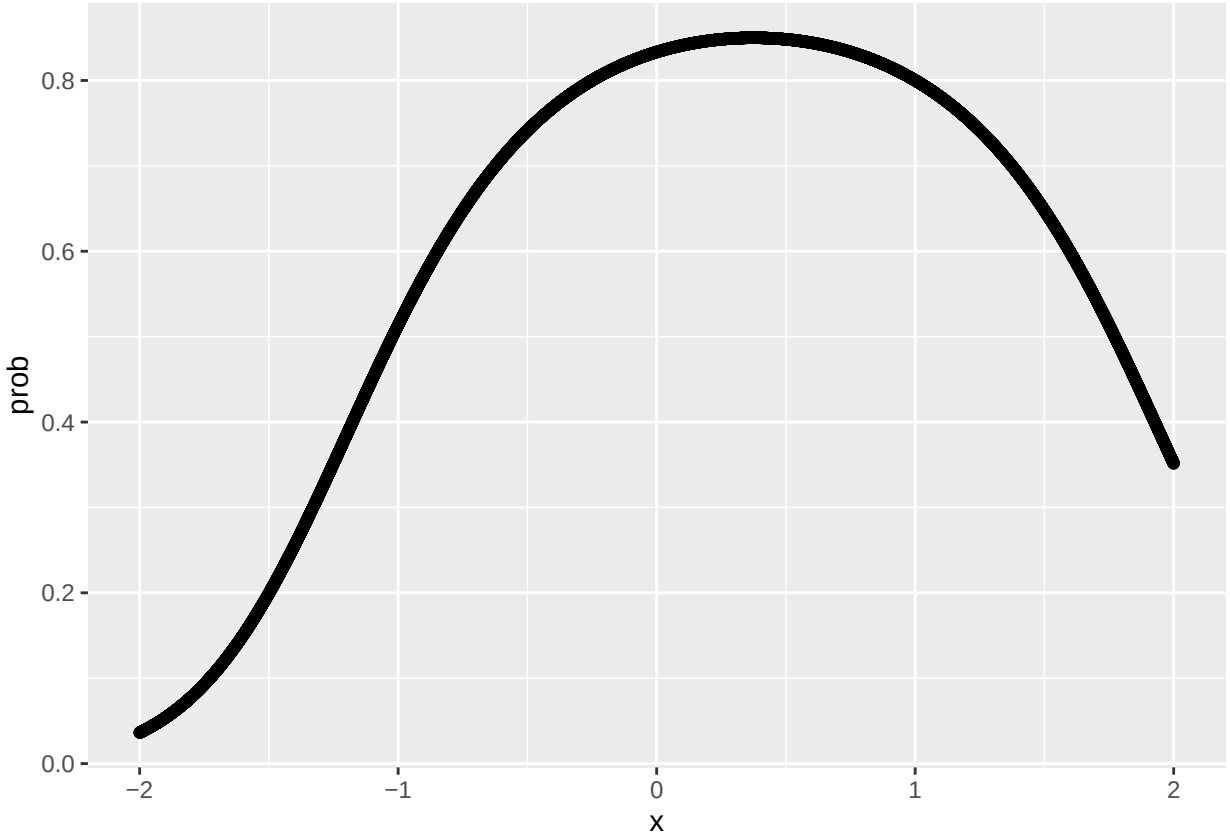
Treat this data as your "population" data. From this data set you will draw samples to create test/train data sets. As well, you also know the underlying probability model and classfication of each point based on that probability model. In practice you would not know the probability model!

Read the data into a data frame called "data.df."

```
data.df <- read.csv(file.path(dir,file))
```

It never hurts to look at the data.

```
data.df %>%
  ggplot()+
  geom_point(aes(x,prob))+
  labs("Probabilites")
```



We can see that the probability is above 0.5 near the center of the interval. This is the region in which we are most like to have the class="0"

Bayes Classifier

Use data.df to estimate the Bayes Decision Rule (Boundary) This is a rule for determining into which class you should place an observed value x. Include some sort of graphic with illustrates this rule.

