R workshop #6: Classification methods

Nicola Romanò

Introduction

In the lectures we have introduced some key concepts in classification through machine learning, and have looked at some simple classification methods. In this workshop you will apply what you learned to real-life problems!

Learning objectives

After completing this workshop you will be able to:

- Divide your data into training and test set
- Create and interpret the output of a regression tree classifier
- Create and interpret the output of a Random Forest classifier
- Create ROC curves and evaluating classification performance

The Wisconsin Breast Cancer dataset

For this workshop we are going to use a simplified version of the Wisconsin Diagnostic Breast Cancer (WDBC) dataset. This was created in 1995 by scientists at University of Wisconsin[1].

The dataset consists of *cellular features* computed from images of a fine needle aspirate (FNA) of a breast mass, and describing the cell nuclei.

The dataset contains:

- Diagnosis (M = malignant, B = benign, our output value)
- radius (mean of distances from center to points on the perimeter of the cell nucleus)
- texture (standard deviation of gray-scale values)
- smoothness (local variation in radius lengths)

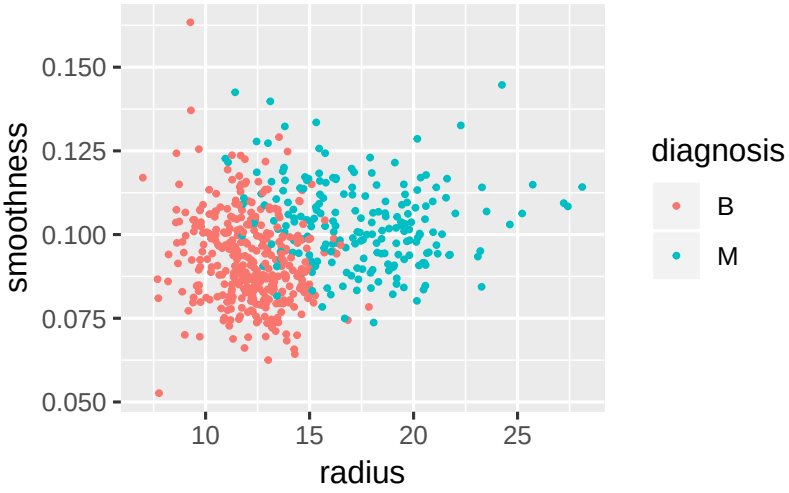
Data can be found in the `WDBC-workshop6.csv` file

We can start by inspecting the data. How many samples we have? How many were benign and how many malignant? Are there any missing values[2]?

You can also plot some of the variables to get a better feeling of the data; for example here is a plot of cell radius and smoothness vs the diagnosis

[1] UCI Machine Learning Repository: 1995. Center for Machine Learning and Intelligent Systems. Breast Cancer Wisconsin (diagnostic) dataset.

[2] Remember, you can use the `complete.cases` function to test that.



Creating training and test sets

In the lectures, you have learnt about using a training and test set (and cross-validation) to improve the accuracy of your classifiers. We can now proceed to create a training and a test set out of our data.

The easiest thing to do so is to use the `sample` function, to randomly sample a data frame. `sample` is one of many R functions that uses *randomly* generated data. To ensure reproducibility of these examples we will use the `set.seed` function [3].

[3] In most cases, computers do not actually generate random numbers. What they use is called a pseudo-random number generator, a deterministic algorithm that generates numbers that look, at all effects, random. The `set.seed` function can be used to initialise this generator, and using the same seed will guarantee that you will get the same result. If choose a different seed, you will get different results! If you omit the `set.seed` function you will have different results each time you run the script.

```
# You can use any number you like as a seed If you use
# '123' you will get the same results as me, otherwise
# results may be different
set.seed(123)

num.samples <- nrow(wdbc)
# We sample 1/3 of the values from 1 to num.samples and
# choose the corresponding lines as the test set
test.id <- sample(1:num.samples, size = 1/3 * num.samples,
    replace = FALSE)

# The training set consists of all the samples that are
# not in the test set
wdbc.test <- wdbc[test.id, ]
wdbc.train <- wdbc[-test.id, ]
```

