classification

October 8, 2019

0.1 Classification

A classifier assign new data points to different groups by learning the associations between a set of characteristics and what groups they were assigned to in the past. Classifiers are a important part of regression techniques. Models that classify data into groups have several applications, for example, classifiers are used to separate email into spam and ham, determine what multimedia we may be interested in based on our preferences (think Netflix), and predict whether we're at high risk for cancer.

The types of classifiers we will explore assign data to one of two groups. The first group will be assigned the value 1 and the second group the value 0. For example, patients suspected of cardiomyopathy can be assigned the group 1 and patients not suspected the group 0. Consumers that could be interested in a specific type of marketing ad are assigned to group 1 and potentially uninterested customers assigned the group 0. The assignment to groups with values 0 and 1 is arbitrary.

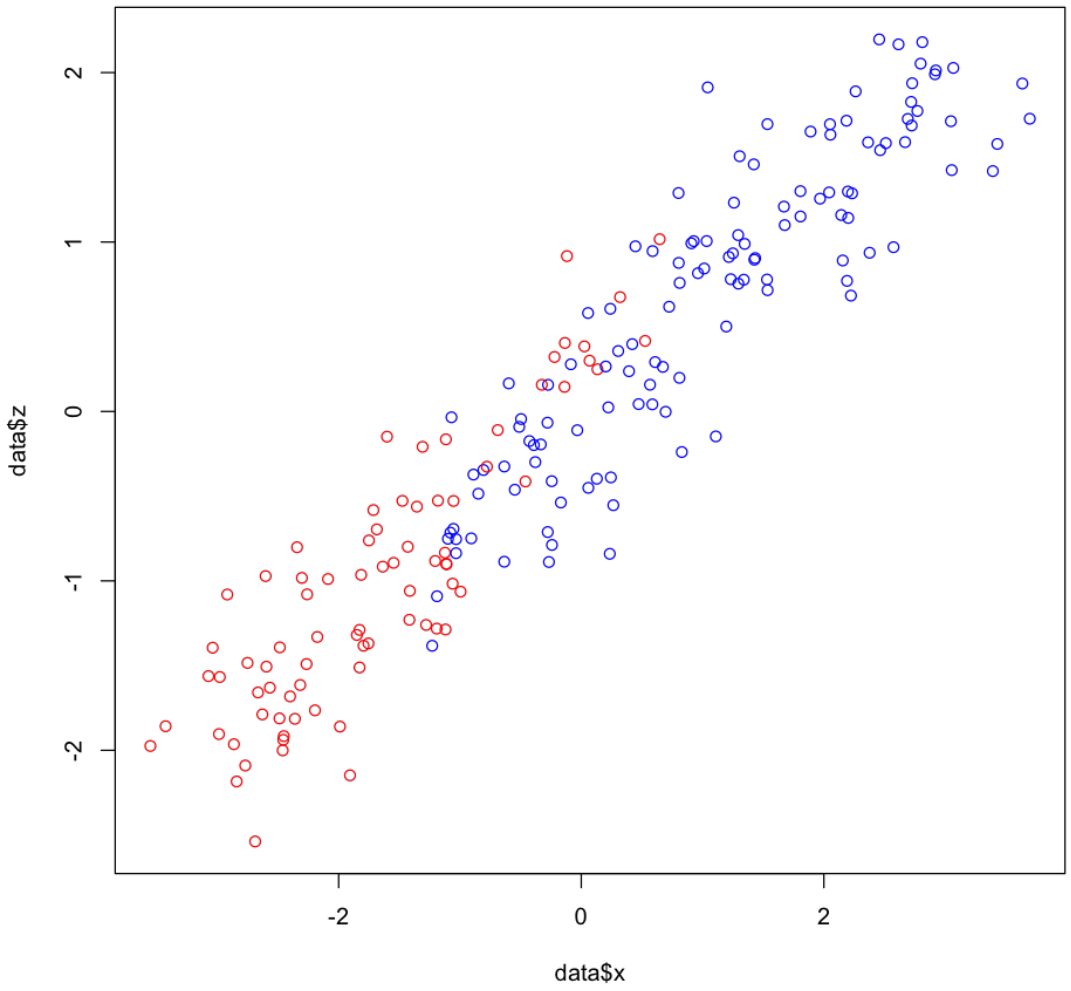
We can try to fit regression models like we have in the past. To predict a continuous variable, we defined class of models that could change based on a fixed set of parameters. Classifiers, assigning new data points to one of two groups, predicts the probability a data point belongs to group 1 versus group 0. Our past regression approaches may not work in this case.

