Problem 1

(a)

$$R_{tr}(\hat{(\beta)}) = \frac{1}{N}\sum_{i=1}^{n}\left(y_i - \beta^T x_i\right)^2 < \frac{1}{N-p}\sum_{i=1}^{n}\left(y_i - \beta^T x_i\right)^2 = \text{MSE}$$
$$\Rightarrow E(R_{tr}) < E(MSE) = \sigma^2$$

(b)

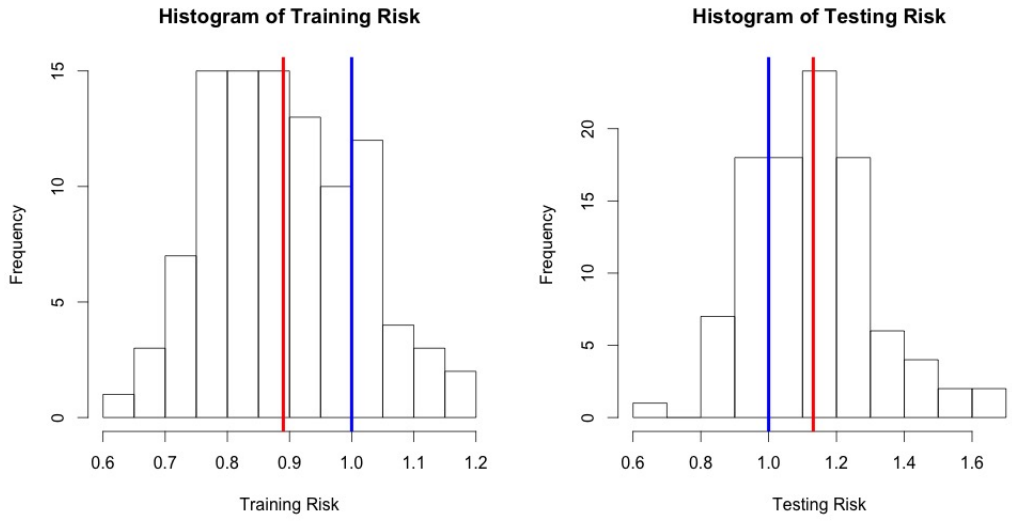
$$\begin{aligned}
E(R_{te}(\hat{(\beta)})) &= \frac{1}{M}E\sum_{i=1}^{M}\left(\tilde{y}_i - \hat{\beta}^T\tilde{x}_i\right)^2 \\
&= \frac{1}{M}E\left[\sum_{i=1}^{M}\left(\left(\tilde{y}_i - \hat{\beta}^T\tilde{x}_i\right) + \left(\beta^T\tilde{x}_i - \beta^T\tilde{x}_i\right)\right)^2\right] \\
&= \frac{1}{M}E\left[\sum_{i=1}^{M}\left(\tilde{y}_i - \beta^T\tilde{x}_i\right)^2 + \sum_{i=1}^{M}\left(\hat{\beta}^T\tilde{x}_i - \beta^T\tilde{x}_i\right)^2\right] \\
&= \frac{1}{M}E\sum_{i=1}^{M}\left(\tilde{y}_i - \beta^T\tilde{x}_i\right)^2 + \frac{1}{M}E\sum_{i=1}^{M}\left(\hat{\beta}^T\tilde{x}_i - \beta^T\tilde{x}_i\right)^2 \\
&= \sigma^2 + (\text{something squared}) > \sigma^2
\end{aligned}$$

(c) On average, the training set error will be *less* than the testing set error. This has many implications in both prediction and inference.