We can now create our first classifier using the training set. We are going to use a logistic regression as a classifier.

```
# We use all possible classifier for our model
model.logistic <- glm(diagnosis ~ ., data = wdbc.train, family = binomial)
```

Evaluating the model

We can now create a confusion matrix. We can start with the training set

```
# Use the model to predict the response. This
# will be a value between 0 and 1
pr <- predict(model.logistic, wdbc.train, type = "response")

# Print the confusion matrix
tb <- table(Prediction = ifelse(pr < 0.5, "B",
    "M"), Real = wdbc.train$diagnosis)
tb
```

```
##           Real
## Prediction   B   M
##          B 224  14
##          M   9 133
```

Logistic regression does a fairly good job at classifying the tumours. We can calculate the false positive and false negative rate, as well as the accuracy of the model.

```
# Accuracy
(tb[1, 1] + tb[2, 2])/sum(tb)
```

```
## [1] 0.9394737
```

```
# FP rate
tb[1, 2]/sum(tb)
```

```
## [1] 0.03684211
```

```
# FN rate
tb[2, 1]/sum(tb)
```

```
## [1] 0.02368421
```

We can now do the same for the test set

```
pr <- predict(model.logistic, wdbc.test, type = "response")

tb <- table(ifelse(pr < 0.5, 0, 1), wdbc.test$diagnosis)

tb
```

```
##
##       B   M
##   0 114   6
##   1  10  59
```

```
# Accuracy
(tb[1, 1] + tb[2, 2])/sum(tb)
```

```
## [1] 0.9153439
```

```
# FP rate
tb[1, 2]/sum(tb)
```

```
## [1] 0.03174603
```

```
# FN rate
tb[2, 1]/sum(tb)
```

```
## [1] 0.05291005
```

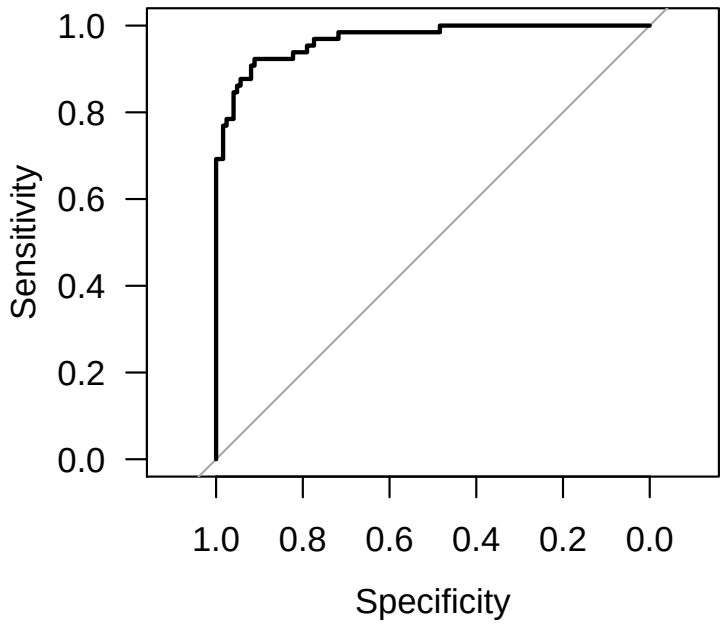
In this case, values are very similar, with >90% accuracy and a very small number of false positive or false negatives.