Let's first try our previous approach and see if we come across any problems. Lets look at some data.

0.1.1 EDA

We can color our data points so that the color **blue** corresponds to points that belong to group 1 and **red** for points that belong to group 0.

[60]:
```
data = read.csv('./dataSetClassification.csv')
colors = ifelse(data$C==1,'blue','red')
plot(data$x,data$z,col=colors)
```

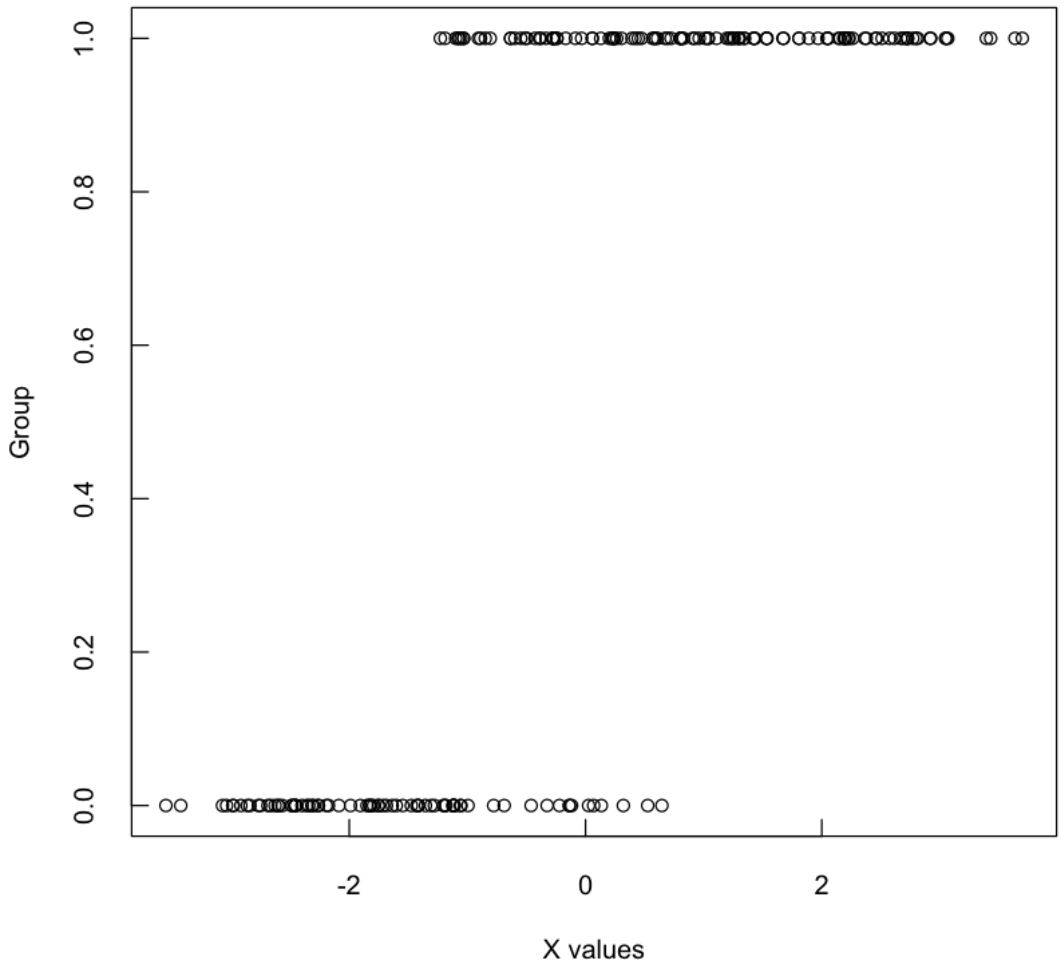


Our classifier should learn how to tell the **blue** points from the **red** points.