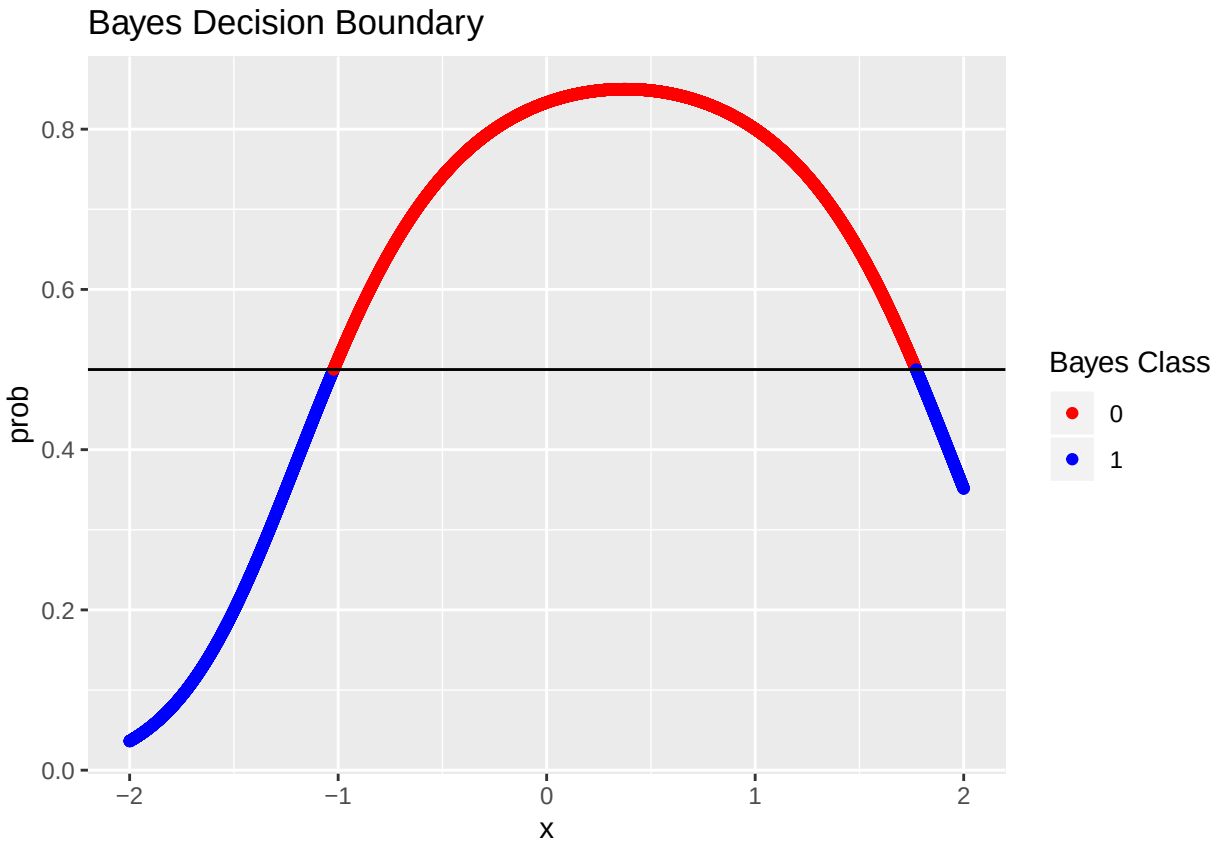
The Bayes Decison boundary occurs at the points where the probability changes from below 0.5 to above 0.5.

First, create the Bayes classification.

```
data.df <- data.df %>%
  rowwise() %>%
  mutate(bayesClass=ifelse(prob>0.5,"0","1"))
```

Now, look at what we have.

```
data.df %>%
  ggplot()+
  geom_point(aes(x,prob,color=bayesClass))+
  scale_color_manual(values=c("red","blue"))+
  geom_hline(aes(yintercept=0.5))+
  labs(title="Bayes Decision Boundary",
       color="Bayes Class")
```



