Logistic Regression

We move now to the classification problem from the regression problem and study the technique ot logistic regression. The setting for the classification problem is the same as that of the regression problem. We have a response variable y and p explanatory variables $x_1, \ldots, x_p$. We collect data from n subjects on these variables.

The only difference between regression and classification is that in classification, the response variable y is binary (takes only two values; usually coded as 0 and 1) while in regression, the response variable is continuous. The explanatory variables, as before, are allowed to be both continuous and discrete.

There are many examples for the classification problem. Two simple examples are given below. We shall look at more examples later on.

0.1 Frogs Dataset

This dataset is available in R via the package DAAG.

```
library(DAAG)
data(frogs)
```

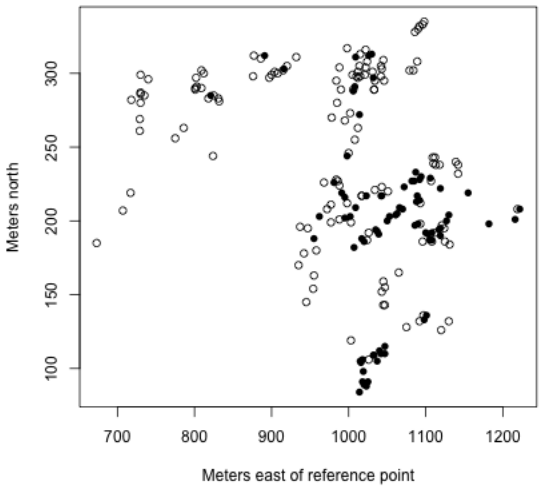
212 sites of the Snowy Mountain area of New South Wales, Australia were surveyed to understand the distribution of the Southern Corroboree frog. The variables are

1. pres.abs – 0/1 indicates whether frogs were found.
2. easting – reference point
3. northing – reference point
4. altitude – altitude in meters
5. distance – distance in meters to nearest extant population
6. NoOfPools– number of potential breeding pools

7. NoOfSites– number of potential breeding sites within a 2 km radius
8. avrain – mean rainfall for Spring period
9. meanmin – mean minimum Spring temperature
10. meanmax – mean maximum Spring temperature

The variable *easting* refers to the distance (in meters) east of a fixed reference point. Similarly *northing* refers to the distance (in meters) north of the reference point. These two variables allow us to plot the data in terms of a map as follows:

```
plot(northing ~ easting, data = frogs, pch = c(1, 16)[frogs$pres.abs +
    1], xlab = "Meters east of reference point", ylab = "Meters north")
```



In this plot, the filled points are for sites where frogs were found.

A natural goal is to under the relation between the *pres.abs* variable and the other geographic and environmental variables. This naturally falls under the classification problem because the response variable *pres.abs* is binary.

0.2 Email Spam Dataset

This dataset is from Chapter 10 of the book *Data Analysis and Graphics* using R. The original dataset is from the UC Irvine Repository of Machine Learning. The original dataset had 4607 observations and 57 explanatory variables. The authors of the book selected 6 of the 57 variables.

```
library(DAAG)
data(spam7)
head(spam7)
```

```
##   crl.tot dollar  bang money n000 make yesno
## 1     278  0.000 0.778  0.00 0.00 0.00     y
## 2    1028  0.180 0.372  0.43 0.43 0.21     y
## 3    2259  0.184 0.276  0.06 1.16 0.06     y
## 4     191  0.000 0.137  0.00 0.00 0.00     y
## 5     191  0.000 0.135  0.00 0.00 0.00     y
## 6      54  0.000 0.000  0.00 0.00 0.00     y
```

```
spam = spam7
```

The main variable here is *yesno* which indicates if the email is spam or not. The other variables are eplanatory variables. They are:

1. crl.tot - total length of words that are in capitals
2. dollar - frequency of the $ symbol, as percentage of all characters
3. bang - freqency of the symbol, as a percentage of all characters,
4. money - freqency of the word *money*, as a percentage of all words,
5. n000 - freqency of the text string *000*, as percentage of all words,
6. make - freqency of the word *make*, as a percentage of all words.

The goal is mainly to predict whether a future email is spam or not based on these explanatory variables. This is once again a classification problem because the response is binary.

There are, of course, many more examples where the classification problem arises naturally.

1 Probabilities, Odds and Log Odds

Here is some basic terminology that is necessary for understanding logistic regression. We denote the probability of an event E by $P(E)$. For example $P(y_i = 1)$ denotes the probability that the i^{th} response variable equals 1.

The odds of an event E is denoted by $odds(E)$ and defined as

$$odds(E) := \frac{P(E)}{1 - P(E)} = \frac{P(E \text{ happens})}{P(E \text{ does not happen})}.$$

The log-odds of an event E is simply defined as $\log(odds(E))$.

An important thing to note is that $P(E)$ lies between 0 and 1, the odds $odds(E)$ is only restricted to be nonnegative while there is no restriction on $\log(odds(E))$ i.e., log-odds can both be positive and negative.

Note the following simple formulae relating probability and odds as well as probability and log-odds:

$$P(E) = \frac{odds(E)}{1 + odds(E)} \tag{1}$$

and

$$P(E) = \frac{e^{odds(E)}}{1 + e^{odds(E)}}.$$

2 Logistic Regression

In logistic regression, one models $\log(odds(y_i = 1))$ as a linear function of the explanatory variable values of the i^{th} individual. The R function for logistic regression in R is *glm()* and it is not very different from *lm()* in terms of syntax.

For the frogs dataset,

```
m1 = glm(pres.abs ~ altitude + distance + NoOfPools +
    NoOfSites + avrain + meanmin + meanmax, family = binomial,
    data = frogs)
summary(m1)
```

```
##
## Call:
## glm(formula = pres.abs ~ altitude + distance + NoOfPools + NoOfSites +
##     avrain + meanmin + meanmax, family = binomial, data = frogs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7215  -0.7590  -0.2237   0.8320   2.6789
##
```

```
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.105e+02  1.388e+02   0.796  0.42587
## altitude    -3.086e-02  4.076e-02  -0.757  0.44901
## distance    -4.800e-04  2.055e-04  -2.336  0.01949 *
## NoOfPools    2.986e-02  9.276e-03   3.219  0.00129 **
## NoOfSites    4.364e-02  1.061e-01   0.411  0.68077
## avrain      -1.140e-02  5.995e-02  -0.190  0.84920
## meanmin      4.899e+00  1.564e+00   3.133  0.00173 **
## meanmax     -5.660e+00  5.049e+00  -1.121  0.26224
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 279.99  on 211  degrees of freedom
## Residual deviance: 198.74  on 204  degrees of freedom
## AIC: 214.74
##
## Number of Fisher Scoring iterations: 6
```

glm stands for generalized linear model. Logistic regression is a special case of a generalized linear model; the *family = binomial* clause in the function tells R to fit a logistic regression equation to the data.

Logistic regression fits the model:

$$\log\left(odds(y=1)\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

to the data. It can be thought of as a linear model for the log odds of the event $y = 1$. The parameter β_j is interpreted as the change in log-odds of the event $y = 1$ for a unit change in the variable x_j provided all other explanatory variables are kept unchanged. Equivalently, e^{β_j} can be interpreted as the multiplicative change in odds for a unit change in the variable x_j provided all other explanatory variables are kept unchanged.