(d) The attached code generates random noise data sets, calculates training and test set risks, then makes a histogram based on 100 simulations. The blue line is $\sigma^2 = 1$, and the red line is the average risk for training and test set.

```
generateData = function(n, m, p){
  #Takes in training and testing sample sizes and number of predictors.
  #Generates simple noise data
  yTrain = rnorm(n) #Responses
  xTrain = matrix(rnorm(n*p), ncol=p) #Predictors

  yTest = rnorm(m) #Responses
  xTest = matrix(rnorm(m*p), ncol=p) #Predictors
  list(yTrain=yTrain, xTrain=xTrain, yTest=yTest, xTest=xTest)
}

calcRisks = function(trainTest){
  # Takes in training and testing data in a list.
  #Unrolls into matrices, then calculates
  # the training and testing errors (scalars). Returns a 2-vector.
  yTrain = trainTest$yTrain
  xTrain = trainTest$xTrain
```

```
    yTest = trainTest$yTest
    xTest = trainTest$xTest

    #Compute training OLS estimate, get residuals/coeffs from it.
    trainLM = lm(yTrain ~ xTrain)
    trainResid = trainLM$resid
    trainCoeffs = trainLM$coefficients

    #Use training OLS estimates to make testing fits. Compute errors.
    testFits = cbind(rep(1,nrow(xTest)), xTest) %*% trainCoeffs
    testResid = yTest - testFits

    riskTrain = 1/length(yTrain) * sum(trainResid^2)
    riskTest = 1/length(yTest) * sum(testResid^2)
    c(riskTrain, riskTest)
}

n = 100 #Train sample size
m = 100 #Test sample size
p = 10 #Number of predictors
nsim = 100 #Number of simulations

simErrors = t(sapply(1:nsim, function(i) calcRisks(generateData(n, m, p))))
par(mfrow=1:2)
hist(simErrors[,1], main='Histogram of Training Risk', xlab='Training Risk')
abline(v = mean(simErrors[,1]), col='red', lwd=4)
abline(v = 1, col='blue', lwd=4)
hist(simErrors[,2], main='Histogram of Testing Risk', xlab='Testing Risk')
abline(v = mean(simErrors[,2]), col='red', lwd=4)
abline(v = 1, col='blue', lwd=4)
```



Problem 2

(a)

(b)

Problem 3

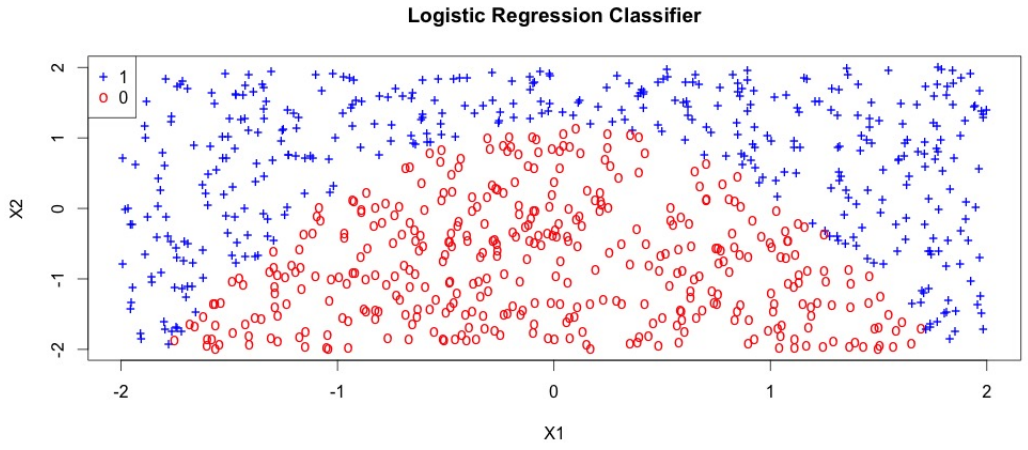
(a) A possible rule would be to classify observations to $Y = 1$ if $P(Y = 1) = \text{sigmoid}(\hat{\beta}^T X) \geq 0.5$. This is equivalent to classifying to $Y = 1$ if $\hat{\beta}^T X \geq 0$.

```
load("hw1prob3.Rdata")
logisticY1 = glm(y1~x, family='binomial')
plot(x[,1], x[,2], col=c("red","blue")[y1+1], pch = c('o', '+')[y1+1],
     ylab="X2", xlab="X1", main='Logistic Regression Classifier')
legend('topleft', legend=unique(y1), col=c('blue', 'red'), pch=c('+', 'o'))
coef(logisticY1) #Classification Rule: b'X >= 0, classify to Y=1.

truePred = cbind(y1, logisticY1$fitted.values >= 0.5) #Obs Y1 and Pred Y1
table(Actual=truePred[,1], Predicted=truePred[,2])
(101+87)/length(y1)
```

Using our logistic regression, classify to $Y = 1$ if $0.127 + 0.063 \cdot x_1 + 1.390 \cdot x_2 \geq 0$. The confusion matrix:

	Predict 0	Predict 1
Actual 0	308	87
Actual 1	101	304

The misclassification rate is $(101 + 87)/800 = .235$.

