Practical 3: Classification

The aim of this lab exercise is to give you practical experience in classification. We will be using the titanic data set from Kaggle. You can find this on the week 3 page of Keats. You can run this tutorial in *R* or in *Python*.

About the titanic dataset

The titanic data set is a classic data set that is used for data science. This was used on *Kaggle* (www.kaggle.com) as a basis for a machine learning competition titled *Titanic: Machine Learning from Disaster*, where the challenge was to analyse what sorts of people were more likely to survive. The training data provided contained information about the characteristics of 891 passengers and whether they survived.

Attributes in the dataset

- Survival Survival (0 = No; 1 = Yes)
- pclass Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- name Name
- sex Sex
- age Age
- sibsp Number of Siblings/Spouses Aboard
- parch Number of Parents/Children Aboard
- ticket Ticket Number
- fare Passenger Fare
- cabin Cabin
- embarked Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

1 Classification

This data is the passenger list from the Titanic, and it includes a target attribute called *Survival*. The aim is to train models to predict whether a passenger survived or not.

1. Data exploration

   (a) Read in the data (`titanic.csv`).

(b) Perform some data exploration. Look at the summaries of the contents of each attribute, and create some plots in order to gain an understanding of the contents of the attributes. Look out for attributes that will not be useful as inputs to a classification model.

(c) Decide which attributes you need to exclude from the model inputs. For example: do some attributes include missing data, or information that is of no relevance to predicting survival? (Note: if you are using python then split your data frame into two: one containing your input attributes and the other containing the target attribute only)

(d) Split the data into training and test data, start off with using 60% of the data frame for training the model.

2. Train different classifiers and compare the results

   (a) Build a decision tree on the training data.

   (b) Test the model you have on the test data set. Compute the confusion matrix and the error rate.

   (c) Run logistic regression on the training data.

   (d) Test the performance of the logistic regression model on the test data using a confusion matrix and the error rate.

   (e) Plot the ROC curve for the decision tree and logistic regression on the same plot.

   (f) Which model has a lower error rate? which model would you chose to use and why?

   (g) Run a random forest model on this data, calculate its accuracy.

3. Refine the input data

   (a) Some of the attributes could not be used as they had missing data. Impute the missing values for the age of the passenger.

   (b) Create some additional transformations: for example the size of the family.

   (c) Re-train decision trees and logistic regression on this enhanced data set. How do the results compare?

4. Modify the model parameters

   (a) Try to change the default parameters of the different approaches and see how they affect the output. For example limit the depth of the decision tree or use a different splitting criterion.

5. Apply *oneR (1R)*. In order to assess how accurate a prediction can be made using only one attribute. Consider only two input attributes: *sex, pclass*

   (a) For each input attribute in turn create a classification rule by counting how often each value of the target attribute appears and selecting the most frequent class.

   (b) Calculate the error rate of each of inputs used.

   (c) Chose the input with the lowest error rate.

   (d) How does this compare to the error rates of the more sophisticated approaches?

2 Some useful reference code

In order to complete this task in R or Python the following functions will be useful:

2.1 Python

```python
#--
#split the data into test and train
#--
from sklearn.model_selection import train_test_split
df_analysis_train , df_analysis_test , df_target_train , df_target_test =
    train_test_split(df_analysis , df_target , test_size=0.4, random_state=0)
#--
#Decision Tree classifier
#--
from sklearn import tree
clf=tree.DecisionTreeClassifier()
#--
#In order to display the tree inline in jupyter
#--
import pydotplus
from IPython.display import Image
dot_data = tree.export_graphviz(clf , out_file=None,
                                filled=True, rounded=True, feature_names=(['list of
    names of the attributes']),
                                class_names=(['list of class levels']),
                                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
#--
#Logistic Regression
#--
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
#--
#To calculate the metrics for the ROC curve
#--
from sklearn.metrics import roc_curve , auc
import random
#--
# Random forests
#--
from sklearn import ensemble
clf = ensemble.RandomForestClassifier(n_estimators=100)
```

2.2 R

```r
#--
#Libraries of relevance
#--
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)
library(ROCR)
library(gmodels)
library(randomForest)
#--
#Formulas
#--
data.formula<-target_ind ~attribute1+attribute2
#--
# Trees
#--
data.rpart<-rpart(data.formula, data=training)
#--
#Logistic Regression
#--
data.logit<-glm(data.formula,family=binomial(logit), data=training)
#--
#Random Forest
#--
rf<-randomForest(data.formula, data=training, ntree=100)
```