Homework 7

Megan Robertson

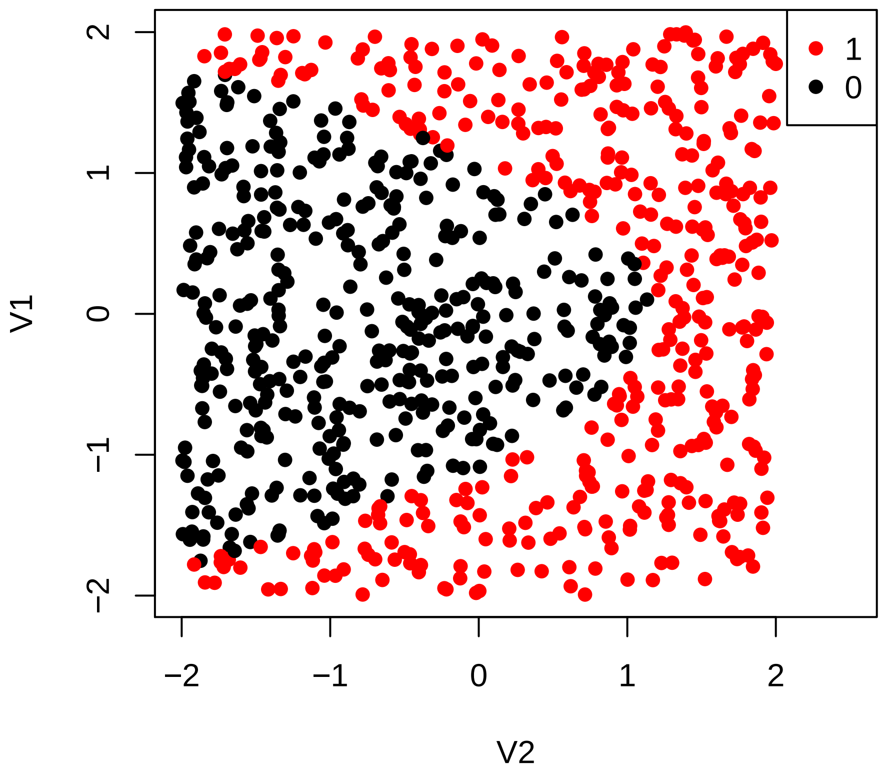
Friday, November 20, 2015

1. Here you will learn how to draw more flexible boundaries than simple linear ones, using logistic regression. Download the data from Sakai and load it into your R session. There are two scenarios for the class labels, given by y_1 and y_2.

```
#setting working directory and loading data
setwd("C:/Users/Megan Robertson/Documents/STA521/Homework7")
data <- load("hw6prob1.Rdata")
data = x
```

a. Plot the data in x with the class labels given by y_1. (Use the option col or pch or both to distinguish between the classes). Run logistic regression, using the glm function with family="binomial", to build a prediction rule. What is the training misclassification rate of this rule? (Hint: use the predict function to read the relevant documentation, type ?predict.glm)

```
#creating data with the y1 lables as a column
data.y1 = cbind(x, y1)

#creating training set and test set
set.seed(10)
train.index = sample(nrow(data.y1), size = .7*nrow(data.y1), replace = FALSE)
train = data.y1[train.index,]
test = data.y1[-train.index,]

#plotting the data in data.y1 with labels
plot(V1~V2, col = as.factor(y1), data = data.y1, pch=16, xlim = c(-2, 2.5))
legend("topright", c("1", "0"), pch=c(16, 16), col=c("red", "black"))
```