The R function provides estimates of the parameters $\beta_0, \ldots, \beta_p$. For example, in the frogs dataset, the estimated coefficient of the variable $NoOfPools$ is 0.02986. This is interpreted as the change in log-odds of the event of finding a frog when the NoOfPools increases by one (provided the other variables remain unchanged). Equivalently, the odds of finding a frog get multiplied by exp(0.02986) = 1.03031 when the NoOfPools increases by one.

Now suppose a new site is found in the area for which the explanatory variable values are: altitude 1700 m, distance 400 m, NoOfPools 30, NoOfSites 8, avrain 150

(mm ??), meanmin 4 (degrees celsius ??) and meanmax is 16 (degrees celsius ??). What can our logistic regression equation say about the presence or absence of frogs in this area? Our logistic regression allows us to calculate the log(*odds*) of finding frogs in this area as:

```
x0 = c(1, 1700, 400, 30, 8, 150, 4, 16)
sum(x0 * m1$coefficients)
```

```
## [1] -13.58643
```

Remember that this is log(*odds*). From here, the odds of finding frogs is calculated as

```
exp(sum(x0 * m1$coefficients))
```

```
## [1] 1.257443e-06
```

These are very low odds. If one wants to obtain an estimate of the **probability** of finding frogs at this new location, we can use the formula (1) as:

```
exp(sum(x0 * m1$coefficients))/(1 + exp(sum(x0 * m1$coefficients)))
```

```
## [1] 1.257441e-06
```

Therefore, we will predict that this species of frog will not be present at this new location.

Similar to fitted values in linear regression, we can obtain fitted probabilities in logistic regression for each of the observations in our sample:

```
m1$fitted.values
```

```
##            2            3            4            5            6
## 9.994421e-01 9.391188e-01 8.683363e-01 7.443973e-01 9.427198e-01
##            7            8            9           10           11
## 7.107780e-01 7.940785e-01 7.134759e-01 6.791648e-01 7.487912e-01
##           12           13           14           15           16
## 7.031597e-01 7.081588e-01 7.051631e-01 7.895711e-01 7.724424e-01
```

```
##           17           18           19           20           21
## 5.410053e-01 4.002440e-01 5.520545e-01 6.308968e-01 6.015028e-01
##           22           23           24           25           26
## 5.282532e-01 4.096861e-01 5.839915e-01 7.693920e-01 5.815080e-01
##           27           28           29           30           31
## 4.784519e-01 6.059715e-01 7.600451e-01 7.538528e-01 7.706771e-01
##           32           33           34           35           36
## 6.434393e-01 6.079763e-01 3.708934e-01 6.463018e-01 5.586491e-01
##           37           38           39           40           41
## 5.183736e-01 8.117926e-01 6.602267e-01 5.538921e-01 5.806614e-01
##           42           43           44           45           46
## 7.341727e-01 6.316477e-01 9.355365e-01 7.829397e-01 5.807378e-01
##           47           48           49           50           51
## 6.410415e-01 7.011278e-01 6.990184e-01 7.945711e-01 5.983744e-01
##           52           53           54           55           56
## 5.464413e-01 6.694340e-01 9.160177e-01 6.497045e-01 7.793881e-01
##           57           58           59           60           61
## 5.989858e-01 8.742394e-01 4.269678e-01 5.230343e-01 4.696082e-01
##           62           63           64           65           66
## 9.058378e-01 8.310582e-01 2.065644e-01 6.630455e-01 2.304770e-01
##           67           68           69           70           71
## 2.055509e-01 4.103215e-01 3.107167e-01 2.173230e-01 6.849344e-02
##           72           73           74           75           76
## 4.689979e-01 1.964217e-01 3.039997e-01 3.140598e-02 2.960040e-02
##           77           78           79           80           81
## 2.764354e-02 9.572332e-02 2.708695e-02 3.894189e-01 4.384454e-01
##           82           83           84           85           86
## 1.801431e-02 2.958477e-02 2.245579e-02 1.716036e-02 3.869230e-01
##           87           88           89           90           91
## 1.422145e-01 8.723242e-02 1.258070e-01 2.402273e-01 4.425048e-02
##           92           93           94           95           96
## 1.245244e-01 6.044002e-02 8.657481e-02 3.093076e-01 7.608919e-01
##           97           98           99          100          101
## 7.581414e-02 2.519808e-01 2.299453e-01 1.333157e-01 3.224717e-01
##          102          103          104          105          106
## 1.382807e-01 6.569506e-01 2.173365e-02 1.894038e-02 1.502382e-02
##          107          108          109          110          111
## 1.980114e-02 1.768114e-02 1.294026e-02 1.940694e-01 9.026678e-02
##          112          113          114          115          116
## 2.401582e-01 1.801129e-01 1.950594e-01 1.012234e-01 1.100011e-01
##          117          118          119          120          121
## 7.922404e-02 4.383750e-01 1.381195e-01 1.202370e-03 7.190976e-03
##          122          123          124          125          126
## 5.285983e-03 1.501354e-02 6.798822e-03 5.557685e-01 3.414967e-01
```

```
##          127          128          129          130          131
## 3.538713e-01 1.215839e-02 3.587468e-03 4.820373e-01 6.801676e-01
##          132          133          134          135          136
## 4.905071e-01 5.450916e-01 6.139790e-01 5.265536e-01 5.113742e-01
##          137          138          139          140          141
## 1.487969e-03 1.537802e-03 9.498472e-02 1.584109e-01 5.019242e-01
##          142          143          144          145          146
## 2.217258e-01 5.499423e-01 8.072697e-02 2.867668e-01 5.892181e-01
##          147          148          149          150          151
## 4.793328e-01 3.172925e-01 2.456935e-01 3.648957e-01 4.426478e-01
##          152          153          154          155          156
## 3.659800e-01 2.737078e-02 1.261571e-03 8.908085e-04 8.917574e-04
##          157          158          159          160          161
## 1.564091e-03 4.022912e-01 3.554192e-01 3.771886e-01 6.108751e-01
##          162          163          164          165          166
## 1.135652e-01 8.305509e-02 3.248273e-01 2.027767e-01 1.320571e-01
##          167          168          169          170          171
## 2.998112e-01 4.705620e-01 4.664971e-01 1.893874e-01 3.202811e-01
##          172          173          174          175          176
## 2.350189e-01 2.264083e-01 1.720698e-01 3.624017e-01 2.497301e-01
##          177          178          179          180          181
## 7.719488e-01 7.008708e-01 7.727554e-01 1.249298e-01 7.968601e-04
##          182          183          184          185          186
## 9.646891e-05 6.891946e-01 3.317216e-01 6.494793e-01 6.489840e-01
##          187          188          189          190          191
## 6.574312e-01 5.476705e-01 4.789413e-02 7.693637e-02 6.853117e-02
##          192          193          194          195          196
## 1.346704e-01 4.074532e-01 3.981571e-02 7.631064e-02 1.221353e-01
##          197          198          199          200          201
## 6.273025e-01 5.747749e-01 4.664689e-01 2.265998e-01 1.467408e-01
##          202          203          204          205          206
## 1.589058e-01 4.219191e-01 3.164506e-01 2.998835e-01 1.540243e-01
##          207          208          209          210          211
## 6.556601e-02 9.745198e-02 2.134439e-01 8.078311e-04 9.752315e-04
##          212          213
## 2.536105e-03 3.889125e-01
```

