Logistic Regression with the Titanic Data

Karen Mazidi

There are many versions of the Titanic data. The one used here was downloaded from this link. and then converted to csv format.

Load the data

```
df <- read.csv("data/titanic3.csv", header=TRUE)
str(df)
```

```
## 'data.frame':    1310 obs. of  14 variables:
##  $ pclass   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ survived : int  1 1 0 0 0 1 1 0 1 0 ...
##  $ name     : Factor w/ 1308 levels "","Abbing, Mr. Anthony",..: 23 25 26 27 28 32 47 48 52 56 ...
##  $ sex      : Factor w/ 3 levels "","female","male": 2 3 2 3 2 3 2 3 2 3 ...
##  $ age      : num  29 0.917 2 30 25 ...
##  $ sibsp    : int  0 1 1 1 1 0 1 0 2 0 ...
##  $ parch    : int  0 2 2 2 2 0 0 0 0 0 ...
##  $ ticket   : Factor w/ 930 levels "","110152","110413",..: 189 51 51 51 51 126 94 17 78 827 ...
##  $ fare     : num  211 152 152 152 152 ...
##  $ cabin    : Factor w/ 187 levels "","A10","A11",..: 45 81 81 81 81 151 147 17 63 1 ...
##  $ embarked : Factor w/ 4 levels "","C","Q","S": 4 4 4 4 4 4 4 4 4 2 ...
##  $ boat     : Factor w/ 28 levels "","1","10","11",..: 13 4 1 1 1 14 3 1 28 1 ...
##  $ body     : int  NA NA NA 135 NA NA NA NA NA 22 ...
##  $ home.dest: Factor w/ 370 levels "","?Havana, Cuba",..: 310 232 232 232 232 238 163 25 23 230 ...
```

Data cleaning

First we subset the data frame because we only care about columns pclass, survived, sex, and age. Then we make survived and pclass factors, sex is already a factor.

```
df <- df[,c(1,2,4,5)]
df$pclass <- factor(df$pclass)
df$survived <- factor(df$survived)
head(df)
```

```
##   pclass survived    sex     age
## 1      1        1 female 29.0000
## 2      1        1   male  0.9167
## 3      1        0 female  2.0000
## 4      1        0   male 30.0000
## 5      1        0 female 25.0000
## 6      1        1   male 48.0000
```

Handle missing values

We first find out how many missing values we have for each of our 4 columns with the sapply() function. The first argument is the object we wish to apply the function to. In this case the function sums the number of NAs for each column of the data frame.

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##   pclass survived      sex      age
```

```
##        1        1        0      264
```

We see that there are no NAs for sex but one NA each for pclass and survived. We can just delete those rows. There are 264 observations out of the total 1310 where we have NA for the age. We could just delete those but that's a lot of data to lose. Instead we will replace the NAs with the median age.

```
df <- df[!is.na(df$pclass),]
df <- df[!is.na(df$survived),]
df$age[is.na(df$age)] <- median(df$age,na.rm=T)
```

Divide into train and test

```
set.seed(1234)
i <- sample(1:nrow(df), 0.75*nrow(df), replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

Build a logistic regression model

```
glm1 <- glm(survived~., data=train, family="binomial")
summary(glm1)
```

```
##
## Call:
## glm(formula = survived ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9509  -0.6567  -0.4336   0.6703   2.4834
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.858516   0.360478  10.704  < 2e-16 ***
## pclass2     -1.417739   0.249787  -5.676 1.38e-08 ***
## pclass3     -2.437512   0.233637 -10.433  < 2e-16 ***
## sexmale     -2.552619   0.175795 -14.520  < 2e-16 ***
## age         -0.042339   0.007198  -5.882 4.06e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1305.05  on 980  degrees of freedom
## Residual deviance:  897.03  on 976  degrees of freedom
## AIC: 907.03
##
## Number of Fisher Scoring iterations: 4
```

Evaluate on the test set

```
probs <- predict(glm1, newdata=test, type="response")
pred <- ifelse(probs>0.5, 1, 0)
acc <- mean(pred==test$survived)
print(paste("accuracy = ", acc))
```

```
## [1] "accuracy =  0.765243902439024"
```

```
table(pred, test$survived)
```

```
##
## pred   0   1
##    0 164  38
##    1  39  87
```

Additional metrics

The confusion matrix in the caret package gives us more information than our simple table above. One of the more useful statistics is the Kappa value which adjusts for the distribution of the data set. In this case the data set was only slightly unbalanced, with about 60% survived to 40$ not.