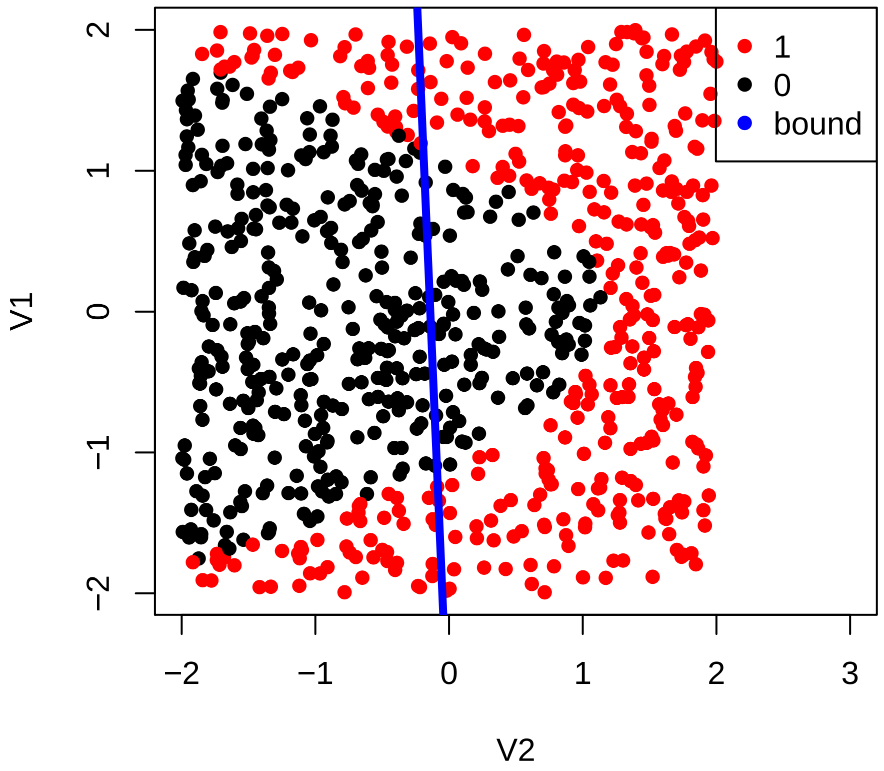


```r
#running logistic regression to create prediction rule
train = data.frame(train)
y1.logit = glm(y1 ~ V1 + V2, data = train, family = binomial)

#finding the training misclassification rate
#predicting for the test set
test = data.frame(test)
test$predict = predict(y1.logit, newdata = test, type="response")

#classifying predictions as 0 or 1
test$predicted = 0
test[which(test$predict >0.5),]$predicted=1

test$diff = test$predicted - test$y1
Incorrect = length(which(test$diff != 0))
MisclassRate = Incorrect/nrow(test)
```

The training missclassification rate of this rule is approximately 25.83%.

b. Draw the decision boundary in R^2 of the logistic regression from (a), on top of your plot from (a). What shape is it? Does this boundary look like it adequately separates the classes?

```
data.y1.df = data.frame(data.y1)
colnames(data.y1.df) = c("v1", "v2", "y1")

#defining the boundary (v1 in terms of v2)
v2.values = seq(from = -2, to = 2, length.out = 1000)
v1.line = (coef(y1.logit)[1] + coef(y1.logit)[3]*v2.values)/(-coef(y1.logit)[2])


plot(V1~V2, col = as.factor(y1), data = data.y1, pch=16, xlim = c(-2, 3))
lines(v1.line~v2.values, col = "blue", lwd=4)
legend("topright", c("1", "0", "bound"), pch=c(16, 16, 16),
  col=c("red", "black", "blue"))
```



The decision boundary is a straight line with a large negative slope. It does not look like this boundary adequately separates the classes.There are a large number of points of both colors on both sides of the boundary. The data appears to have a boundary that should be curved, possibly quadratic, and as a result the straight line does not adequately separate the classes.

c. Run logistic regression on the predictors in x, as well as the predictor $x[,1]^2$. This is analogous to adding a quadratic term to a linear regression. To do this, define a new predictor matrix x.quad = cbind(x, $x[,1]^2$), and run a logistic regression of y_1 on x.quad. What is the training misclassification rate of this rule? Why is this better than the rule from (a)?

```
#creating matrix with the quadratic predictor
x.quad = cbind(x, y1, x[,1]^2)

#creating training and testing data sets
train.quad = x.quad[train.index,]; train.quad = data.frame(train.quad)
colnames(train.quad) = c("V1", "V2", "y1", "V1.sq")
test.quad = x.quad[-train.index,]; test.quad = data.frame(test.quad)
colnames(test.quad) = c("V1", "V2", "y1", "V1.sq")

#running logistic regression on training data
y1.logit.quad = glm(y1 ~ V1 + V2 + V1.sq, data = train.quad,
  family = binomial, control = list(maxit = 50))
```