Let's now create a ROC curve, using the pROC package[4]

[4] As usual, you can install this using `install.packages("pROC")`.

```
library(pROC)

# Pass the observed and the predict values to
# the roc function...
ROC.curve <- roc(wdbc.test$diagnosis, predict(model.logistic,
    wdbc.test, "response"))
# ... and plot the curve!
plot(ROC.curve, las = 1)
```



As explained in the lectures, this corresponds to various levels of threshold. You can see that the prediction is very good in all cases, however we can try and optimise it even more, by using the `coords` function.

```
best.thr <- coords(ROC.curve, "best")
best.thr
```

```
##   threshold specificity sensitivity
##   0.4123572   0.9112903   0.9230769
```

Let's try and recalculate our confusion matrix with the new threshold of 0.4123572

```
pr <- predict(model.logistic, wdbc.test, type = "response")

# Print the confusion matrix
tb.1 <- table(Prediction = ifelse(pr < 0.5, "B",
    "M"), Real = wdbc.test$diagnosis)
tb.2 <- table(Prediction = ifelse(pr < best.thr["threshold"],
    "B", "M"), Real = wdbc.test$diagnosis)

tb.1
```

```
##           Real
## Prediction   B   M
##          B 114   6
##          M  10  59
```

```
tb.2
```

```
##           Real
## Prediction   B   M
##          B 113   5
##          M  11  60
```

In this particular case changing the threshold has not improved the overall accuracy (there are still 16 misclassified patients). The number of false positives has slightly increased while the number of false negative has decreased, due to the slight decreased threshold. The choice of which threshold to use should be evaluated on a case-by-case basis, and a good knowledge of the biological topic is often necessary to make a good choice.

Cross-validation

We can use the `train` function in the `caret`[5] package. The `trControl`

[5] short for Classification And REgression Training

argument to this function takes an object created through the `trainControl` function, where we can specify options, such as the fold for the cross-validation[6]. Below we specify a 10-fold cross-validation; you can experiment with other types of cross-validation and see whether that changes the accuracy of the model. Since this is a generic function, that can be used for any type of classification model, we also need to specify that we want a binomial GLM; the function will return a cross-validated model[7].

[6] Another parameter, `preProcess` allows you to apply some per-processing to the data. I will leave the experimenting on this to you; look at the help for the `train` function for more information!

[7] Note that we use the full dataset here, since `train` creates training/test sets internally during the cross-validation process.

```
library(caret)

train.control <- trainControl(method = "cv", number = 10)
model.logistic.cv <- train(diagnosis ~ ., data = wdbc,
    trControl = train.control, method = "glm",
    family = binomial)
print(model.logistic.cv)
```

```
## Generalized Linear Model
##
## 569 samples
##   3 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 512, 512, 513, 512, 511, 512, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.927906  0.8452918
```

The output of train is a cross validated GLM, with 92.8% accuracy[8], so slightly better than the previous one. Since the model already performed quite well, it is not so surprising that the accuracy is not that improved!

[8] This is the average of the accuracy over the 10 test sets. Note that you can see the accuracy for each fold by looking at `model.logistic.cv$resample$Accuracy`

Trees and random forests

We have discussed classification trees and random forests in detail during lecture 22.2