These fitted values are the fitted probabilities for each observation in our sample. For example, for $i = 45$, we can also calculate the fitted value manually as:

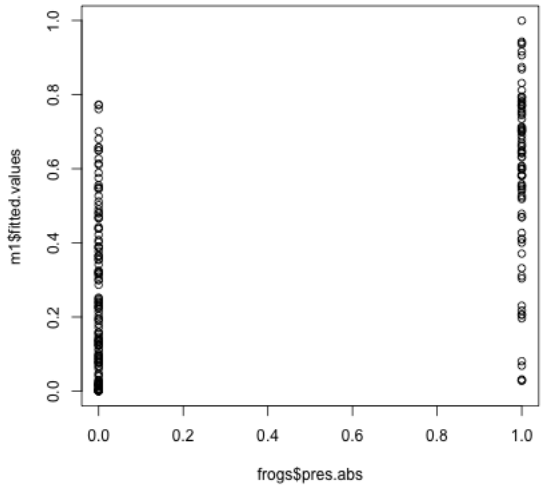
```
i = 45
rrg = c(1, frogs$altitude[i], frogs$distance[i], frogs$NoOfPools[i],
    frogs$NoOfSites[i], frogs$avrain[i], frogs$meanmin[i],
    frogs$meanmax[i])
```

```
eta = sum(rrg * m1$coefficients)
prr = exp(eta)/(1 + exp(eta))
c(prr, m1$fitted.values[i])
```

```
##                   46
## 0.5807378 0.5807378
```

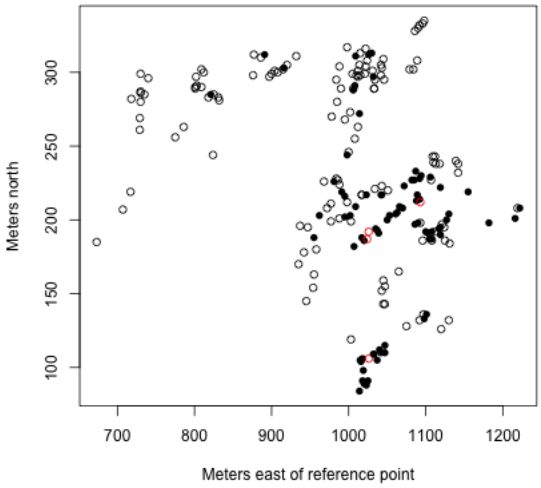
The following plots the fitted values against the actual response:

```
plot(frogs$pres.abs, m1$fitted.values)
```

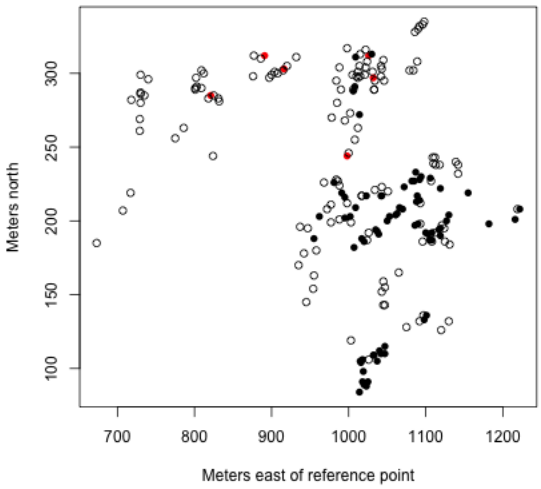


Some of the regions where frogs were present seems to have received very low fitted probability under the model (and conversely, some of the regions with high fitted probability did not actually have any frogs). We can look at these unusual points in the following plot:

```
sel <- frogs$pres.abs == 0 & m1$fitted > 0.7
plot(northing ~ easting, data = frogs, pch = c(1, 16)[frogs$pres.abs +
    1], col = sel + 1, xlab = "Meters east of reference point",
    ylab = "Meters north")
```




```
sel <- frogs$pres.abs == 1 & m1$fitted < 0.2
plot(northing ~ easting, data = frogs, pch = c(1, 16)[frogs$pres.abs +
    1], col = sel + 1, xlab = "Meters east of reference point",
    ylab = "Meters north")
```



3 Prediction in Logistic Regression

Suppose that, for a new site, our logistic regression model predicts that the probability that a frog will be found at that site to be p. How large should p be so that we predict

that frogs will be found at that site? 0.5 sounds like a fair threshold but would 0.6 be better?

Let us now introduce the idea of a **confusion matrix**. Given any chosen threshold, we can form obtain predictions in terms of 0 and 1 for each of the sample observations by applying the threshold to the fitted probabilities given by logistic regression. The confusion matrix is created by comparing these predictions with the actual observed responses. The confusion matrix will have four entries a, b, c, d. a denotes the number of observations where both the observed response as well as our prediction are equal to zero. b denotes the number of observatrions where the observerd response equals 0 but our prediction equals one. c is the number of observations where observed response is one and the prediction is zero. Finally, d is the number of observations where the observed response and the prediction are both equal to 1.

	pred = 0	pred = 1
obs = 0	a	b
obs = 1	c	d

For example, for the frogs data, if we choose the threshold 0.5, then the entries of the confusion matrix can be calculated as:

```
thr = 0.5
y = frogs$pres.abs
yhat = m1$fitted.values
a <- sum((!y) & (yhat <= thr))
b <- sum((!y) & (yhat > thr))
c <- sum((y) & (yhat <= thr))
d <- sum((y) & (yhat > thr))
c(a, b, c, d)
```

```
## [1] 112  21  21  58
```

On the other hand, if we use a threshold of 0.3, the numbers will be:

```
thr = 0.3
y = frogs$pres.abs
yhat = m1$fitted.values
a <- sum((!y) & (yhat <= thr))
b <- sum((!y) & (yhat > thr))
c <- sum((y) & (yhat <= thr))
d <- sum((y) & (yhat > thr))
c(a, b, c, d)
```

```
## [1] 84 49 10 69
```

Note that a and d denote the extent to which the response is in agreement to the thresholded fitted values. And b and c measure the extent to which they disagree. An optimal threshold can be chosen to be one which minimizes $b + c$. We can compute the entries of the confusion matrix for each value of the threshold in the following way.