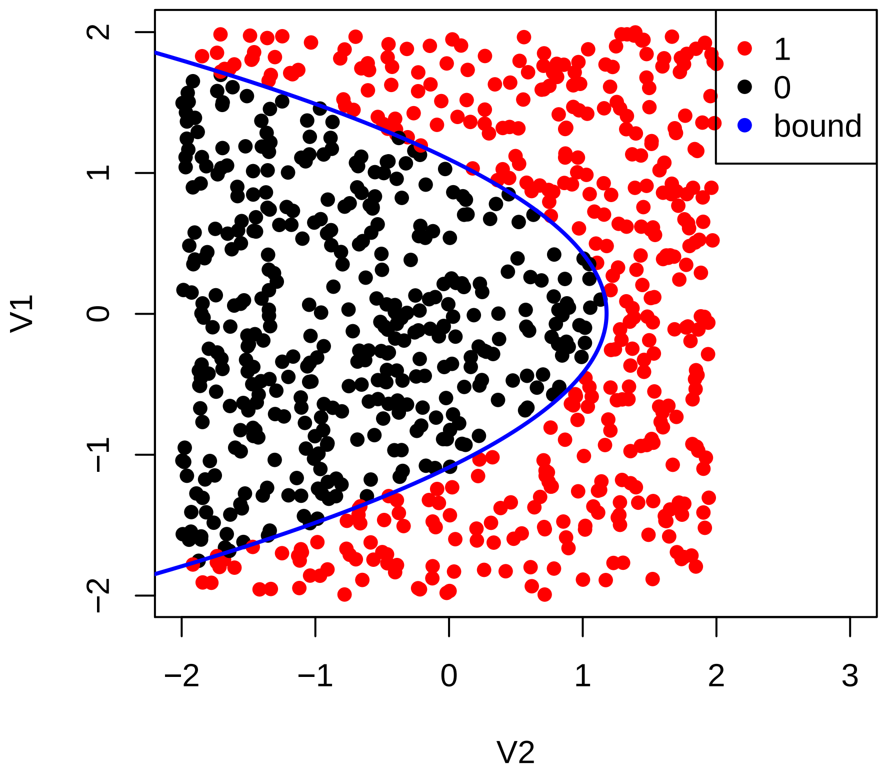
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
#finding the training misclassification rate
#predicting for the test set
test.quad$predict = predict(y1.logit.quad, newdata = test.quad, type = "response")