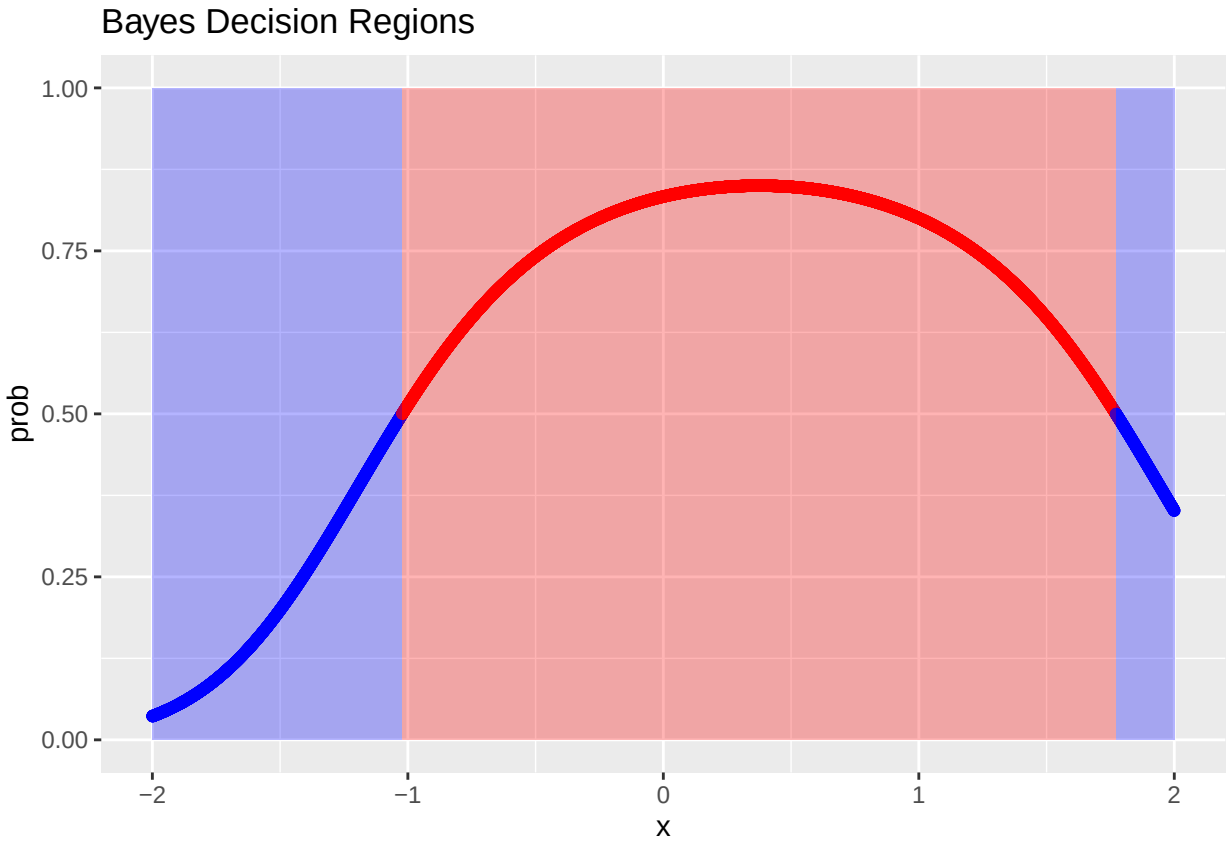
The precise location of these is easy to find.

```
x1 <- data.df %>%
  filter(x<0,prob<0.5) %>%
  with(max(x))


x2 <- data.df %>%
  filter(x >0,prob<0.5) %>%
  with(min(x))

c(x1,x2)
```

```
## [1] -1.021902  1.771977
```

Thus a Bayes decision rule for this data is: classify as “0” if and only if xx1.

```
data.df %>%
  ggplot()+
  geom_point(aes(x,prob,color=bayesClass))+
  geom_rect(data=data.frame(xLeft=c(-2,x1,x2),xRight=c(x1,x2,2),class=c("0","1","0")),
            aes(xmin=xLeft,xmax=xRight,ymin=0,ymax=1,fill=class),alpha=.3)+
  scale_color_manual(values=c("red","blue"))+
  scale_fill_manual(values=c("blue","red"))+
  labs(title="Bayes Decision Regions") +
  theme(legend.position = "none")
```



Class Assignments

Using the probability values, assign a class to each of the observations in data.df. In other works, if at x=1.2 (say) the corresponding probability is 0.72, then x=1.2 will be in class="0" with prob=0.72 and in class="1" with probability 0.28.