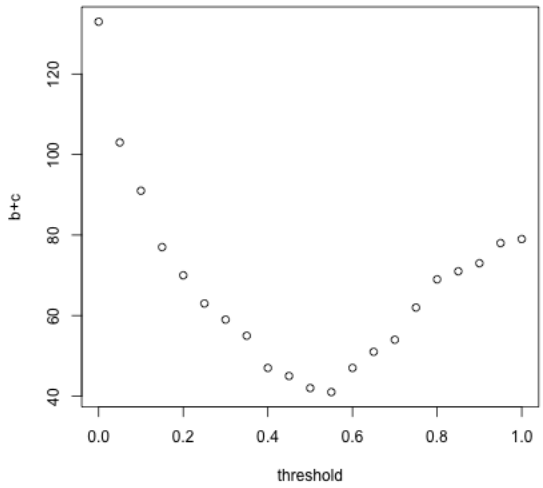
```
conf <- matrix(0, nrow = 21, ncol = 5)
colnames(conf) <- c("thr", "a", "b", "c", "d")
conf[, 1] <- seq(0, 1, by = 0.05)
y <- frogs$pres.abs
yhat <- m1$fitted.values
for (i in 1:21) {
    a <- sum((!y) & (yhat <= conf[i, 1]))
    b <- sum((!y) & (yhat > conf[i, 1]))
    c <- sum((y) & (yhat <= conf[i, 1]))
    d <- sum((y) & (yhat > conf[i, 1]))
    conf[i, 2:5] <- c(a, b, c, d)
}
conf
```

```
##         thr   a   b  c  d
##  [1,] 0.00   0 133  0 79
##  [2,] 0.05  33 100  3 76
##  [3,] 0.10  47  86  5 74
##  [4,] 0.15  61  72  5 74
##  [5,] 0.20  69  64  6 73
##  [6,] 0.25  80  53 10 69
##  [7,] 0.30  84  49 10 69
##  [8,] 0.35  91  42 13 66
##  [9,] 0.40 100  33 14 65
## [10,] 0.45 106  27 18 61
## [11,] 0.50 112  21 21 58
## [12,] 0.55 118  15 26 53
## [13,] 0.60 121  12 35 44
## [14,] 0.65 126   7 44 35
## [15,] 0.70 129   4 50 29
## [16,] 0.75 130   3 59 20
## [17,] 0.80 133   0 69 10
## [18,] 0.85 133   0 71  8
## [19,] 0.90 133   0 73  6
## [20,] 0.95 133   0 78  1
```

```
## [21,] 1.00 133   0 79  0
```

We can then plot the value of $b+c$ for each value of the threshold in the following plot:

```
plot(conf[, 1], conf[, 3] + conf[, 4], xlab = "threshold",
    ylab = "b+c")
```



The smallest value of $b+c$ corresponds to the threshold 0.55. It is sensible therefore to use this threshold for predictions.

Let us now consider the email spam dataset. Recall the dataset:

```
library(DAAG)
data(spam7)
head(spam7)
```

```
##   crl.tot dollar  bang money n000 make yesno
## 1     278  0.000 0.778  0.00 0.00 0.00     y
## 2    1028  0.180 0.372  0.43 0.43 0.21     y
## 3    2259  0.184 0.276  0.06 1.16 0.06     y
## 4     191  0.000 0.137  0.00 0.00 0.00     y
## 5     191  0.000 0.135  0.00 0.00 0.00     y
## 6      54  0.000 0.000  0.00 0.00 0.00     y
```

The response variable is *yesno* which indicates if the email is spam or not. The other variables are eplanatory variables. They are:

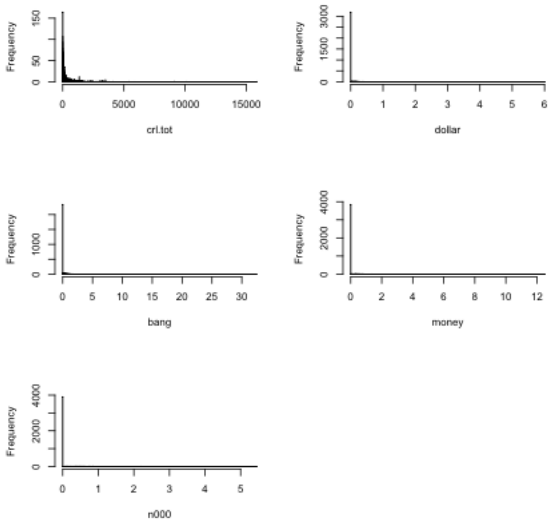
1. crl.tot - total length of words that are in capitals
2. dollar - frequency of the $ symbol, as percentage of all characters
3. bang - freqency of the symbol, as a percentage of all characters,
4. money - freqency of the word *money*, as a percentage of all words,
5. n000 - freqency of the text string *000*, as percentage of all words,
6. make - freqency of the word *make*, as a percentage of all words.

Before fitting a logistic regression model, let us first look at the summary and histograms of the explanatory variables:

```
summary(spam)
```

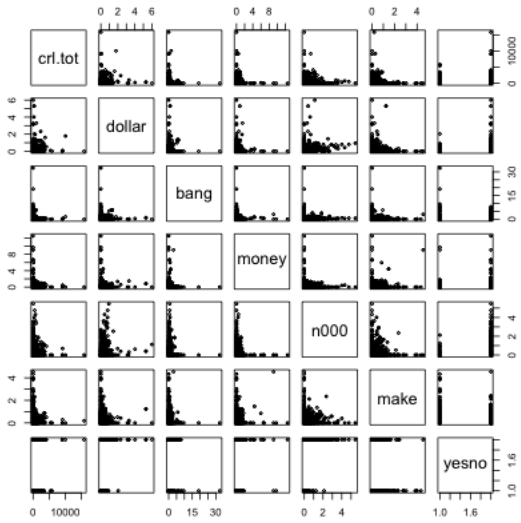
```
##     crl.tot            dollar             bang              money
##  Min.   :    1.0   Min.   :0.00000   Min.   : 0.0000   Min.   : 0.00000
##  1st Qu.:   35.0   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.: 0.00000
##  Median :   95.0   Median :0.00000   Median : 0.0000   Median : 0.00000
##  Mean   :  283.3   Mean   :0.07581   Mean   : 0.2691   Mean   : 0.09427
##  3rd Qu.:  266.0   3rd Qu.:0.05200   3rd Qu.: 0.3150   3rd Qu.: 0.00000
##  Max.   :15841.0   Max.   :6.00300   Max.   :32.4780   Max.   :12.50000
##       n000              make        yesno
##  Min.   :0.0000   Min.   :0.0000   n:2788
##  1st Qu.:0.0000   1st Qu.:0.0000   y:1813
##  Median :0.0000   Median :0.0000
##  Mean   :0.1016   Mean   :0.1046
##  3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :5.4500   Max.   :4.5400
```

```
par(mfrow = c(3, 2))
for (i in 1:5) hist(spam[, i], main = "", xlab = names(spam)[i],
    breaks = 10000)
par(mfrow = c(1, 1))
```



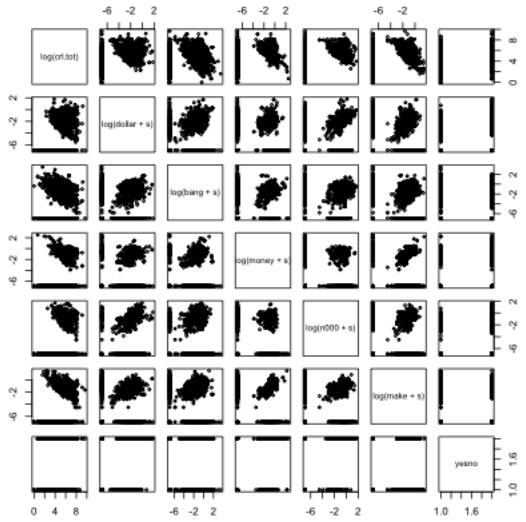
The following is a pairs plot of the variables.

```
pairs(spam, cex = 0.5)
```