The problem is that the relationship between our covariates (x, z) and the group each observation belongs to don'y quite behave when regressing on continuous variables.

Lets plot x versus C.

[74]:
```
options(repr.plot.width=7,repr.plot.height=7)

data = read.csv('./dataSetClassification.csv')
plot(data$x,data$C,xlab='X values',ylab='Group',tck=0.02)
```



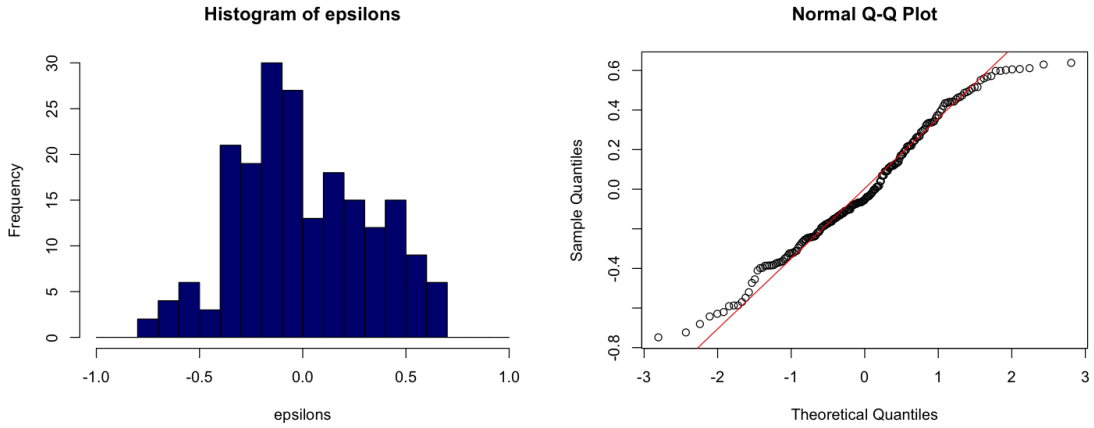
We see a different picture than we have before. The predicted value takes two values: 1 or 0, and we're left to try and predict a binary value. Our techniques for continuous regression may work, but likely not very well.

Lets fit a linear regression, check if it satisfies the LINE assumptions, and report the mean squared error. We can assume the data observations are independent from one another, and so satisfy the *I* assumption.

We can check the *N* assumption by plotting a histogram of residuals and QQ-plot. The histogram of residuals should look "bell-shaped". The QQ plot will compare the empirical cdf of our errors to a theoretical normal cdf, and when plotted against one another, look linear.

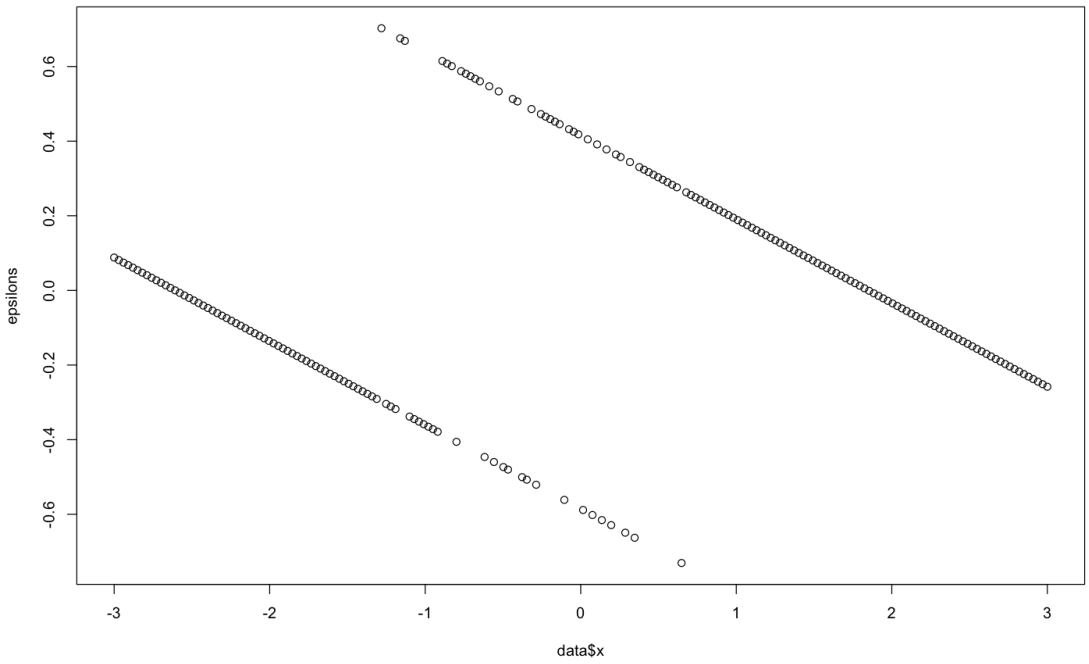
[67]:
```
options(repr.plot.width=12,repr.plot.height=5)
```

```
continuousModel = lm(C~x,data=data)

epsilons = residuals(continuousModel)

par(mfrow=c(1,2))
hist(epsilons,breaks=seq(-1,1,0.1),col='navy',ylim=c(0,30))
qqnorm(epsilons)
qqline(epsilons,col='red')
```