#classifying predictions as 0 or 1
test.quad$predicted = 0
for (i in (1:nrow(test.quad))){
  if (test.quad[i,]$predict > 0.5){test.quad[i,]$predicted = 1}
}
test.quad$diff = test.quad$predicted - test.quad$y1
Incorrect = length(which(test.quad$diff != 0))
MisclassRate = Incorrect/nrow(test.quad)
```

The training missclassification rate of this rule is 0. This is better than the rule from a because it includes a quadratic term and the data appears to have a quadratic relationship.

d. In R^2, i.e. in the space x[,1] versus x[,2], what is the shape of the decison boundary of the logistic regression rule from (c)? Draw this decision boundary on top of a plot of the (appropriately color-coded or pch-coded) data x. What shape is it?

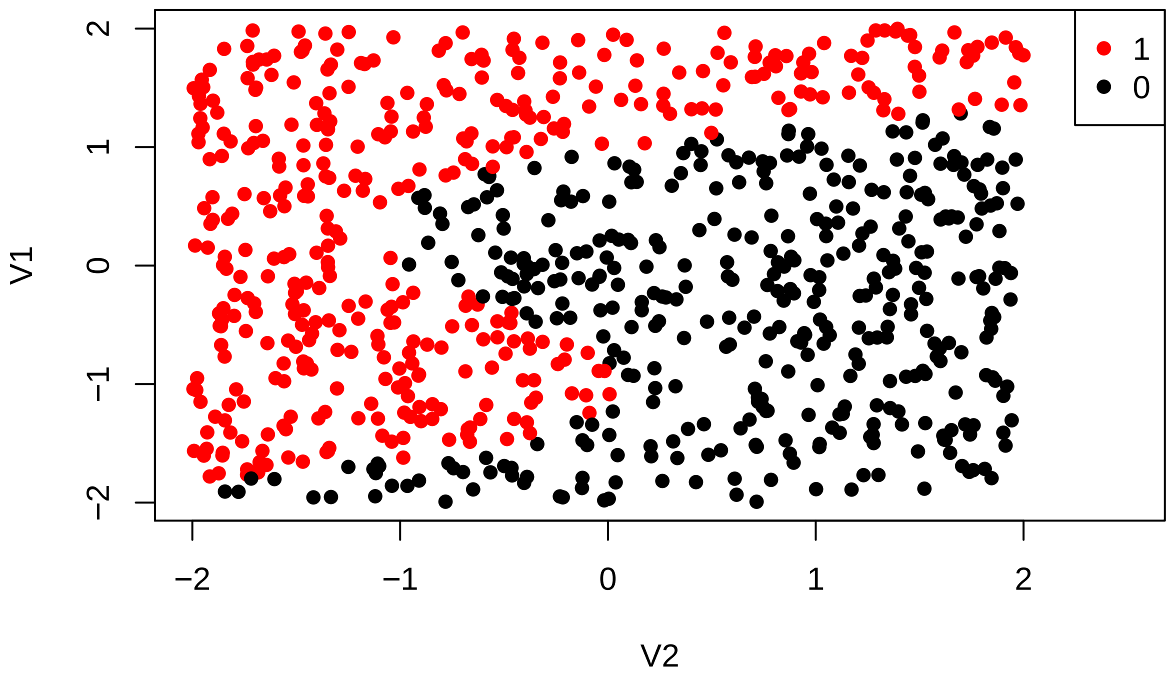
```
#getting line for decision boundary
v1.values = seq(from = -2, to = 2, length.out = 1000)
v2.line = (coef(y1.logit.quad)[1]
  + coef(y1.logit.quad)[2]*v1.values
  + coef(y1.logit.quad)[4]*(v1.values)^2)/(-coef(y1.logit.quad)[3])

#plotting
plot(V1~V2, col = as.factor(y1), data = data.y1, pch=16, xlim = c(-2, 3))
lines(v1.values ~ v2.line, col = "blue", lwd=2)
legend("topright", c("1", "0", "bound"), pch=c(16, 16, 16), col=c("red", "black", "blue"))
```



The decision boundary in this situation is a parabola. The decision boundary does an excellent job of separating the classes.

e. Plot the data in x with the labels given by y_2. Try a running logistic regression of y_2 on x, and also on x.quad = cbind(x, $x[,1]^2$). What are the training misclassification rates of these rules? Draw the decision boundaries of each rule on top of a plot of the data.

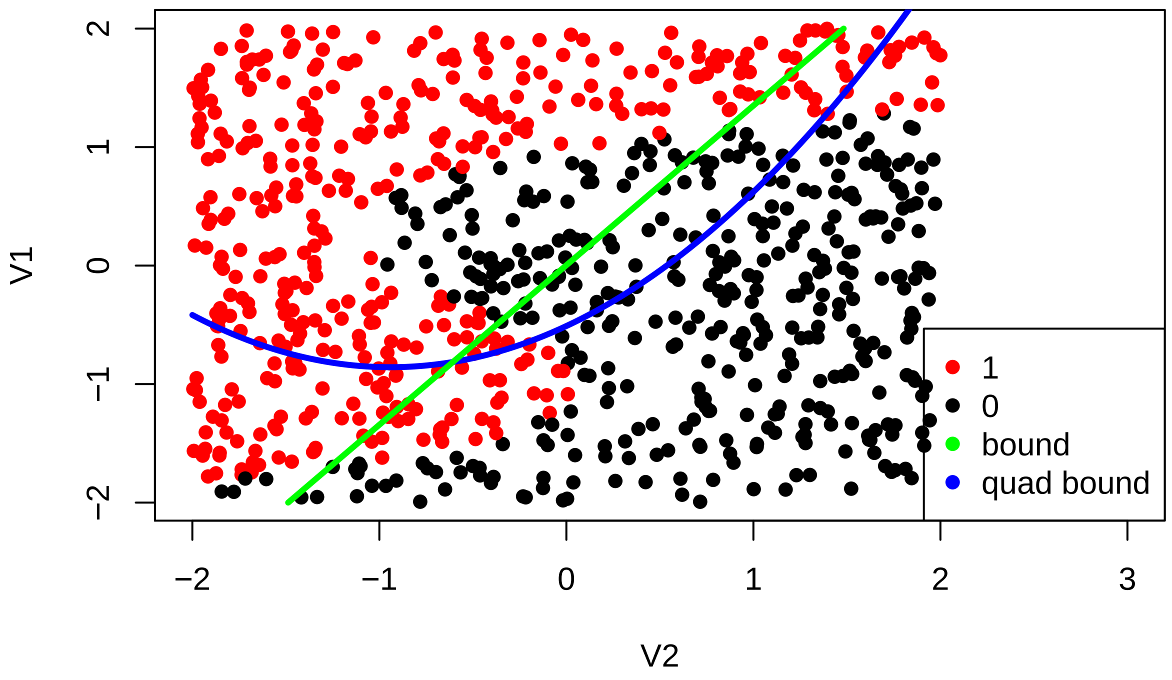
```
#creating data with the y1 lables as a column
data.y2 = cbind(x, y2)