It is clear from these plots that the explanatory variables are highly skewed and it is hard to see any structure in these plots. Visualization will be much easier if we take logarithms of the explanatory variables.

```
s = 0.001
pairs(~log(crl.tot) + log(dollar + s) + log(bang +
    s) + log(money + s) + log(n000 + s) + log(make +
    s) + yesno, data = spam, cex = 0.5)
```



We now fit a logistic regression model for *yesno* based on the logged explanatory variables.

```
spam.glm <- glm(yesno ~ log(crl.tot) + log(dollar +
    s) + log(bang + s) + log(money + s) + log(n000 +
    s) + log(make + s), family = binomial, data = spam)
summary(spam.glm)
```

```
##
## Call:
## glm(formula = yesno ~ log(crl.tot) + log(dollar + s) + log(bang +
##     s) + log(money + s) + log(n000 + s) + log(make + s), family = binomial,
##     data = spam)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1657  -0.4367  -0.2863   0.3609   2.7152
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      4.11947    0.36342  11.335  < 2e-16 ***
## log(crl.tot)     0.30228    0.03693   8.185 2.71e-16 ***
## log(dollar + s)  0.32586    0.02365  13.777  < 2e-16 ***
## log(bang + s)    0.40984    0.01597  25.661  < 2e-16 ***
## log(money + s)   0.34563    0.02800  12.345  < 2e-16 ***
## log(n000 + s)    0.18947    0.02931   6.463 1.02e-10 ***
## log(make + s)   -0.11418    0.02206  -5.177 2.25e-07 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6170.2  on 4600  degrees of freedom
## Residual deviance: 3245.1  on 4594  degrees of freedom
## AIC: 3259.1
##
## Number of Fisher Scoring iterations: 6
```

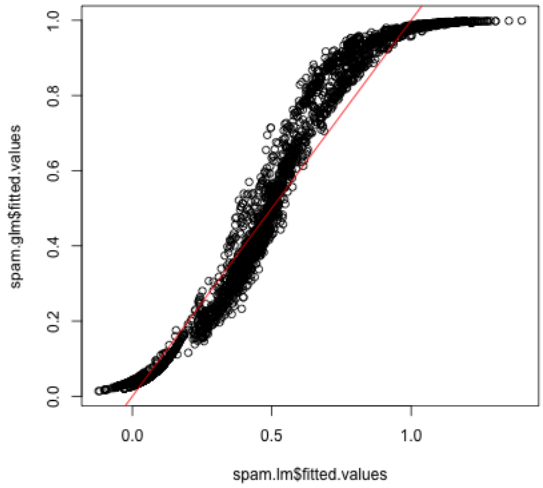
Note that all the variables are significant. We actually could have fitted a linear model as well (even though the response variable is binary).

```
spam.lm <- lm(as.numeric(yesno == "y") ~ log(crl.tot) +
    log(dollar + s) + log(bang + s) + log(money + s) +
    log(n000 + s) + log(make + s), data = spam)
summary(spam.lm)
```

```
##
## Call:
## lm(formula = as.numeric(yesno == "y") ~ log(crl.tot) + log(dollar +
##     s) + log(bang + s) + log(money + s) + log(n000 + s) + log(make +
##     s), data = spam)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.10937 -0.13830 -0.05674  0.15262  1.05619
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.078531   0.034188  31.547  < 2e-16 ***
## log(crl.tot)     0.028611   0.003978   7.193 7.38e-13 ***
## log(dollar + s)  0.054878   0.002934  18.703  < 2e-16 ***
## log(bang + s)    0.064522   0.001919  33.619  < 2e-16 ***
## log(money + s)   0.039776   0.002751  14.457  < 2e-16 ***
## log(n000 + s)    0.018530   0.002815   6.582 5.16e-11 ***
## log(make + s)   -0.017380   0.002370  -7.335 2.61e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3391 on 4594 degrees of freedom
## Multiple R-squared:  0.5193,Adjusted R-squared:  0.5186
## F-statistic: 827.1 on 6 and 4594 DF,  p-value: < 2.2e-16
```

A comparison plot of the fitted values for the linear regression and logistic regression is given below.

```
plot(spam.lm$fitted.values, spam.glm$fitted.values)
abline(c(0, 1), col = "red")
```



Note that some of the fitted values for the linear model are less than 0 and some are more than one. We can formally compare the prediction performance of the linear model and the generalized linear model by the confusion matrix. Recall the function to compute the confusion matrix:

```
confusion <- function(y, yhat, thres) {
    n <- length(thres)
    conf <- matrix(0, length(thres), ncol = 4)
    colnames(conf) <- c("a", "b", "c", "d")
    for (i in 1:n) {
        a <- sum((!y) & (yhat <= thres[i]))
        b <- sum((!y) & (yhat > thres[i]))
        c <- sum((y) & (yhat <= thres[i]))
        d <- sum((y) & (yhat > thres[i]))
        conf[i, ] <- c(a, b, c, d)
    }
    return(conf)
}
```

For various thresholds on the fitted values, the confusion matrices of linear regression and logistic regression can be computed as:

```
v <- seq(0.05, 0.95, by = 0.05)
y <- as.numeric(spam$yesno == "y")
glm.conf <- confusion(y, spam.glm$fitted, v)
glm.conf
```

```
##           a    b    c    d
##  [1,]  769 2019   32 1781
##  [2,] 1709 1079  151 1662
##  [3,] 1825  963  164 1649
##  [4,] 1896  892  174 1639
##  [5,] 1996  792  189 1624
##  [6,] 2079  709  210 1603
##  [7,] 2231  557  247 1566
##  [8,] 2382  406  295 1518
##  [9,] 2536  252  351 1462
## [10,] 2607  181  406 1407
## [11,] 2642  146  478 1335
## [12,] 2671  117  546 1267
## [13,] 2685  103  623 1190
## [14,] 2700   88  667 1146
## [15,] 2714   74  704 1109
## [16,] 2735   53  771 1042
## [17,] 2747   41  901  912
## [18,] 2767   21 1027  786
## [19,] 2775   13 1235  578
```