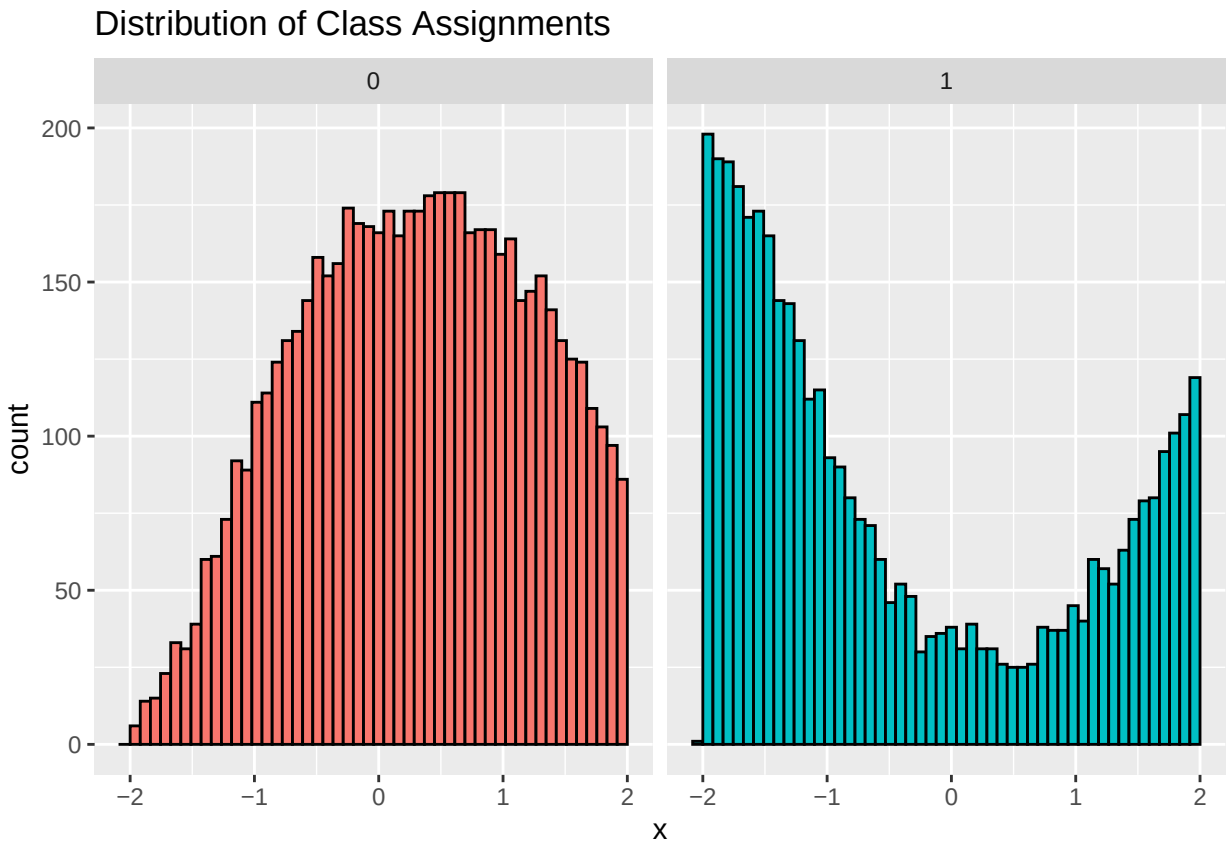
```
data.df <- data.df %>%
  rowwise() %>%
  mutate(class=ifelse(runif(1,0,1) < prob,"0","1"))
```

How well did this work? Start by looking at the counts in each class.

```
with(data.df,table(class))
```

```
## class
##    0    1
## 6018 3982
```

```
data.df %>%
  ggplot()+
  geom_histogram(aes(x,fill=class),color="black",bins=50)+
  facet_wrap(~class)+
  labs(title="Distribution of Class Assignments") +
  theme(legend.position = "none")
```



As expected, most of the class==“0” observations are near the middle, the x==“1” observations are grouped at the edges.