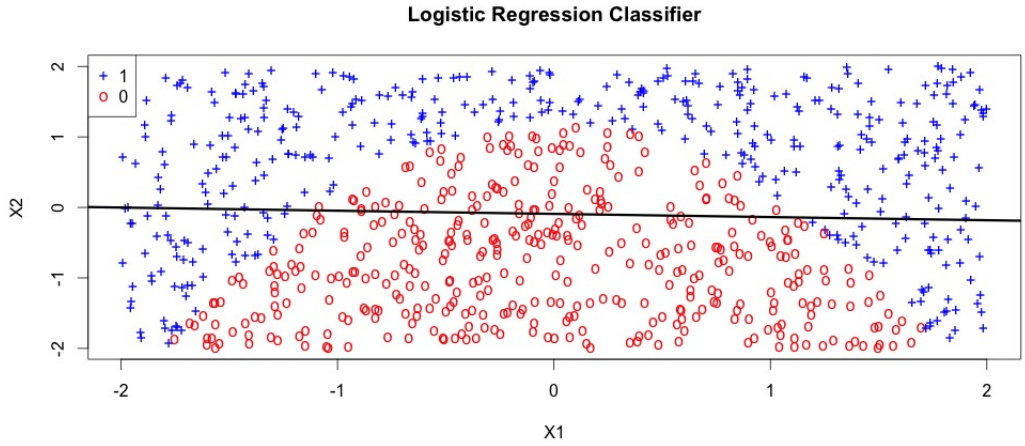


(b) Classification rule drawn on the plot, corresponds to the line

$$x_2 = -0.063x_1/1.390 - 0.127/1.390.$$

This is obtained by taking the classification rule and converting it to slope-intercept form.

```
plot(x[,1], x[,2], col=c("red","blue")[y1+1], pch = c('o', '+')[y1+1],
     ylab="X2", xlab="X1", main='Logistic Regression Classifier')
legend('topleft', legend=unique(y1), col=c('blue', 'red'), pch=c('+', 'o'))

logistCVSlope = -coef(logisticY1)[2] / coef(logisticY1)[3]
logisticCVInt = -coef(logisticY1)[1] / coef(logisticY1)[3]
abline(a = logisticCVInt, b = logistCVSlope, col='black', lty=1, lwd=3)
```



The classifier is clearly nonlinear (it looks like it could be quadratic or periodic with a restricted support), and yet we fit a linear rule to the data set. Not smart!

(c) Similar to the plotting method described in (b), convert general form to slope-intercept form.

```
logisticY1.X1Square = glm(y1~x + I(x[,1]^2), family='binomial')
truePred = cbind(y1, logisticY1.X1Square$fitted.values >= 0.5)
table(Actual=truePred[,1], Predicted=truePred[,2])
```

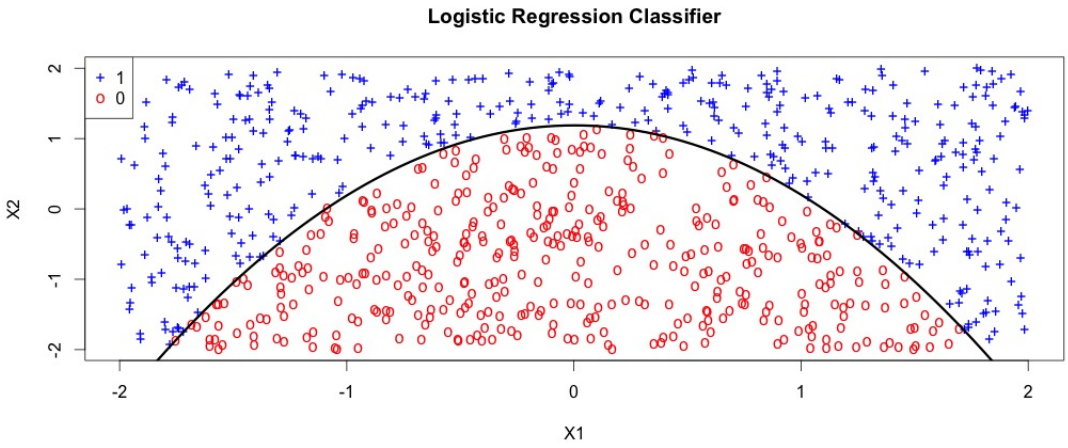
We arrive at the conclusion that we classify to $Y = 1$ if $-2322.5 - 11.6x_1 + 1955.6x_2 + 1937.1x_1^2 \geq 0$.