The error distribution looks normal. Let's plot the residuals by x to see if the error distribution has a constant spread independent of x.

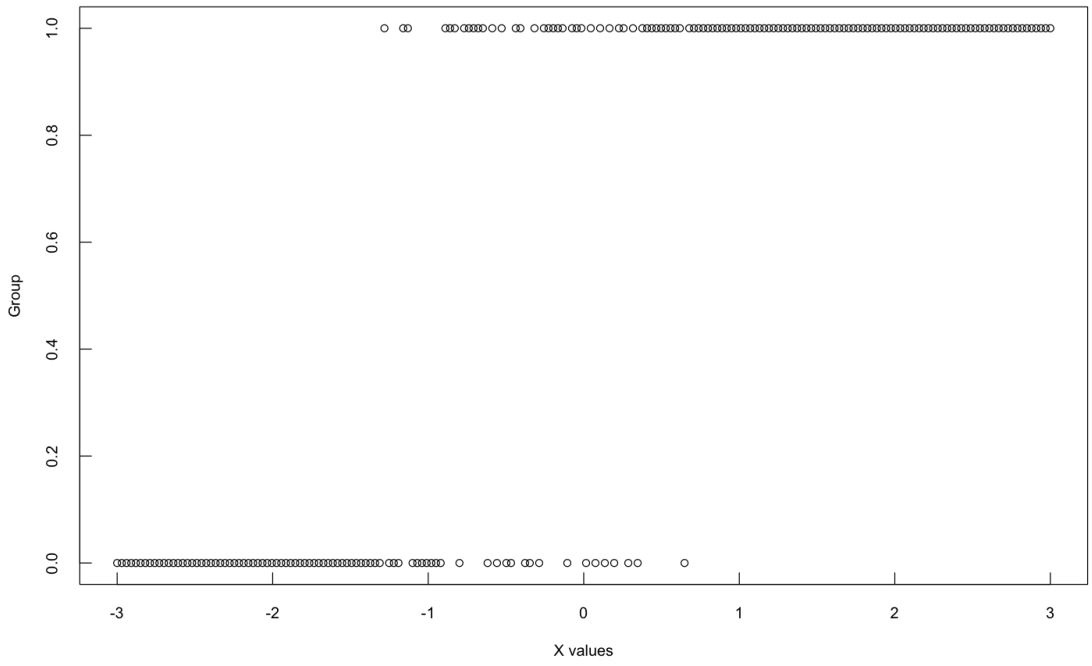
[39]:
```
plot(data$x,epsilons)
```



A pattern is present in the residuals. This may indicate a linear regression model violates the *E* assumption.

Finally, the *L* assumption is violated.

[40]:
```
plot(data$x,data$C,xlab='X values',ylab='Group',tck=0.02)
```



The pattern between x and C is not linear. Fitting a linear regression model fails to satisfy the L and E assumptions.

We saw before that we can fix assumptions like L and E by transforming our data. From the probabilistic form of our regression we see our model assigns a probability to values below 0 and above 1. But the goal is to estimate the probability a new data point belongs to one group vs another—a value that should be between 0 and 1. We need to find a transformation that converts values from $(-\infty, \infty)$ to the interval $[0, 1]$.