Bayes Error Rate

What is the Bayes error rate? Use your classfications of the observations in data.df and your Bayes Decision Rule (above) to estimate the Bayes error rate.

```
(err.bayes <- with(data.df,mean(class != bayesClass)))
```

```
## [1] 0.2563
```

Based on this initial class assignment, this is a 0.25 good estimate of the Bayes Error rate. One should probably reassign the classes several times to make sure this value is stable.

KNN modeling

Let $N = 100$. Create training and testing data sets of size N to estimate the KNN error rate as a function of kVal (the number nearest neighbors). Present your work as a graph which shows the estimated error rate as a function of kVal. Identify your best estimate of the optimal choice of kVal.

```
N <- 100
dataSize <- nrow(data.df)
train.df <- data.df[sample(1:dataSize,N,rep=F),]
test.df <- data.df[sample(1:dataSize,N,rep=F),]
train.X <- as.matrix(train.df[c("x")])
train.Y <- as.matrix(train.df["class"])
test.X <- as.matrix(test.df[c("x")])
```

```
kVal <- 20
mod.knn <- knn(train.X,test.X,train.Y,k=kVal,prob=TRUE)
with(test.df,mean(class != mod.knn))
```

```
## [1] 0.27
```

```
maxK <- N/2
reps <- 100
errs.knn <- matrix(nrow=maxK,ncol=reps)

for(kVal in 1:maxK){
  print(sprintf("K=%s",kVal))
  for(rep in 1:reps){
    ##New train and test data
    train.df <- data.df[sample(1:dataSize,N,rep=F),]
    test.df <- data.df[sample(1:dataSize,N,rep=F),]
    train.X <- as.matrix(train.df[c("x")])
    train.Y <- as.matrix(train.df["class"])
    test.X <- as.matrix(test.df[c("x")])
    mod.knn <- knn(train.X,test.X,train.Y,k=kVal,prob=TRUE)
    errs.knn[kVal,rep] <- with(test.df,mean(class!=mod.knn))
  }
}
```

```
## [1] "K=1"
## [1] "K=2"
## [1] "K=3"
## [1] "K=4"
## [1] "K=5"
## [1] "K=6"
## [1] "K=7"
## [1] "K=8"
## [1] "K=9"
## [1] "K=10"
```

```
## [1] "K=11"
## [1] "K=12"
## [1] "K=13"
## [1] "K=14"
## [1] "K=15"
## [1] "K=16"
## [1] "K=17"
## [1] "K=18"
## [1] "K=19"
## [1] "K=20"
## [1] "K=21"
## [1] "K=22"
## [1] "K=23"
## [1] "K=24"
## [1] "K=25"
## [1] "K=26"
## [1] "K=27"
## [1] "K=28"
## [1] "K=29"
## [1] "K=30"
## [1] "K=31"
## [1] "K=32"
## [1] "K=33"
## [1] "K=34"
## [1] "K=35"
## [1] "K=36"
## [1] "K=37"
## [1] "K=38"
## [1] "K=39"
## [1] "K=40"
## [1] "K=41"
## [1] "K=42"
## [1] "K=43"
## [1] "K=44"
## [1] "K=45"
## [1] "K=46"
## [1] "K=47"
## [1] "K=48"
## [1] "K=49"
## [1] "K=50"
```

