Machine Learning and Deep Learning Architectures

Logistic Regression

Logistic regression is a method for fitting a regression curve, *y = f(x)*, when y is a categorical variable. The typical use of this model is predicting *y* given a set of predictors *x*. The predictors can be continuous, categorical or a mix of both.

- Binomial Logistic Regression
- Multinomial Logistic Regression

R makes it very easy to fit a logistic regression model. The function to be called is glm() and the fitting process is not so different from the one used in linear regression.

We'll be working on the *Titanic dataset.*

The dataset (training) is a collection of data about some of the passengers (889 to be precise), and the goal of the competition is to predict the survival (either 1 if the passenger survived or 0 if they did not) based on some features such as the *class of service*, the *sex*, the *age* etc.

Data Cleaning Process

Make sure that the parameter na.strings is equal to c("") so that each missing value is coded as a NA. This will help us in the next steps.

```
training.data.raw <- read.csv('train.csv',header=T,na.strings=c(""))
```

Now we need to check for missing values and look how many unique values there are for each variable using the sapply() function which applies the function passed as argument to each column of the data frame.

```
sapply(training.data.raw,function(x) sum(is.na(x)))
```

```
PassengerId    Survived      Pclass        Name         Sex 
          0           0           0           0           0 
        Age       SibSp       Parch      Ticket        Fare 
        177           0           0           0           0 
      Cabin    Embarked 
        687           2 
```

```
sapply(training.data.raw, function(x) length(unique(x)))
```

```
PassengerId    Survived      Pclass        Name         Sex 
        891           2           3         891           2 
        Age       SibSp       Parch      Ticket        Fare 
         89           7           7         681         248 
      Cabin    Embarked 
        148           4 
```

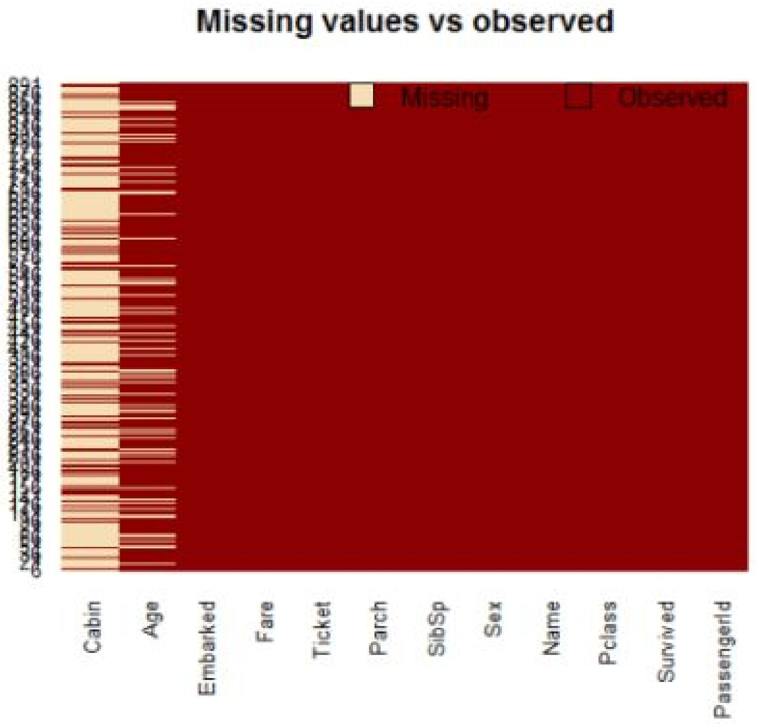
A visual take on the missing values might be helpful: the Amelia package has a special plotting function missmap() that will plot your dataset and highlight missing values:

```
library(Amelia)
missmap(training.data.raw, main = "Missing values vs observed")
```

The variable cabin has too many missing values, we will not use it. We will also drop PassengerId since it is only an index and Ticket.