#creating training set and test set
train.2 = data.y2[train.index,]
test.2 = data.y2[-train.index,]

#plotting the data in data.y2 with labels
plot(V1~V2, col = as.factor(y2), data = data.y2, pch=16, xlim = c(-2, 2.5))
legend("topright", c("1", "0"), pch=c(16, 16), col=c("red", "black"))
```



```
#running logistic regression to create prediction rule
train.2 = data.frame(train.2)
y2.logit = glm(y2 ~ V1 + V2, data = train.2, family = binomial)

#getting line to plot
v1.line.2 = (coef(y2.logit)[1] + coef(y2.logit)[2]*v2.values)/(-coef(y2.logit)[3])

#finding the training misclassification rate
#predicting for the test set
test.2 = data.frame(test.2)
test.2$predict = predict(y2.logit, newdata = test.2)

#classifying predictions as 0 or 1
test.2$predicted = 0
for (i in (1:nrow(test.2))){
  if (test.2[i,]$predict > 0.5){test.2[i,]$predicted = 1}
}
test.2$diff = test.2$predicted - test.2$y2
Incorrect = length(which(test.2$diff != 0))
MisclassRate = Incorrect/nrow(test.2)
```

```
###quadratic term included
#creating matrix with the quadratic predictor
x.quad.2 = cbind(x, y2, x[,1]^2)

#creating training and testing data sets
```

```r
train.quad.2 = x.quad.2[train.index,]; train.quad.2 = data.frame(train.quad.2)
colnames(train.quad.2) = c("V1", "V2", "y2", "V1.sq")
test.quad.2 = x.quad.2[-train.index,]; test.quad.2 = data.frame(test.quad.2)
colnames(test.quad.2) = c("V1", "V2", "y2", "V1.sq")

#running logistic regression on training data
y2.logit.quad = glm(y2 ~ V1 + V2 + V1.sq, data = train.quad.2, family = binomial,
  control = list(maxit = 50))

#finding the training misclassification rate
#predicting for the test set
test.quad.2$predict = predict(y2.logit.quad, newdata = test.quad.2, type="response")

#classifying predictions as 0 or 1
test.quad.2$predicted = 0
for (i in (1:nrow(test.quad.2))){
  if (test.quad.2[i,]$predict > 0.5){test.quad.2[i,]$predicted = 1}
}
test.quad.2$diff = test.quad.2$predicted - test.quad.2$y2
Incorrect = length(which(test.quad.2$diff != 0))
MisclassRate = Incorrect/nrow(test.quad.2)