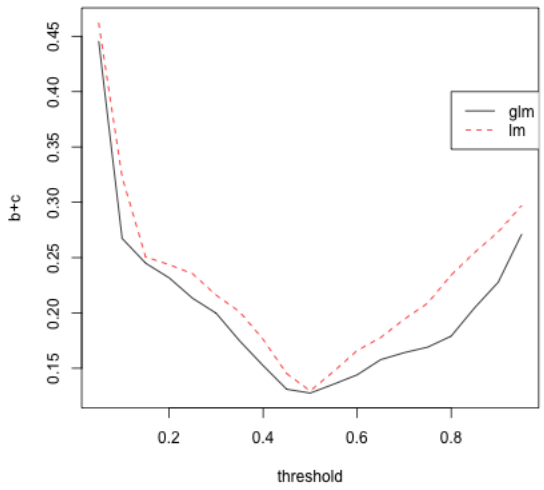
```
lm.conf <- confusion(y, spam.lm$fitted, v)
lm.conf
```

```
##           a    b    c    d
##  [1,]  683 2105   22 1791
##  [2,] 1400 1388   97 1716
##  [3,] 1798  990  163 1650
##  [4,] 1833  955  166 1647
##  [5,] 1876  912  171 1642
##  [6,] 1972  816  178 1635
##  [7,] 2058  730  195 1618
##  [8,] 2235  553  257 1556
##  [9,] 2450  338  330 1483
## [10,] 2633  155  441 1372
## [11,] 2676  112  565 1248
## [12,] 2697   91  673 1140
```

```
## [13,] 2722   66  752 1061
## [14,] 2738   50  844  969
## [15,] 2755   33  928  885
## [16,] 2767   21 1057  756
## [17,] 2774   14 1158  655
## [18,] 2780    8 1249  564
## [19,] 2782    6 1361  452
```

A plot of the misclassification error ($= b + c$) is now given by:

```
matplot(v, cbind((glm.conf[, "b"] + glm.conf[, "c"])/4601,
    (lm.conf[, "b"] + lm.conf[, "c"])/4601), xlab = "threshold",
    ylab = "b+c", type = "l")
legend(0.8, 0.4, lty = 1:2, col = 1:2, c("glm", "lm"))
```



It is clear from this plot that 0.5 is the best threshold for both linear and logistic regression as the misclassification error is minimized at 0.5. Logistic regression seems to be slightly better than linear regression at other thresholds.

The log-transformation on the explanatory variables is quite important in this case. To see this, let us perform a linear regression without the transformations:

```
spam.lm1 = lm(as.numeric(yesno == "y") ~ crl.tot +
    dollar + bang + money + n000 + make, data = spam)
summary(spam.lm1)
```

```
##
## Call:
## lm(formula = as.numeric(yesno == "y") ~ crl.tot + dollar + bang +
##     money + n000 + make, data = spam)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8650 -0.2758 -0.2519  0.4459  0.7499
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.498e-01  7.488e-03  33.365   <2e-16 ***
## crl.tot     1.241e-04  1.058e-05  11.734   <2e-16 ***
## dollar      3.481e-01  2.733e-02  12.740   <2e-16 ***
## bang        1.113e-01  7.725e-03  14.407   <2e-16 ***
## money       1.765e-01  1.440e-02  12.262   <2e-16 ***
## n000        3.218e-01  1.891e-02  17.014   <2e-16 ***
## make        3.212e-02  2.101e-02   1.529    0.126
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4223 on 4594 degrees of freedom
## Multiple R-squared:  0.2543,Adjusted R-squared:  0.2533
## F-statistic: 261.1 on 6 and 4594 DF,  p-value: < 2.2e-16
```

```
summary(spam.lm)
```

```
##
## Call:
## lm(formula = as.numeric(yesno == "y") ~ log(crl.tot) + log(dollar +
##     s) + log(bang + s) + log(money + s) + log(n000 + s) + log(make +
##     s), data = spam)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.10937 -0.13830 -0.05674  0.15262  1.05619
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      1.078531   0.034188  31.547  < 2e-16 ***
## log(crl.tot)     0.028611   0.003978   7.193 7.38e-13 ***
## log(dollar + s)  0.054878   0.002934  18.703  < 2e-16 ***
## log(bang + s)    0.064522   0.001919  33.619  < 2e-16 ***
```

```
## log(money + s)   0.039776   0.002751  14.457  < 2e-16 ***
## log(n000 + s)    0.018530   0.002815   6.582 5.16e-11 ***
## log(make + s)   -0.017380   0.002370  -7.335 2.61e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3391 on 4594 degrees of freedom
## Multiple R-squared:  0.5193,Adjusted R-squared:  0.5186
## F-statistic: 827.1 on 6 and 4594 DF,  p-value: < 2.2e-16
```

There is a noticeable difference between the two R-squared values.

3.1 Precision and Recall

Precision and Recall are two metrics that people look at while comparing classification methods. To understand them, let us take an analogy to search. Suppose you have a set of records and you want to find all records belonging to a particular topic, say logistic regression. Suppose you perform a search and retrieve a set of records. Let d denote the number of relevant records that you have found (i.e., records relating to logistic regression), c denote the number of relevant records that you have not found, b denote the number of irrelevant records that you have found and a denote the number of irrelevant records that you have not found.

The recall of your search is defined as the proportion of relevant records that you have found. In the notation above, recall equals $d/(c+d)$ (the denominator $c+d$ is the total number of relevant records and you have found d of these).

The precision of your search is defined as the proportion of relevant records among the records that you have found. In the notation above, precision equals $d/(b+d)$ (the denominator $b+d$ equals the total number of records that you have found and, of these, d are relevant).

A good search will ideally have both high recall and high precision.

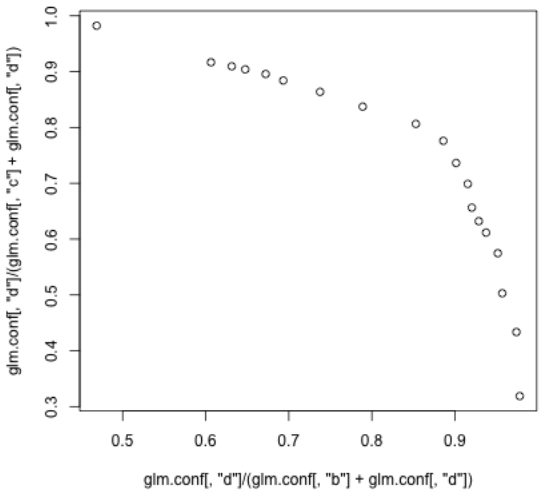
By analogy with search, we shall call $d/(c+d)$ as the recall and $d/(b+d)$ as the precision when a, b, c and d are given by the entries of the confusion matrix. In other words, we are thinking of the observations with response being equal to one as the relevant records and the observations with the predicted response equal to one as the retrieved records.