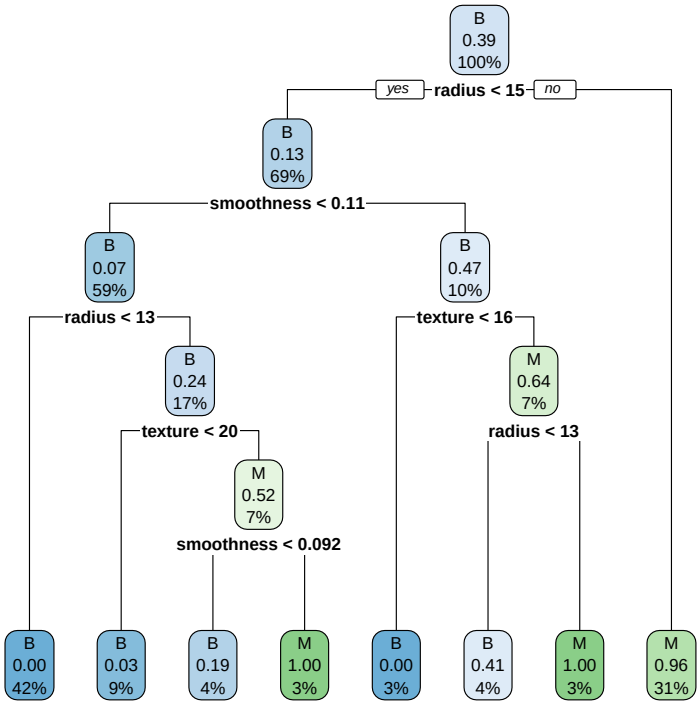
Let's create two new models using our dataset.

```
library(rpart)
library(rpart.plot)
head(wdbc.train)
```

```
##   diagnosis radius texture smoothness
## 2         M  15.08   25.74    0.10240
## 3         M  19.68   21.68    0.09797
## 4         M  14.22   23.12    0.10750
## 6         B  13.34   15.86    0.10780
## 7         M  25.73   17.46    0.11490
## 8         M  23.51   24.27    0.10690
```

```
model.tree <- rpart(diagnosis ~ ., wdbc.train)
```

```
rpart.plot(model.tree)
```

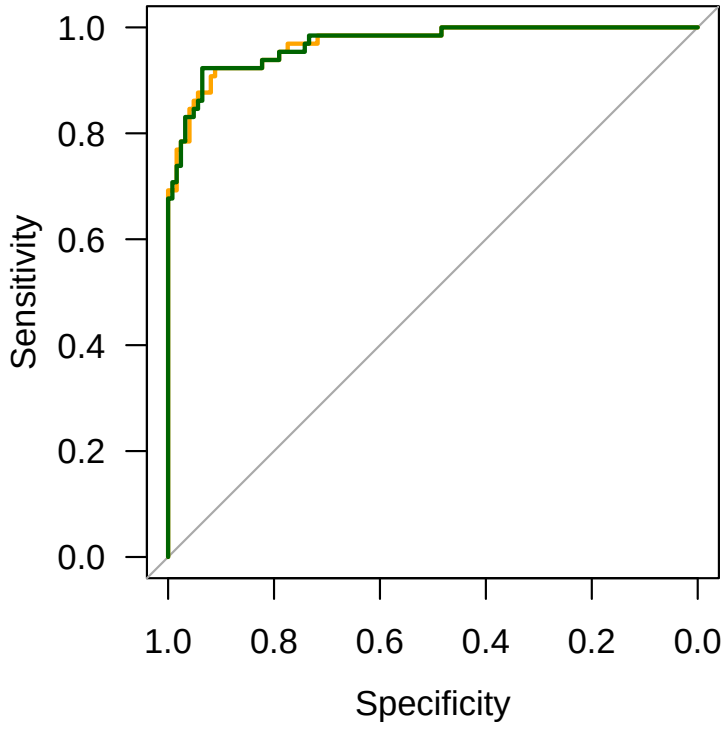


We can now plot the ROC curve for both models, and calculate the corresponding AUC using the `auc` function[9]. Again, unsurprisingly, the two models are essentially equivalent.

[9] Note that the syntax for predicting the results of the CV model is slightly different… I will let you figure out why

```
ROC.curve <- roc(wdbc.test$diagnosis, predict(model.logistic,
    wdbc.test, "response"))
ROC.curve.cv <- roc(wdbc.test$diagnosis, predict(model.logistic.cv,
    wdbc.test, "prob")[, 1])
# ... and plot the curve!
```

```
plot(ROC.curve, las = 1, col = "orange")
plot(ROC.curve.cv, las = 1, add = TRUE, col = "darkgreen")
```




```
auc(ROC.curve)
```

```
## Area under the curve: 0.9687
```

```
auc(ROC.curve.cv)
```

```
## Area under the curve: 0.9691
```