#getting line to plot
v2.line.2 = (coef(y2.logit.quad)[1] + coef(y2.logit.quad)[2]*v1.values +
  coef(y2.logit.quad)[4]*(v1.values)^2)/(-coef(y2.logit.quad)[3])

#plotting
plot(V1~V2, col = as.factor(y2), data = data.y2, pch=16, xlim = c(-2, 3))
lines(v2.line.2~v1.values, lwd = 3, col = "blue")
lines(v2.values~v1.line.2, lwd = 3, col = "green")
legend("bottomright", c("1", "0", "bound", "quad bound"),
  pch=c(16, 16, 16, 16), col=c("red", "black", "green", "blue"))
```



The training missclassification rate of the rule without the quadratic term is about 20.83% and the training missclassification rate of the rule with the quadratic term is 13.75%.

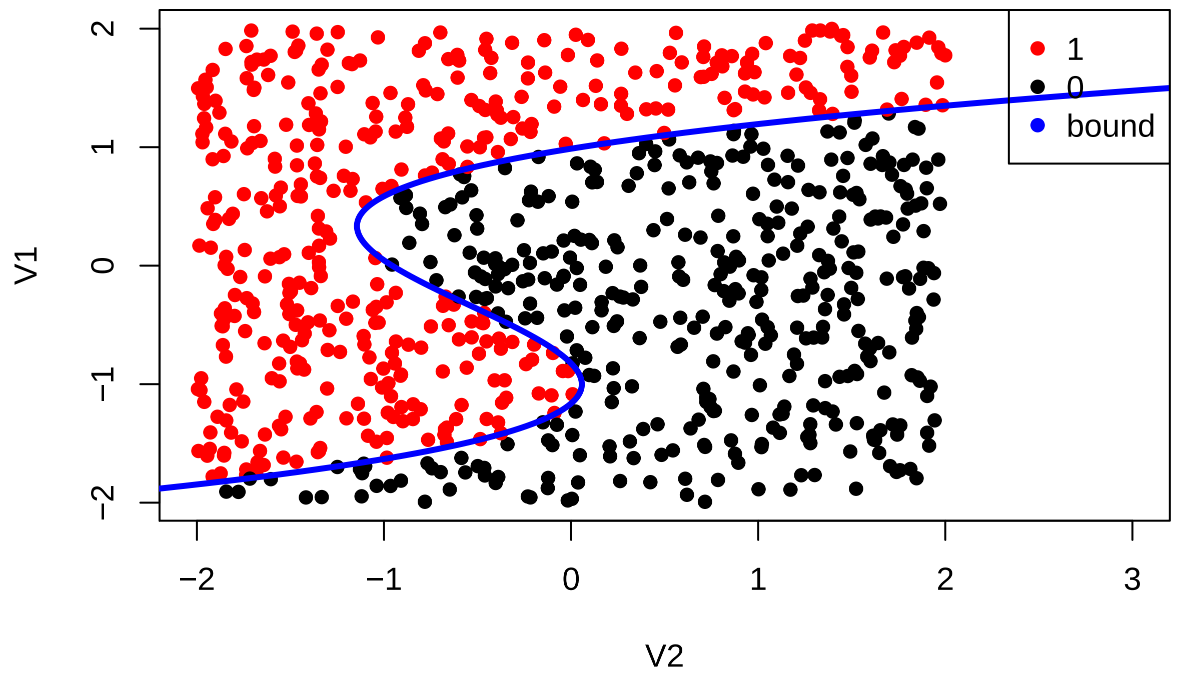
f. Why are neither of the decision boundaries from (e) adequate? What additional predictors can you pass to logistic regression in order for it to do a better job of separating the classes? (Hint: draw a curve between the classes by eye... what shape does this have?) Run a logistic regression with these additional predictors, report the training misclassification rate, and draw the new decision boundary. What shape is it?

Neither of the decision boundaries from (e) are adequate because the data does not have a linear or quadratic relationship. Looking at the plot, it appears as if a cubic relationship might be appropriate, so a cubic term will be added to the logistic regression in order to do a better job of separating the classes.