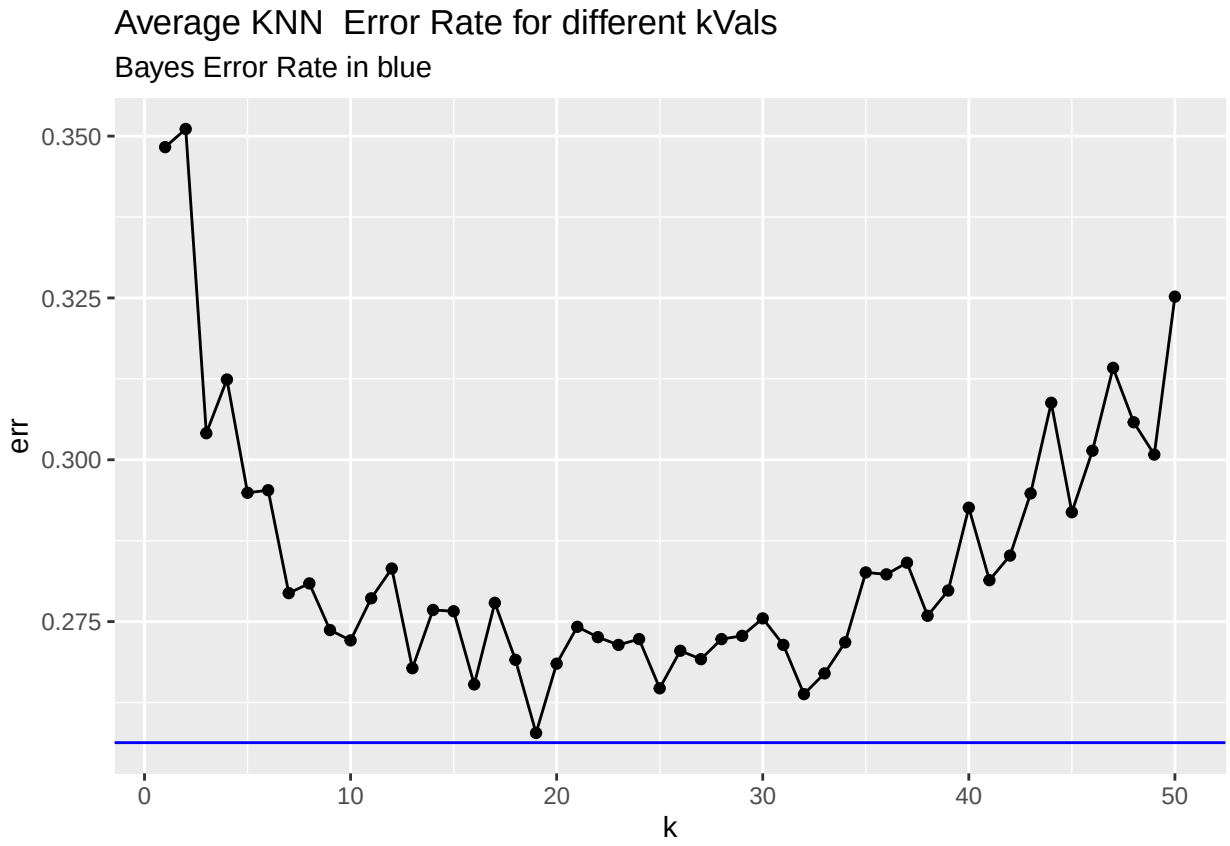
Take the average of each kVal errors

```
aveErrs <- apply(errs.knn,1,mean)
```

Now plot the results

```
data.frame(k=1:maxK,err=aveErrs) %>%
  ggplot()+
  geom_point(aes(k,err))+
  geom_line(aes(k,err))+
  ##geom_smooth(aes(k,err),method="loess",se=FALSE)+
  geom_hline(aes(yintercept=err.bayes),color="blue")+
```

```
  labs(title="Average KNN  Error Rate for different kVals",
       subtitle="Bayes Error Rate in blue")
```



It looks as if the optimal choice of k is around 20-25 with an error rate of about 0.265 or so.

Extra: Logistic Regression

How well does logistic regression work? Is the error rate comparable to the best error rate from KNN.

```
mod.log <- glm(factor(class)~x,data=train.df,family="binomial")
```

Convert the model to probabilities and then to actual predictions. (Note: there is no reason anyone should know how to do this.

```
## Probabilites...this predicts the probability of being in class == 0
prob.log <- predict(mod.log,newdata=test.df,type="response")
## Class Predictions..convert from probability to class
class.log <- ifelse(prob.log < 0.5,"0","1")
```

How did logistic regression do?

```
with(test.df,table(class, class.log))
```

```
##      class.log
```

```
## class  0  1
##     0 42 16
##     1 28 14
```

```
(err.log <- with(test.df,mean(class != class.log)))
```

```
## [1] 0.44
```