0.2 Logistic function

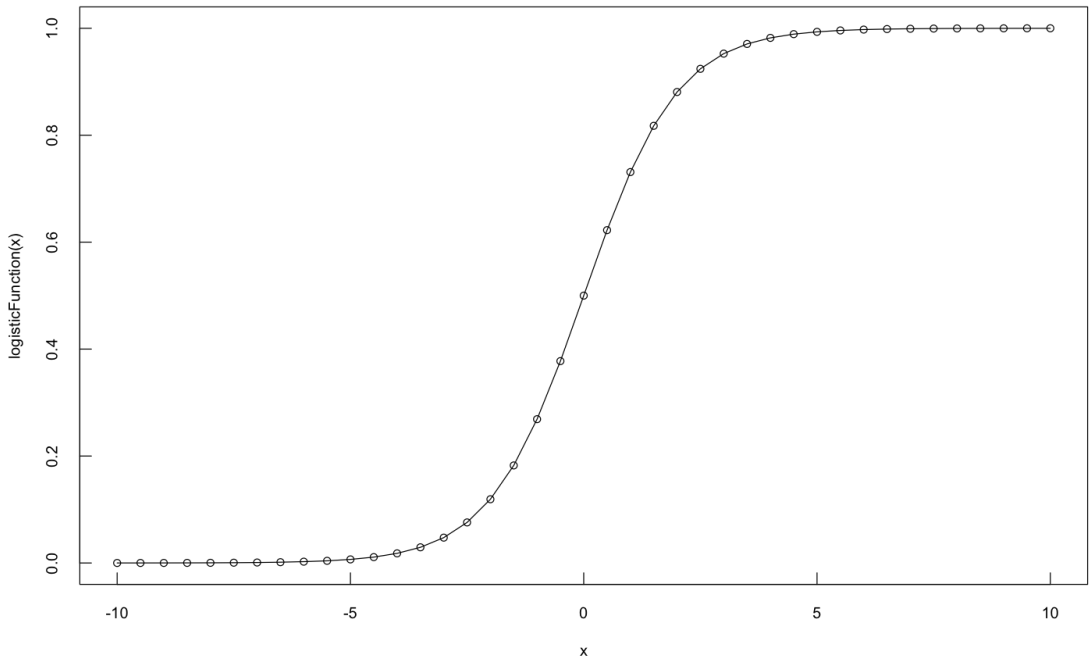
Define the logistic function as

$$f(x) = \frac{e^x}{1 + e^x}$$

and lets plot this function for values from -10 to 10.

[45]:
```
logisticFunction = function(x){
    return(exp(x)/(1+exp(x)))
}

x = seq(-10,10,0.5)
```

```
plot(x,logisticFunction(x),ylim=c(0,1),tck=0.02)
lines(x,logisticFunction(x),ylim=c(0,1))
```



We can see that as x moves towards positive infinity, the logistic function moves towards one. As x moves towards negative infinity the logistic function moves towards 0. The logistic function maps values $(-\infty, \infty)$ to the interval $[0, 1]$.

In typical linear regression predictions can range from negative to positive infinity. Placing y inside the logistic function will guarantee values range between 0 to 1, values consistent with probabilities.

0.3 Logistic Regression

Logistic regression models the probability of assigning a data point to group 1 as

$$p(Y = 1|X) = f(\beta_0 + \beta_1 x)$$

where f is the logistic function, or

$$p(Y = 1|X) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}}.$$

The goal then for logistic function is to find optimal parameters (β_0, β_1) such that $p(Y = 1|X)$ is close to 1 when the data point should be classified into group 1 and close to 0 when the data point should be classified into group 0.