```
#getting data with the cubic term
x.cub = cbind(x, y2, x[,1]^2, x[,1]^3)
colnames(x.cub) = c("V1", "V2", "Y2", "V1.sq", "V1.cu")

#creating training and testing data sets
train.cub = x.cub[train.index,]; train.cub = data.frame(train.cub)
colnames(train.cub) = c("V1", "V2", "y2", "V1.sq", "V1.cub")
test.cub = x.cub[-train.index,]; test.cub = data.frame(test.cub)
colnames(test.cub) = c("V1", "V2", "y2", "V1.sq", "V1.cub")

#running logistic regression on training data
y2.logit.cub = glm(y2 ~ V1 + V2 + V1.sq + V1.cub, data = train.cub,
    family = binomial, control = list(maxit = 50))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
#finding the training misclassification rate
#predicting for the test set
test.cub$predict = predict(y2.logit.cub,
    newdata = test.cub, type="response")

#classifying predictions as 0 or 1
test.cub$predicted = 0
for (i in (1:nrow(test.cub))){
  if (test.cub[i,]$predict > 0.5){test.cub[i,]$predicted = 1}
}
test.cub$diff = test.cub$predicted - test.cub$y2
Incorrect = length(which(test.cub$diff != 0))
MisclassRate = Incorrect/nrow(test.cub)

#getting line to plot
v2.line.cub = (coef(y2.logit.cub)[1]
  + coef(y2.logit.cub)[2]*(v1.values)
  + coef(y2.logit.cub)[4]*(v1.values)^2
  + coef(y2.logit.cub)[5]*(v1.values)^3)/(-coef(y2.logit.cub)[3])