The precision and recall for the logistic regression model for different values of the threshold are given below:

```
cbind(glm.conf[, "d"]/(glm.conf[, "b"] + glm.conf[,
    "d"]), glm.conf[, "d"]/(glm.conf[, "c"] + glm.conf[,
    "d"]))
```

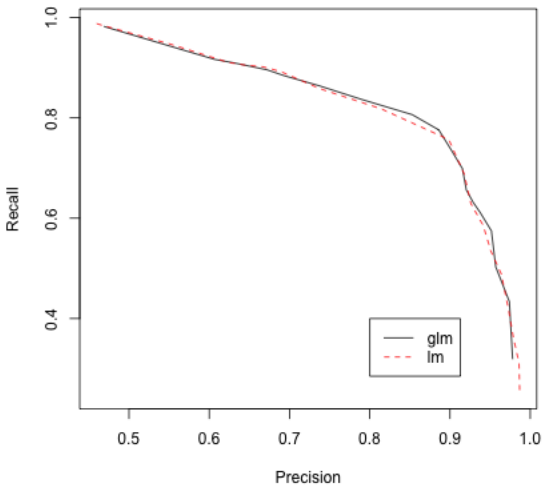
```
##             [,1]      [,2]
##  [1,] 0.4686842 0.9823497
##  [2,] 0.6063480 0.9167126
##  [3,] 0.6313170 0.9095422
##  [4,] 0.6475701 0.9040265
##  [5,] 0.6721854 0.8957529
##  [6,] 0.6933391 0.8841699
##  [7,] 0.7376354 0.8637617
##  [8,] 0.7889813 0.8372863
##  [9,] 0.8529755 0.8063982
## [10,] 0.8860202 0.7760618
## [11,] 0.9014180 0.7363486
## [12,] 0.9154624 0.6988417
## [13,] 0.9203403 0.6563707
## [14,] 0.9286872 0.6321015
## [15,] 0.9374472 0.6116933
## [16,] 0.9515982 0.5747380
## [17,] 0.9569780 0.5030336
## [18,] 0.9739777 0.4335356
## [19,] 0.9780034 0.3188086
```

```
plot(glm.conf[, "d"]/(glm.conf[, "b"] + glm.conf[,
    "d"]), glm.conf[, "d"]/(glm.conf[, "c"] + glm.conf[,
    "d"]))
```



We can plot the precision and recall for both linear and logistic regressions in one plot as follows.

```
matplot(x = cbind(glm.conf[, "d"]/(glm.conf[, "b"] +
    glm.conf[, "d"]), lm.conf[, "d"]/(lm.conf[, "b"] +
    lm.conf[, "d"])), y = cbind(glm.conf[, "d"]/(glm.conf[,
    "c"] + glm.conf[, "d"]), lm.conf[, "d"]/(lm.conf[,
    "c"] + lm.conf[, "d"])), xlab = "Precision", ylab = "Recall",
    type = "l")
legend(0.8, 0.4, lty = 1:2, col = 1:2, c("glm", "lm"))
```



Both methods perform very similarly for this dataset.

4 Residual Deviance and AIC

How do we measure the goodness of fit of a logistic regression equation? In linear regression, we simply looked at RSS (Residual Sum of Squares); is there an analogue in logistic regression? The right analogue is the notion of Residual Deviance.

Let $\hat{p}_1, \ldots, \hat{p}_n$ denote the fitted probabilities in logistic regression. The actual response values are $y_1, \ldots, y_n$ (note that these are binary). If the fit is good, we would expect $\hat{p}_i$ to be small (close to zero) when y_i is 0 and $\hat{p}_i$ to be large (close to one) when y_i is 1. Conversely, if the fit is not good, we would expect $\hat{p}_i$ to be large for some y_i that is zero and $\hat{p}_i$ to be small for some y_i that is 1. A commonly used function for measuring if a probability p is close to 0 is $-2\log p$. This quantity $-2\log p$ is always nonnegative and it becomes very large if p is close to zero. Similarly, one can measure if a probability p is close to 1 by $-2\log(1-p)$. Using these quantities, we measure the quality of fit of $\hat{p}_i$ to y_i by

$$Fit(\hat{p}_i, y_i) = \begin{cases} -2\log \hat{p}_i & : y_i = 1 \\ -2\log\left(1-\hat{p}_i\right) & : y_i = 0 \end{cases}$$

If $Fit(\hat{p}_i, y_i)$ is large, it means that $\hat{p}_i$ is **not** a good fit for y_i. Because y_i is either 0 or 1, the above formula for $Fit(\hat{p}_i, y_i)$ can be written more succinctly as

$$Fit(\hat{p}_i, y_i) = y_i\left(-2\log \hat{p}_i\right) + (1-y_i)\left(-2\log(1-\hat{p}_i)\right).$$

Note that this is for the i^{th} observation. We can get a measure of the overall goodness of fit (across all observations) by simply summing this quantity over $i = 1, \ldots, n$. The resulting quantity is called the Residual Deviance:

$$RD = \sum_{i=1}^{n} Fit(\hat{p}_i, y_i).$$

Just like RSS, small values of RD are preferred and large values indicate lack of fit.

The function *deviance()* can be used in R to calculate deviance. It can, of course, also be calculated manually using the fitted probabilities.

```
library(DAAG)
data(frogs)
m1 = glm(pres.abs ~ altitude + distance + NoOfPools +
    NoOfSites + avrain + meanmin + meanmax, family = binomial,
    data = frogs)
summary(m1)
```

```
##
```

```
## Call:
## glm(formula = pres.abs ~ altitude + distance + NoOfPools + NoOfSites +
##     avrain + meanmin + meanmax, family = binomial, data = frogs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7215  -0.7590  -0.2237   0.8320   2.6789
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.105e+02  1.388e+02   0.796  0.42587
## altitude    -3.086e-02  4.076e-02  -0.757  0.44901
## distance    -4.800e-04  2.055e-04  -2.336  0.01949 *
## NoOfPools    2.986e-02  9.276e-03   3.219  0.00129 **
## NoOfSites    4.364e-02  1.061e-01   0.411  0.68077
## avrain      -1.140e-02  5.995e-02  -0.190  0.84920
## meanmin      4.899e+00  1.564e+00   3.133  0.00173 **
## meanmax     -5.660e+00  5.049e+00  -1.121  0.26224
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 279.99  on 211  degrees of freedom
## Residual deviance: 198.74  on 204  degrees of freedom
## AIC: 214.74
##
## Number of Fisher Scoring iterations: 6
```

RD (Residual Deviance) can be calculated here as

```
deviance(m1)
```

```
## [1] 198.7384
```

One can also calulate it manually as

```
rd = sum(frogs$pres.abs * (-2) * log(m1$fitted.values)) +
    sum((1 - frogs$pres.abs) * (-2) * log(1 - m1$fitted.values))
rd
```

```
## [1] 198.7384
```

Just like the RSS in linear regression, the RD in logistic regression increases as variables are removed from the model. For example, in the frogs dataset, if we remove the variable *NoOfPools*, the RD changes to:

```
m2 = glm(pres.abs ~ altitude + distance + NoOfSites +
    avrain + meanmin + meanmax, family = binomial,
    data = frogs)
summary(m2)
```

```
##
## Call:
## glm(formula = pres.abs ~ altitude + distance + NoOfSites + avrain +
##     meanmin + meanmax, family = binomial, data = frogs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7814  -0.7902  -0.2938   0.9300   3.3822
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.496e+01  1.376e+02   0.254  0.79938
## altitude    -8.375e-03  4.034e-02  -0.208  0.83552
## distance    -5.350e-04  2.066e-04  -2.589  0.00962 **
## NoOfSites    5.748e-02  1.025e-01   0.561  0.57481
## avrain       1.506e-02  5.993e-02   0.251  0.80154
## meanmin      4.358e+00  1.491e+00   2.922  0.00347 **
## meanmax     -2.782e+00  4.986e+00  -0.558  0.57680
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 279.99  on 211  degrees of freedom
## Residual deviance: 210.84  on 205  degrees of freedom
## AIC: 224.84
##
## Number of Fisher Scoring iterations: 6
```

Note that RD increased from 198.7384 to 210.84.