	Predict 0	Predict 1
Actual 0	395	0
Actual 1	0	405

There is a perfect classification rate (0% misclassification rate). This is definitely a better classifier, not just based on the classification rate, but it is clear that this classifier much more faithfully captures the relationship of y_1 on (x_1, x_2).

(d)
```
plot(x[,1], x[,2], col=c("red","blue")[y1+1], pch = c('o', '+')[y1+1],
     ylab="X2", xlab="X1", main='Logistic Regression Classifier')
legend('topleft', legend=unique(y1), col=c('blue', 'red'), pch=c('+', 'o'))
```

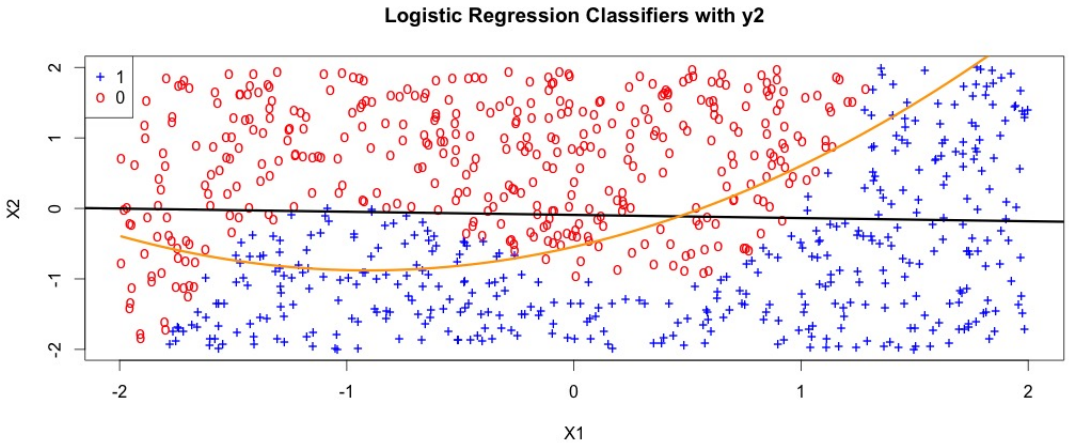
```r
a = -coef(logisticY1.X1Square)[1] / coef(logisticY1.X1Square)[3]
b = -coef(logisticY1.X1Square)[2] / coef(logisticY1.X1Square)[3]
c = -coef(logisticY1.X1Square)[4] / coef(logisticY1.X1Square)[3]
curve(a + b*x + c*x^2, from=min(x[,1]), to=max(x[,1]), add = TRUE, lwd=3)
```



The classifier has a quadratic shape in x_1.

(e) What happens when we look at y_2 instead of y_1?

```r
logisticY2 = glm(y2~x, family='binomial')
logisticY2.X1Square = glm(y2~x + I(x[,1]^2), family='binomial')
plot(x[,1], x[,2], col=c("red","blue")[y2+1], pch = c('o', '+')[y2+1],
     ylab="X2", xlab="X1", main='Logistic Regression Classifiers with y2')
legend('topleft', legend=unique(y1), col=c('blue', 'red'), pch=c('+', 'o'))
a = -coef(logisticY2)[1] / coef(logisticY2)[3]
b = -coef(logisticY2)[2] / coef(logisticY2)[3]
abline(a = logisticCVInt, b = logistCVSlope, col='black', lty=1, lwd=3)
a = -coef(logisticY2.X1Square)[1] / coef(logisticY2.X1Square)[3]
b = -coef(logisticY2.X1Square)[2] / coef(logisticY2.X1Square)[3]
c = -coef(logisticY2.X1Square)[4] / coef(logisticY2.X1Square)[3]
curve(a + b*x + c*x^2, from=min(x[,1]), to=max(x[,1]), add = TRUE, lwd=3, col='orange')
truePredLinear = cbind(y2, logisticY2$fitted.values >= 0.5)
mean(truePredLinear[,1] != truePredLinear[,2])
truePredQuadratic = cbind(y2, logisticY2.X1Square$fitted.values >= 0.5)
mean(truePredQuadratic[,1] != truePredQuadratic[,2])
```