#plotting
plot(V1~V2, col = as.factor(y2), data = data.y2, pch=16, xlim = c(-2, 3))
lines(v1.values~v2.line.cub, lwd = 3, col = "blue")
legend("topright", c("1", "0", "bound"), pch=c(16, 16, 16), col=c("red", "black", "blue"))
```



The missclassification training rate is zero. The decision boundary is a cubic function that does an excellent job of separating the classes.

g. If adding polynomial terms seems to improve the training misclassification rate of the logistic regression rule, why don?t we generally just keep including polynomial terms of higher and higher order? How could we choose how many polynomial terms to include in a principled manner?

We generally don't just keep including polynomial terms of higher and higher order because this makes interpretation difficult. In addition, adding polynomial terms increases the risk of over-fitting the model to the data. We could choose the number of polynomial terms to include using cross-validation.

2. Impurity Measures

a. Suppose that there are only two classes, K = 2. Let p denote the proportion of points in R that are in the first class. Write out each of the above impurity measures as a function of p (and only p).

There are only two classes, so $\hat{p}_1 + \hat{p}_2$ = p + $\hat{p}_2$ = 1, and $\hat{p}_2$ = 1 - p.

Missclassification Error
Suppose $\hat{p}_1 > \hat{p}_2$. Then Missclassification Error = 1 - $\hat{p}_1$ = 1 - p. When $\hat{p}_2 > \hat{p}_1$, the missclassification rate = 1 - $\hat{p}_2$ = 1 - (1 - p) = p.

Gini Index

$\sum_k \hat{p}_k(1-\hat{p}_k) = \hat{p}_1(1-\hat{p}_1) + \hat{p}_2(1-\hat{p}_2) = p(1-p) + \hat{p}_2(1-\hat{p}_2)$
$= p(1-p) + (1-p)(1-(1-p)) = p(1-p) + (1-p)p = 2p(1-p)$

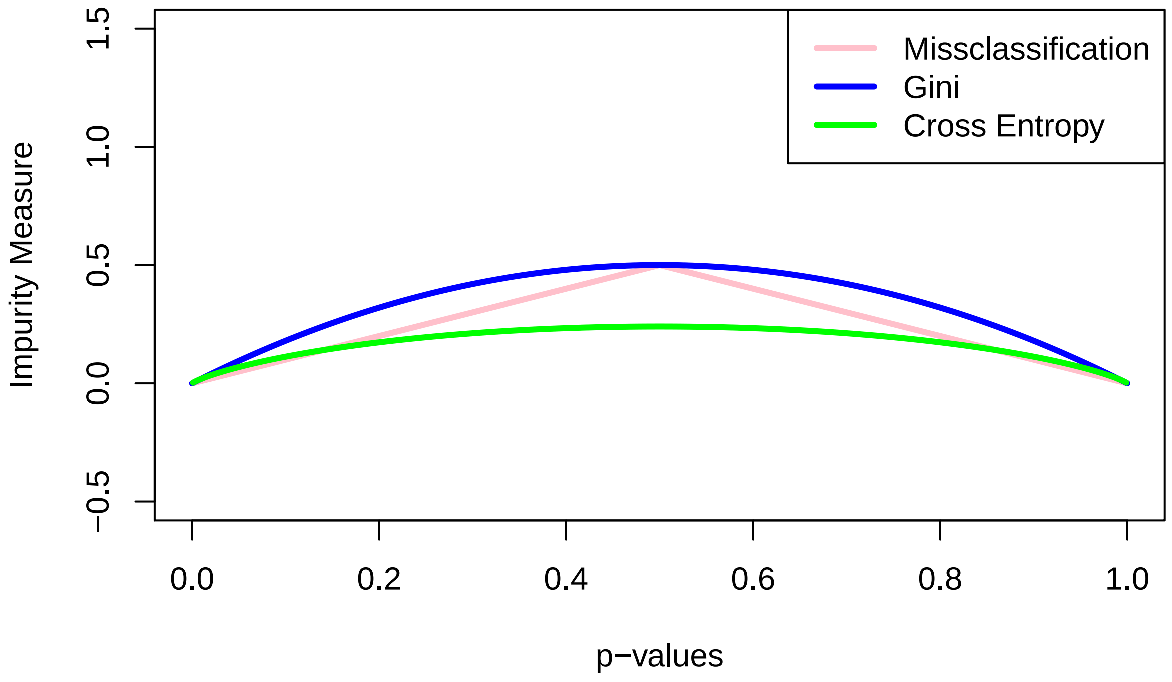
Cross entropy
The scaling constant is found in the following manner:

$-\sum_k \hat{p}_k(1-\hat{p}_k) = -(\hat{p}_1 log(\hat{p}_1) + \hat{p}_2 log(\hat{p}_2))$
$= -(\hat{p}_1 log(\hat{p}_1) + \hat{p}_2(1-\hat{p}_2)) = 0.5log(0.5) + 0.5log(0.5) = log(0.5)$
$-log(0.5)x = 0.5$
$x = -0.5log(0.5)$

Cross entropy is (-(p log(p-1) + (1-p)log(1-p)))(-0.5 log(0.5)).

b. Plot the expressions for misclassification error, Gini index, and cross-entropy that you found in (a) as functions of p. When you plot cross-entropy, you can scale it by a constant so that it is equal to 1/2 when p = 1/2. Are the curves similar?

```r
#getting p.values
p.values = seq(from = 0, to = 1, length.out = 1000)

#calculating the differnt impurity measurements
miss.error = c()
for (i in 1:length(p.values)){
  if (p.values[i] < 0.5){miss.error[i] = p.values[i]}
  else {miss.error[i] = 1 - p.values[i]}
}
gini = (1-p.values)*2*p.values
cross.entropy = -(p.values*log(p.values) + (1-p.values)*log(1-p.values))*(-0.5*log(0.5))

#plotting the impurity measaures
plot(miss.error~p.values, type = "l", col = "pink", lwd = 3, xlim = c(0, 1),
  ylim = c(-0.5, 1.5), xlab = "p-values", ylab = "Impurity Measure")
lines(gini~p.values, type="l", col = "blue", lwd = 3)
lines(cross.entropy ~ p.values, type="l", col="green", lwd = 3)
legend("topright", c("Missclassification", "Gini", "Cross Entropy"),
  lwd = c(3,3,3), col = c("pink", "blue", "green"))
```



The curves are somewhat similar. All three curves have the same shape in that they have a positive slope from 0 to 0.5 and a negative slope to 0.5 to 1. The missclassification rate has a sharp point at 0.5 whereas the Gini index and cross entropy curves are smoother. The Gini index has the largest impurity measure at all of the p-values.

I worked with peers on this assignment and checked answers with them.