Another useful quantity is the null deviance. The Null Deviance (ND) is the analogue of TSS (Total Sum of Squares) in linear regression. It simply refers to the deviance when there are no explanatory variables i.e., when one does logistic

regression with only the intercept. When there are no explanatory variables, the fitted probabilities are all equal to $\bar{y}$ and so the null deviance is

$$\begin{aligned} ND &= \sum_{i=1}^{n} \left[y_i \left(-2 \log \bar{y} \right) + \left(1 - y_i \right) \left(-2 \log (1 - \bar{y}) \right) \right] \\ &= \left(n\bar{y} \right) \left(-2 \log \bar{y} \right) + \left(n \left(1 - \bar{y} \right) \right) \left(-2 \log \left(1 - \bar{y} \right) \right) \end{aligned}$$

You can easily calculate this and check that this is the same as the null deviance reported in the R summary.

The deviances come with degrees of freedom. The degrees of freedom of RD is $n - p - 1$ (exactly equal to the residual degrees of freedom in linear regression) while the degrees of freedom of ND is $n - 1$.

Now let us introduce the AIC. AIC is short for Akaike Information Criterion after Akaike who introduced it. AIC is useful as a model selection criterion. It is defined as

$$AIC = RD + 2 \left(p + 1 \right)$$

where p denotes the number of explanatory variables.

```
m1$aic
```

```
## [1] 214.7384
```

```
m1$deviance + 2 * (7 + 1)
```

```
## [1] 214.7384
```

5 Variable Selection using AIC

Although the Residual Deviance (RD) measures goodness of fit, it cannot be used for variable selection because the full model will have the smallest RD. The AIC however can be used as a goodness of fit criterion (this involves selecting the model with the smallest AIC). The presence of the term $2(p + 1)$ is the key here. For a model with many explanatory variables, RD will typically be small but the second term $2(p + 1)$ will be large. On the other hand, for a model with few explanatory variables, RD will be large but $2(p + 1)$ will be small. Therefore AIC will not automatically spit out the full model.

In principle, one can go over all possible submodels and select the model with the smallest value of AIC. But this involves going over 2^p models which might be computationally difficult if p is moderate or large. A useful alternative is to use the *step*() function in R. This works by sequentially adding and removing variables while comparing the AIC.

```
step(m1)
```

```
## Start:  AIC=214.74
## pres.abs ~ altitude + distance + NoOfPools + NoOfSites + avrain +
##     meanmin + meanmax
##
##             Df Deviance    AIC
## - avrain     1   198.78 212.78
## - NoOfSites  1   198.91 212.91
## - altitude   1   199.30 213.30
## - meanmax    1   199.97 213.97
## <none>           198.74 214.74
## - distance   1   206.39 220.39
## - meanmin    1   209.60 223.60
## - NoOfPools  1   210.84 224.84
##
## Step:  AIC=212.77
## pres.abs ~ altitude + distance + NoOfPools + NoOfSites + meanmin +
##     meanmax
##
##             Df Deviance    AIC
## - NoOfSites  1   198.94 210.94
## - altitude   1   199.58 211.58
## <none>           198.78 212.78
## - meanmax    1   201.76 213.76
## - distance   1   206.39 218.39
## - meanmin    1   210.75 222.75
## - NoOfPools  1   210.90 222.90
##
## Step:  AIC=210.94
## pres.abs ~ altitude + distance + NoOfPools + meanmin + meanmax
##
##             Df Deviance    AIC
## - altitude   1   199.63 209.63
## <none>           198.94 210.94
## - meanmax    1   201.76 211.76
## - distance   1   209.68 219.68
## - meanmin    1   210.83 220.83
```

```
## - NoOfPools  1   211.22 221.22
##
## Step:  AIC=209.63
## pres.abs ~ distance + NoOfPools + meanmin + meanmax
##
##             Df Deviance    AIC
## <none>           199.63 209.63
## - distance   1   209.73 217.73
## - NoOfPools  1   211.43 219.43
## - meanmax    1   216.10 224.10
## - meanmin    1   226.94 234.94
##
## Call:  glm(formula = pres.abs ~ distance + NoOfPools + meanmin + meanmax,
##     family = binomial, data = frogs)
##
## Coefficients:
## (Intercept)     distance    NoOfPools      meanmin      meanmax
##  14.0074032   -0.0005138    0.0285643    5.6230647   -2.3717579
##
## Degrees of Freedom: 211 Total (i.e. Null);  207 Residual
## Null Deviance:     280
## Residual Deviance: 199.6  AIC: 209.6
```

The default version of the *step*() function only removes variables (analogous to backward elimination). If one wants to add variables as well, the following can be done.

```
step(m1, direction = "both")
```

```
## Start:  AIC=214.74
## pres.abs ~ altitude + distance + NoOfPools + NoOfSites + avrain +
##     meanmin + meanmax
##
##             Df Deviance    AIC
## - avrain     1   198.78 212.78
## - NoOfSites  1   198.91 212.91
## - altitude   1   199.30 213.30
## - meanmax    1   199.97 213.97
## <none>           198.74 214.74
## - distance   1   206.39 220.39
## - meanmin    1   209.60 223.60
## - NoOfPools  1   210.84 224.84
##
```

```
## Step:  AIC=212.77
## pres.abs ~ altitude + distance + NoOfPools + NoOfSites + meanmin +
##     meanmax
##
##             Df Deviance    AIC
## - NoOfSites  1   198.94 210.94
## - altitude   1   199.58 211.58
## <none>           198.78 212.78
## - meanmax    1   201.76 213.76
## + avrain     1   198.74 214.74
## - distance   1   206.39 218.39
## - meanmin    1   210.75 222.75
## - NoOfPools  1   210.90 222.90
##
## Step:  AIC=210.94
## pres.abs ~ altitude + distance + NoOfPools + meanmin + meanmax
##
##             Df Deviance    AIC
## - altitude   1   199.63 209.63
## <none>           198.94 210.94
## - meanmax    1   201.76 211.76
## + NoOfSites  1   198.78 212.78
## + avrain     1   198.91 212.91
## - distance   1   209.68 219.68
## - meanmin    1   210.83 220.83
## - NoOfPools  1   211.22 221.22
##
## Step:  AIC=209.63
## pres.abs ~ distance + NoOfPools + meanmin + meanmax
##
##             Df Deviance    AIC
## <none>           199.63 209.63
## + altitude   1   198.94 210.94
## + avrain     1   199.39 211.39
## + NoOfSites  1   199.58 211.58
## - distance   1   209.73 217.73
## - NoOfPools  1   211.43 219.43
## - meanmax    1   216.10 224.10
## - meanmin    1   226.94 234.94
##
## Call:  glm(formula = pres.abs ~ distance + NoOfPools + meanmin + meanmax,
##     family = binomial, data = frogs)
##
## Coefficients:
```

```
## (Intercept)     distance    NoOfPools      meanmin      meanmax
##  14.0074032   -0.0005138    0.0285643    5.6230647   -2.3717579
##
## Degrees of Freedom: 211 Total (i.e. Null);  207 Residual
## Null Deviance:     280
## Residual Deviance: 199.6  AIC: 209.6
```