But the above equation looks horrible. We would prefer the right hand side, if possible, to at least look similar to our previous simple linear regression. Our goal is to algebraically manipulate the above function so that

$$g[p(Y = 1|X)] = \beta_0 + \beta_1 x$$

0.4 Finding the log odds

$$p(Y = 1|X) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}}$$

Our first step will multiply both sides by $1 + e^{\beta_0+\beta_1 x}$

$$p(Y = 1|X)\left(1 + e^{\beta_0+\beta_1 x}\right) = e^{\beta_0+\beta_1 x} \tag{1}$$
$$p(Y = 1|X) + p(Y = 1|X)e^{\beta_0+\beta_1 x} = e^{\beta_0+\beta_1 x} \tag{2}$$
$$p(Y = 1|X) = e^{\beta_0+\beta_1 x}\left[1 - p(Y = 1|X)\right] \tag{3}$$
$$\frac{p(Y = 1|X)}{1 - p(Y = 1|X)} = e^{\beta_0+\beta_1 x} \tag{4}$$

Our final step takes the logarithm of both sides

$$\frac{p(Y = 1|X)}{1 - p(Y = 1|X)} = e^{\beta_0+\beta_1 x} \tag{5}$$
$$\log\left[\frac{p(Y = 1|X)}{1 - p(Y = 1|X)}\right] = \beta_0 + \beta_1 x \tag{6}$$

The function g we were looking for is

$$g(x) = x/(1 - x)$$

where x is the probability a data point belongs to group 1. When x is a probability, the above function g is called the odds. For example, if the probability of doing well on your midterm is 80% the odds is then

$$0.80/(1 - 0.80) = 0.8/0.2 = 4 : 1,$$

you have a 4 to 1 chance of doing well on your midterm.

The $\left[\frac{p(Y=1|X)}{1-p(Y=1|X)}\right]$ is the **odds** of a data point belonging to group 1 versus group 0, and $\log\left[\frac{p(Y=1|X)}{1-p(Y=1|X)}\right]$ is called the **log odds**.

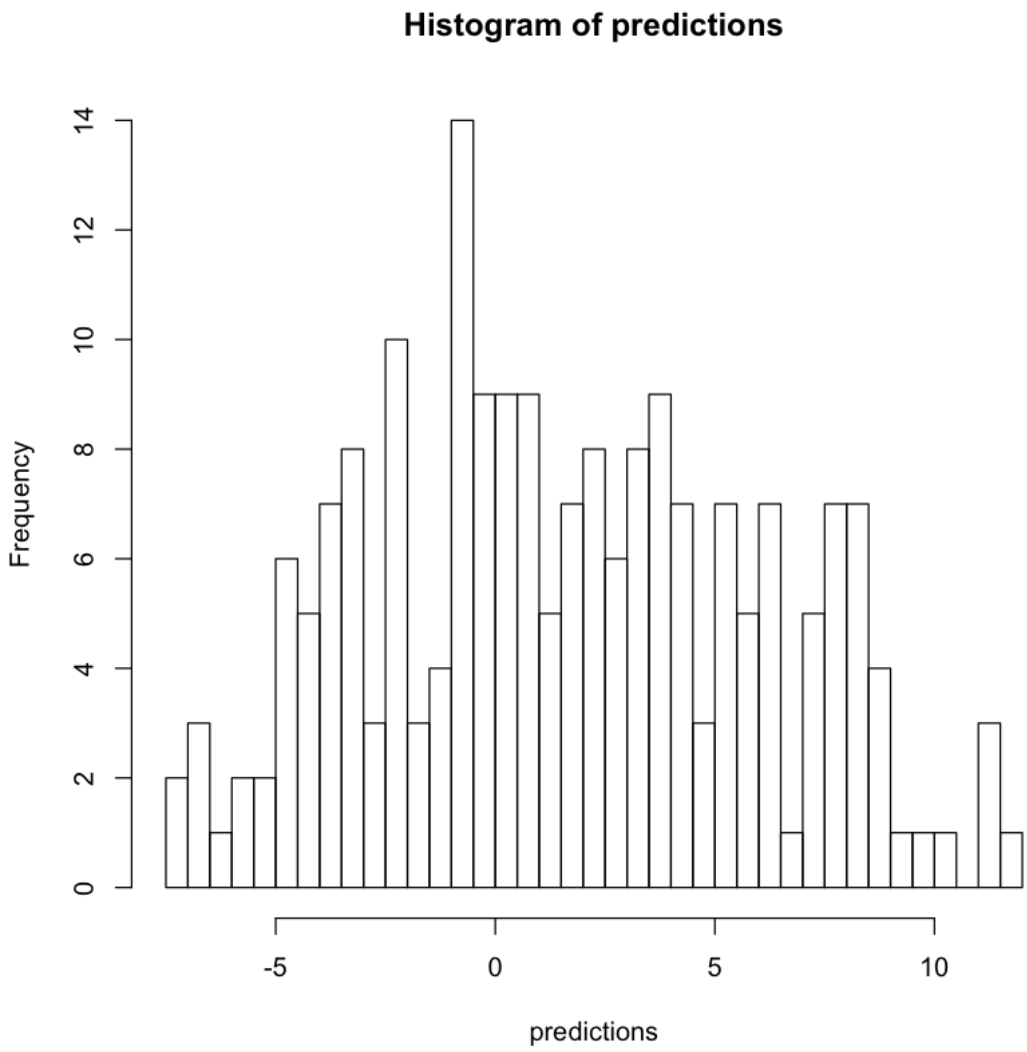
We can fit a logistic regression model in R to find these optimal coefficients.