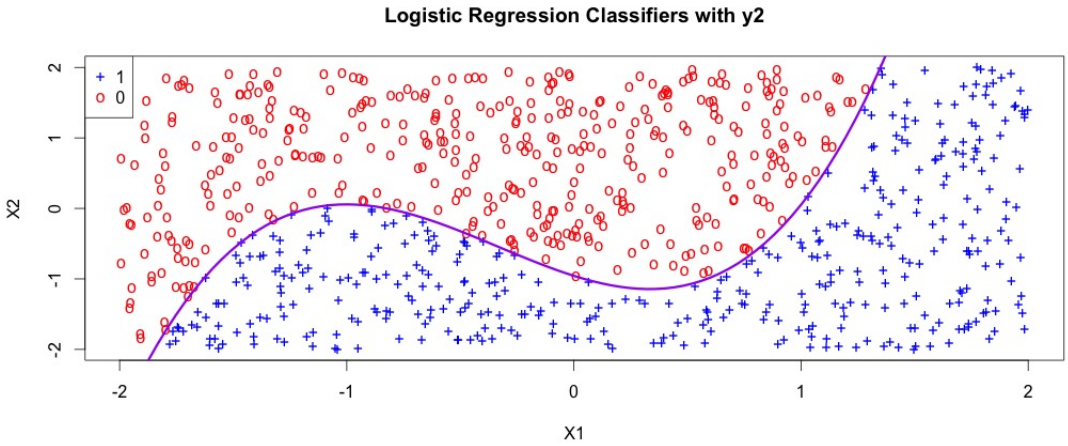
The linear rule has a misclassification rate of 18.5% while the quadratic rule has a misclassification of 13.75%.

(f) Neither rule is particularly good, because neither captures the underlying structure of the data, which appears to be higher degree than just quadratic. One approach we can do is to fit a cubic (third-order) term, which is what the relationship seems to be. If we try third-order, and classify $Y_2 = 1$ if

$$-1455 - 1541x_1 - 1521x_2 + 1537x_1^2 + 1537x_1^3 \geq 0,$$

we can achieve perfect classification.

```
logisticY2.X1Third = glm(y2~x + I(x[,1]^2) + I(x[,1]^3), family='binomial')
plot(x[,1], x[,2], col=c("red","blue")[y2+1], pch = c('o', '+')[y2+1],
     ylab="X2", xlab="X1", main='Logistic Regression Classifiers with y2')
legend('topleft', legend=unique(y1), col=c('blue', 'red'), pch=c('+', 'o'))
a = -coef(logisticY2.X1Third)[1] / coef(logisticY2.X1Third)[3]
b = -coef(logisticY2.X1Third)[2] / coef(logisticY2.X1Third)[3]
c = -coef(logisticY2.X1Third)[4] / coef(logisticY2.X1Third)[3]
d = -coef(logisticY2.X1Third)[5] / coef(logisticY2.X1Third)[3]
curve(a + b*x + c*x^2 + d*x^3, from=min(x[,1]), to=max(x[,1]), add = TRUE, lwd=3, col='purple')
truePredCubic = cbind(y2, logisticY2.X1Third$fitted.values >= 0.5)
mean(truePredCubic[,1] != truePredCubic[,2])
```



(g) Why not keep adding more and more polynomial terms? We have seen before that we run the risk of over-fitting by adding more and more terms. That is, we begin to fine-tune a model that performs well in the sample we have, but may not generalize well to data we haven't collected yet. There are a few approaches to dealing with this over-fitting problem. We've already encountered a few in this course, as well as in previous courses. One popular way is to use cross-validation, fitting several candidate models on one set of data (the training set) and validating the model's performance on a set of data that was not used in training (the testing set), and choosing the model that performs best on the test set. Other ways would involve striking some balance between classification accuracy and the size of the model. These include AIC, BIC, Mallow's C_p, and many more.