Note that the vector 'pred' was an integer vector while survived is a factor. The pred vector needs to be converted to a factor for the confusion matrix code.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
confusionMatrix(as.factor(pred), reference=test$survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 164  38
##          1  39  87
##
##                Accuracy : 0.7652
##                  95% CI : (0.7156, 0.8101)
##     No Information Rate : 0.6189
##     P-Value [Acc > NIR] : 1.181e-08
##
##                   Kappa : 0.5031
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8079
##             Specificity : 0.6960
##          Pos Pred Value : 0.8119
##          Neg Pred Value : 0.6905
##              Prevalence : 0.6189
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.6159
##       Balanced Accuracy : 0.7519
##
##        'Positive' Class : 0
##
```

###ROC

The ROC is a curve that plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The AUC is the area under the ROC curve. A good AUC is close to 1 than 0.5. Also we

like to see the ROC shoot up rather quickly.

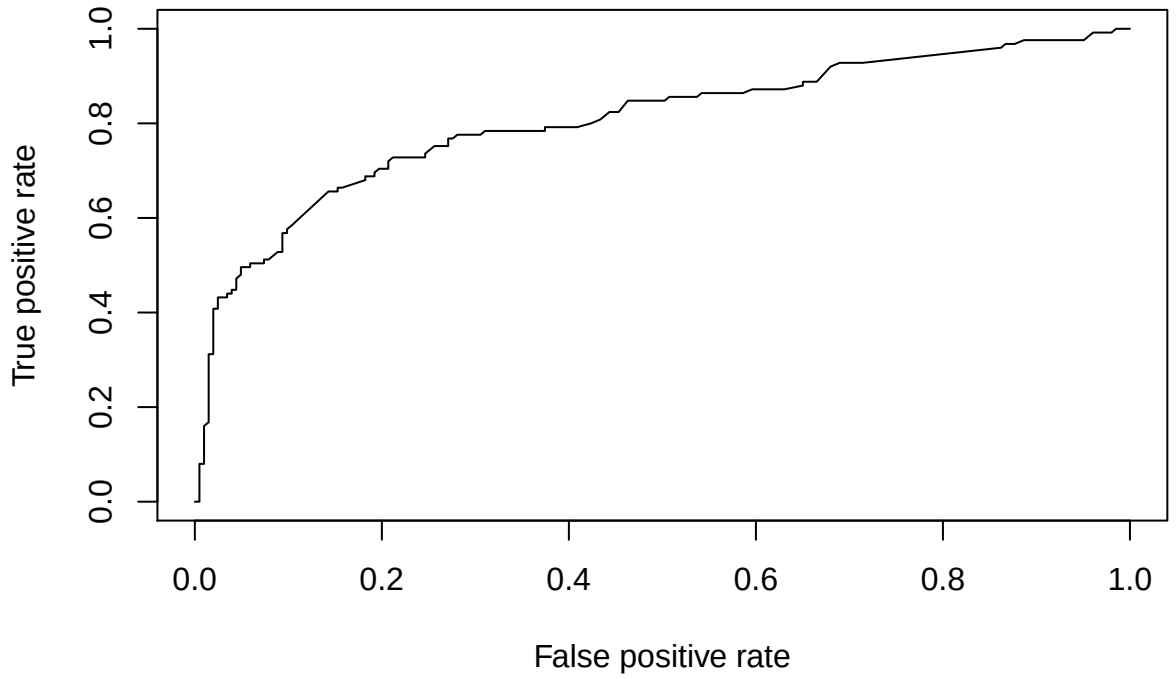
```
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
p <- predict(glm1, newdata=test, type="response")
pr <- prediction(p, test$survived)
# TPR = sensitivity, FPR=specificity
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.8056749
```