[78]:
```
model <- glm(C ~ x+z, data = data, family = binomial)
summary(model)
```

```
Call:
glm(formula = C ~ x + z, family = binomial, data = data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4030  -0.2434   0.0288   0.2583   2.1352

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.7172     0.3673   4.675 2.95e-06 ***
x             3.6984     0.7508   4.926 8.38e-07 ***
z            -2.0047     0.7535  -2.661   0.0078 **
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 267.499  on 199  degrees of freedom
Residual deviance:  99.256  on 197  degrees of freedom
AIC: 105.26

Number of Fisher Scoring iterations: 7
```

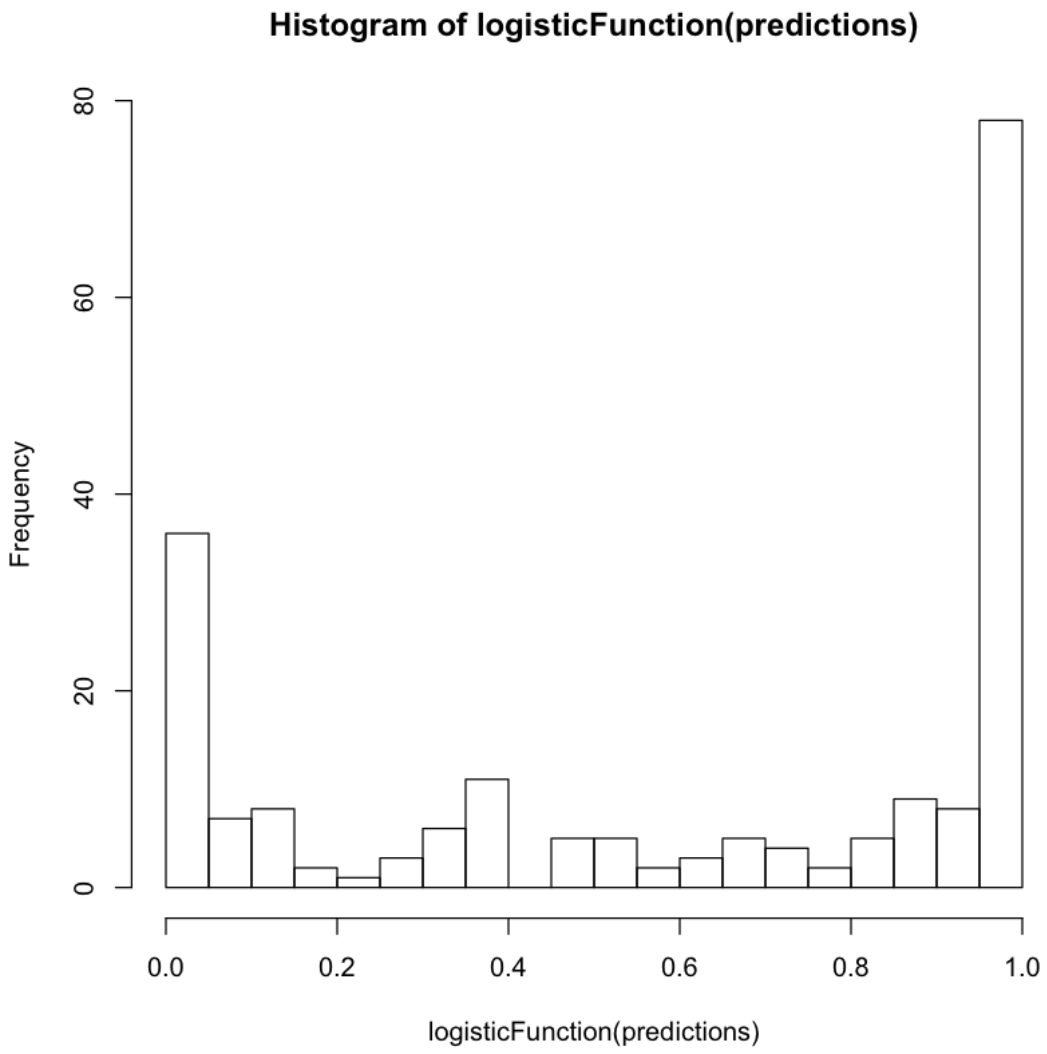
The model is telling us that both x and z are important predictors of a data points membership to either class 0 or 1. But the model and parameters $(\beta_0 + \beta_1 x + \beta_2 z)$ estimated here transform (x, z) to a log odds. We can see this by drawing a histogram of predictions on our training data.