Using the subset() function we subset the original dataset selecting the relevant columns only.

```
data <- subset(training.data.raw,select=c(2,3,5,6,7,8,10,12))
```



Taking care of the missing values

Now we need to account for the other missing values. R can easily deal with them when fitting a generalized linear model by setting a parameter inside the fitting function. However, personally I prefer to replace the NAs “by hand”, when is possible. There are different ways to do this, a typical approach is to replace the missing values with the average, the median or the mode of the existing one. I’ll be using the average.

```
data$Age[is.na(data$Age)] <- mean(data$Age,na.rm=T)
```

As far as categorical variables are concerned, using the read.table() or read.csv() by default will encode the categorical variables as factors. A factor is how R deals categorical variables.
We can check the encoding using the following lines of code

```
is.factor(data$Sex)
TRUE

is.factor(data$Embarked)
TRUE
```

For a better understanding of how R is going to deal with the categorical variables, we can use the contrasts() function. This function will show us how the variables have been dummyfied by R and how to interpret them in a model.

```
contrasts(data$Sex)
       male
female    0
male      1
```

```
contrasts(data$Embarked)
  Q S
C 0 0
Q 1 0
S 0 1
```

As for the missing values in Embarked, since there are only two, we will discard those two rows (we could also have replaced the missing values with the mode and keep the datapoints).

```
data <- data[!is.na(data$Embarked),]
rownames(data) <- NULL
```

Before proceeding to the fitting process, let me remind you how important is*cleaning and formatting of the data*. This preprocessing step often is crucial for obtaining a good fit of the model and better predictive ability.

Model fitting

We split the data into two chunks: training and testing set. The training set will be used to fit our model which we will be testing over the testing set.

```
train <- data[1:800,]
test <- data[801:889,]
```

Now, let’s fit the model. Be sure to specify the parameter family=binomialin the glm() function.

```
model <- glm(Survived ~.,family=binomial(link='logit'),data=train)
```

By using function summary() we obtain the results of our model:

```
summary(model)
```

Assessing the predictive ability of the model

In the steps above, we briefly evaluated the fitting of the model, now we would like to see how the model is doing when predicting *y* on a new set of data. By setting the parameter type='response', R will output probabilities in the form of P(y=1|X). Our decision boundary will be 0.5. If P(y=1|X) > 0.5 then y = 1 otherwise y=0. Note that for some applications different thresholds could be a better option.

```
fitted.results <- predict(model,newdata=subset(test,select=c(2,3,4,5,6,7,8)),type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)

misClasificError <- mean(fitted.results != test$Survived)
print(paste('Accuracy',1-misClasificError))
```

"Accuracy 0.842696629213483"

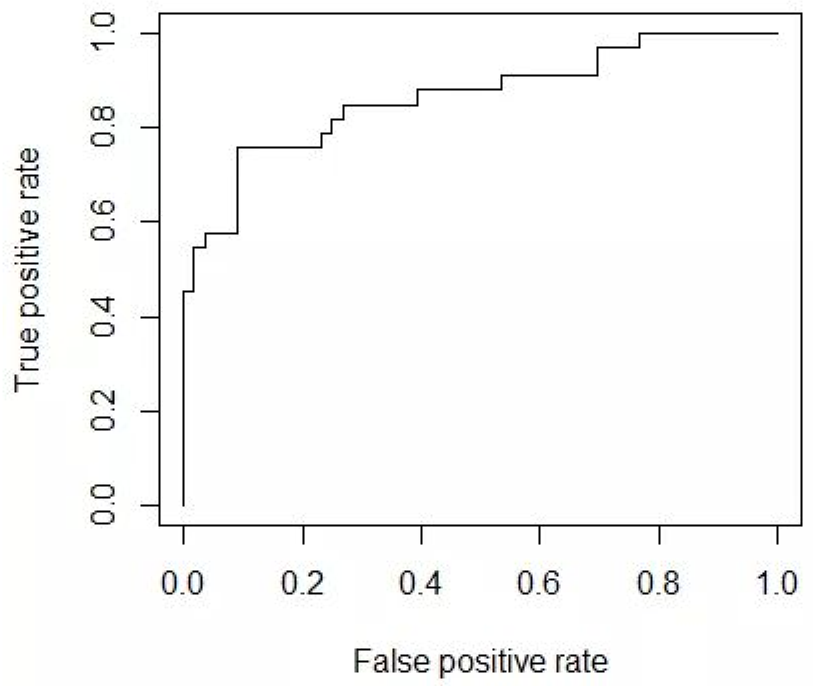
As a last step, we are going to plot the *ROC curve* and calculate the *AUC* (area under the curve) which are typical performance measurements for a binary classifier.
The ROC is a curve generated by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings while the AUC is the area under the ROC curve. As a rule of thumb, a model with good predictive ability should have an AUC closer to 1 (1 is ideal) than to 0.5.

```
library(ROCR)
p <- predict(model, newdata=subset(test,select=c(2,3,4,5,6,7,8)), type="response")
pr <- prediction(p, test$Survived)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)

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

0.8647186

And here is the ROC plot:


True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0