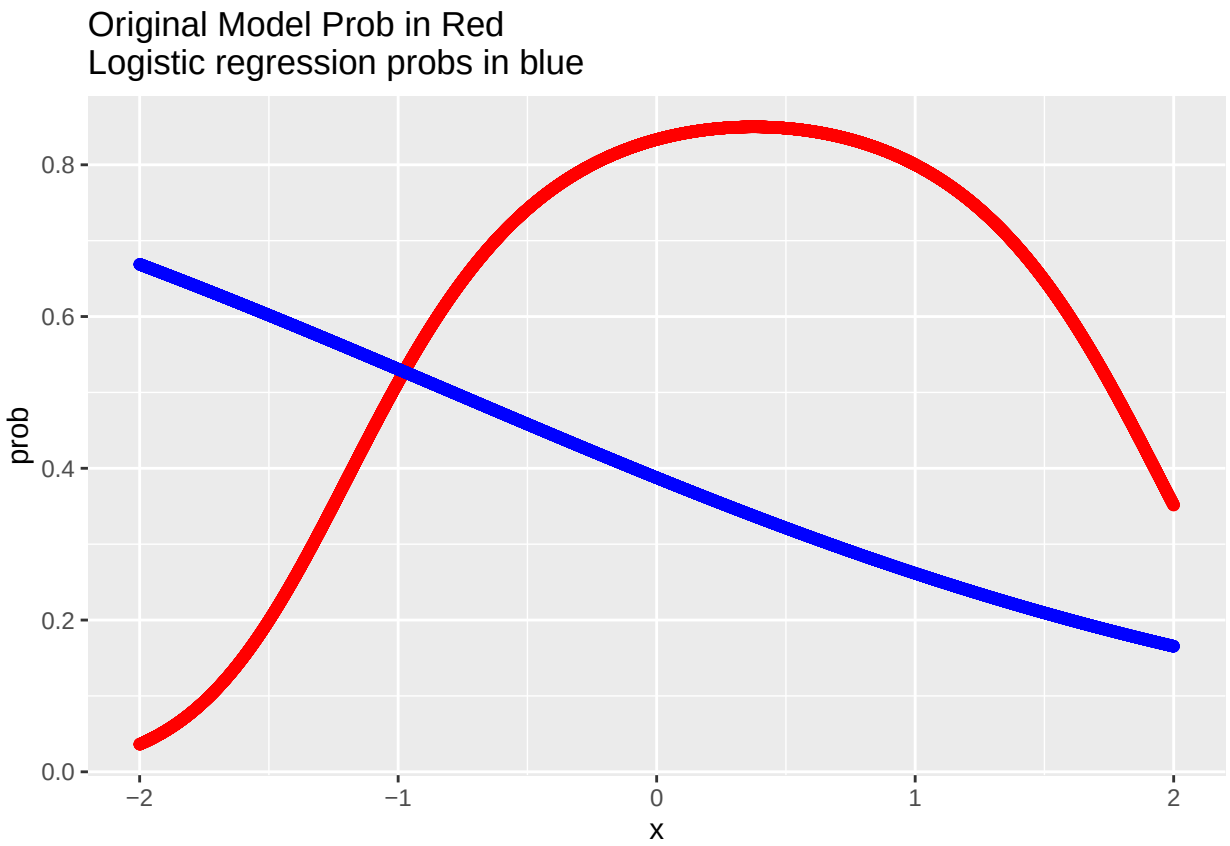
In this case logistic regression is not competitive with KNN.

Why....use logistic regression to predict on the original data set.

```
mod.log2 <- glm(factor(class)~x,data=data.df,family="binomial")
prob.log2 <- predict(mod.log2,newdata=data.df,type="response")
data.df$prob.log <- prob.log2
```

Now look at the predicted probabilities versus the original probabilities.

```
data.df %>%
  ggplot()+
  geom_point(aes(x,prob),color="red")+
  geom_point(aes(x,prob.log),color="blue")+
  labs(title="Original Model Prob in Red\nLogistic regression probs in blue")
```



Not a very good match. The reason is that logistic regression assumes a certain shape (logistic) of the x vs probability curve! In this case, that assumption is clearly not reasonable.