[85]:
```
predictions = predict(model)
hist(predictions,30)
```



Predictions range from less than -5 to over 10, clearly not in the range for probabilities. We need to transform our predictions back, from the log odds, to our original probability scale.

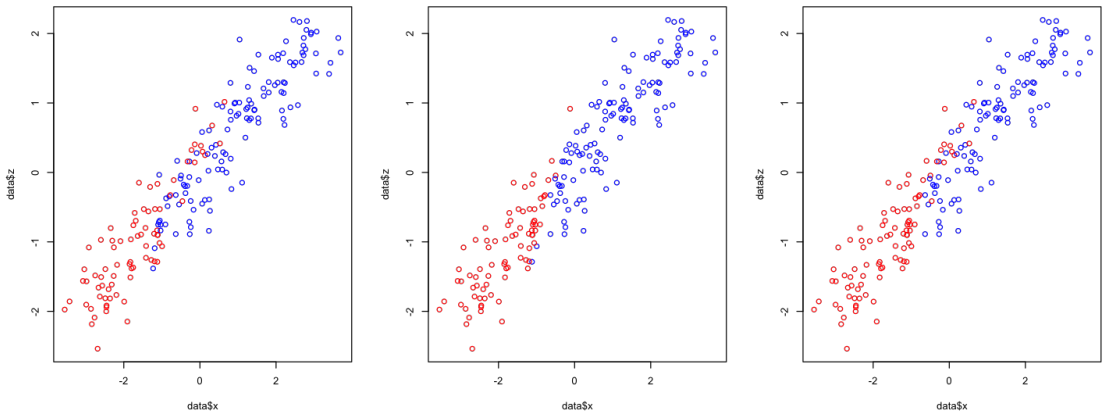
[86]:
```
hist(logisticFunction(predictions),30)
```



[101]:
```
options(repr.plot.width=12,repr.plot.height=5)
par(mfrow=c(1,3))
colors = ifelse(data$C==1,'blue','red')
plot(data$x,data$z,col=colors)

colors = ifelse(predictions>0,'blue','red')
plot(data$x,data$z,col=colors)

colors = ifelse(data$C==1 & predictions >0,'blue','red')
plot(data$x,data$z,col=colors)

TruePositive  = ifelse(data$C==1 & predictions >0,1,0)
falsePositive = ifelse(data$C==0 & predictions >0,1,0)
```

```
falseNegative = ifelse(data$C==1 & predictions <0,1,0)
TrueNegative  = ifelse(data$C==0 & predictions <0,1,0)

Accuracy = (TruePositive + TrueNegative)/(TruePositive + TrueNegative +␣
 ↪falsePositive + falseNegative)
print(mean(Accuracy))
```

```
[1] 0.865
```



[ ]: