R packages

The functionality of R can be considerably extended by loading additional packages. These can be found on CRAN, the public repository of contributed packages (https://cran.r-project.org/).

Installing a package can be done from within R itself using function `install.packages()`: if you are asked to choose a CRAN mirror, pick any from the list, and when asked if you would like to use a personal library, answer "yes".

```
> # these are not required on the computer labs as packages are already installed
> if(!"pROC" %in% rownames(installed.packages())) {
+   install.packages("pROC")     # functions for ROC curves and AUC
+ }
>
> if(!"caret" %in% rownames(installed.packages())) {
+   install.packages("caret")    # functions for cross-validation
+ }
```

After issuing the command, the package will be automatically downloaded, compiled and installed. Once the installation has completed, to load the package into memory use function `library()`.

```
> library(pROC)                 # quotation marks are optional in this case
```

After this, all functions provided by the package and the corresponding documentation will become available in the R session.

Logistic regression

Let's consider a hip fracture example and model it in R. A case-control study includes 1327 women aged 50–81 with hip fractures as well as 3262 randomly selected women in the same age range. Among the cases, 40 women take hormone replacement therapy (HRT); among the controls, 239 women do. Does taking HRT affect the probability of having a hip fracture?

In this case we have summary statistics rather than individual-level data: however we can create a synthetic dataset which has the same characteristics and use that in the analysis.

```
> y <- c(rep(1, 1327), rep(0, 3262))    # cases, controls
> hrt <- c(rep(1, 40), rep(0, 1287),    # HRT, no HRT in cases
+          rep(1, 239), rep(0, 3023))   # HRT, no HRT in controls
```

To fit a logistic regression model, we use function `glm()`: we need to specify the logit link function, which can be done by using option `family="binomial"`. This is crucial, as otherwise R will use the identity link function (corresponding to `family="gaussian"`), which would fit a linear regression model.

```
> regr.hrt <- glm(y ~ hrt, family="binomial")
> summary(regr.hrt)
```

```
Call:
glm(formula = y ~ hrt, family = "binomial")

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.8422  -0.8422  -0.8422   1.5548   1.9710

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.85394    0.03328 -25.656  < 2e-16 ***
hrt         -0.93365    0.17405  -5.364 8.12e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5519.7  on 4588  degrees of freedom
Residual deviance: 5484.8  on 4587  degrees of freedom
AIC: 5488.8

Number of Fisher Scoring iterations: 4
```

In logistic regression the Wald test statistic is distributed according to a normal distribution because the standard deviation of the outcome doesn't need to be estimated (this is what the message "Dispersion parameter for binomial family taken to be 1" refers to). Therefore, the test statistic is now labelled `z value` (instead of `t value`): the p-value is obtained by comparing the test statistic to the quantiles of a standard normal distribution.

The estimated coefficients are the log-odds: the sign of the regression coefficient tells us that the use of HRT has a protective effect on hip fractures (it reduces their odds). However, we need to transform it to an odds ratio in order to better appreciate the magnitude of the effect.

```
> exp(coef(regr.hrt)[2])     # exponentiate to get the odds ratio
      hrt
0.3931169
```

If HRT had no effect, the odds ratio would be 1: given that we see an odds ratio of 0.393, we can claim that HRT reduces the odds of having a hip fracture by 60.7%.

To build a 95% confidence interval on the odds ratio, we can rely on a normal approximation remembering that 95% of the probability mass of a standard normal distribution is within -1.96 and 1.96:

$$e^{\hat{\beta}_i \pm 1.96 \times \text{SE}(\hat{\beta}_i)}$$

```
> beta <- coef(regr.hrt)[2]
> se.beta <- coef(summary(regr.hrt))[2, 2]
> round(exp(beta + 1.96 * se.beta * c(-1, 1)), 3)
[1] 0.279 0.553
```

Equivalently, it is possible to use the `confint()` function: results are not exactly the same, as the computation used above is based on asymptotic normality, while `confint()` uses a t distribution, which is usually better, especially for smaller sample sizes. Again, the intervals are in the log-odds scale, but we are interested in confidence intervals for the odds ratios, so we exponentiate them.

```
> or.ci <- round(exp(confint(regr.hrt)), 3)
> or.ci
            2.5 % 97.5 %
(Intercept) 0.399  0.454
hrt         0.276  0.546
```

The interpretation of a confidence interval is that we are 95% confident that the true value of the odds ratio is between 0.276 and 0.546. The confidence interval does not include 1 (the odds ratio corresponding to no effect), so we can reject the null hypothesis of no effect for HRT. We could have reached the same conclusion by looking at the p-value ($8.1 \times 10^{-8} < 0.05$).

The estimated intercept is -0.854. Transforming the log-odds through the logistic function, we can compute the baseline probability, that is the probability of an event when all other covariates (in our case, just taking HRT) are zero.

```
> baseline.odds <- exp(coef(regr.hrt)[1])
> baseline.prob <- baseline.odds / (1 + baseline.odds)    # logistic function
> round(baseline.prob, 3)
(Intercept)
      0.299
```

This tells us that a woman in the study who does not take HRT has 29.9% probability of experiencing a hip fracture. How does this probability change if a woman takes HRT?

```
> hrt.odds <- exp(coef(regr.hrt)[1] + coef(regr.hrt)[2])
> hrt.prob <- hrt.odds / (1 + hrt.odds)                   # logistic function
> round(hrt.prob, 3)
(Intercept)
      0.143
```

Note that these probabilities do not represent the absolute risk for a generic woman aged 50–81, as the proportion of cases in a case-control study generally does not reflect the proportion in the general population.

Fitted values

Because of the presence of the link function, we have two types of fitted values: the first correspond to the log-odds scale ($\hat{y} = X\hat{\beta}$) and are stored in the `linear.predictors` vector of the regression object.

```
> X <- model.matrix(regr.hrt)
> y.hat <- as.numeric(X %*% coef(regr.hrt))  # turn a 1-column matrix into a vector
> all(y.hat == regr.hrt$linear.predictors)
[1] TRUE
```

Alternatively, we may be talking about fitted values in the probability scale (the response scale), that is by transforming the linear predictors through the logistic function: these are stored in the `fitted.values` vector of the regression object.

```
> logistic <- function(z) exp(z) / (exp(z) + 1)
> prob.case <- logistic(regr.hrt$linear.predictors)
> all(prob.case == regr.hrt$fitted.values)
[1] TRUE
```

It is crucial not to mix the two scales, otherwise the interpretation will be affected.

Log-likelihood and derived measures

The log-likelihood of a fitted model can be extracted with function `logLik()`: this also outputs the number of degrees of freedom, that is the number of parameters estimated by the model.

```
> logLik(regr.hrt)
'log Lik.' -2742.4 (df=2)
```

The deviance of a model is minus twice the log-likelihood: this is what R labels "residual deviance" when using `summary()` over a logistic regression object.

```
> -2 * as.numeric(logLik(regr.hrt))             # same as regr.hrt$deviance
[1] 5484.799
```

The null deviance, instead, is minus twice the log-likelihood of a model which only uses the intercept term and no other predictor: a null model reports the same predicted probabilites for all observations, corresponding to the proportion of cases in the dataset.

```
> null.model <- glm(y ~ 1, family="binomial")   # only the intercept in the model
> head(null.model$fitted.values)                # same probabilities for all observations
        1         2         3         4         5         6
0.2891698 0.2891698 0.2891698 0.2891698 0.2891698 0.2891698
> sum(y) / length(y)                            # proportion of cases
[1] 0.2891698
> -2 * as.numeric(logLik(null.model))           # same as regr$null.deviance
[1] 5519.71
```

Models with better fit have lower deviance: in our case, adding one parameter to the null model decreases the deviance from 5519.71 to 5484.8.

The difference between two deviances has χ^2 distribution with degrees of freedom equal to the difference in parameters of the two models. Therefore we can use it as a test statistic and compute a p-value under the null hypothesis that the smaller model is better: if the p-value $< \alpha$ then we reject the null hypothesis and claim that the larger model is significantly better.

```
> pchisq(regr.hrt$null.deviance - regr.hrt$deviance, df=1, lower.tail=FALSE)
[1] 3.451601e-09
```

The AIC is another quantity derived from the log-likelihood which is used in model comparison, as it also takes into account model complexity: the deviance of the model is penalised by twice the number of parameters estimated in the model (stored in the `rank` field of the regression object).

```
> 2 * regr.hrt$rank - 2 * as.numeric(logLik(regr.hrt))
[1] 5488.799
> AIC(regr.hrt)                                 # same as regr.hrt$aic
[1] 5488.799
> AIC(null.model)
[1] 5521.71
```

As for the deviance, the lower the AIC, the better the model. However, there is no formal test that can be applied when comparing the AICs of two models. The BIC applies a stronger penalty for the use of additional predictors: even so, in this case the proposed model is still better than the null model.

```
> BIC(regr.hrt)
[1] 5501.662
> BIC(null.model)
[1] 5528.141
```

To compute the AUC for this model we will rely on package `pROC`: this provides function `roc()`, which by default reports the area under curve, but offers an option to plot the entire ROC (see `?roc` for full documentation).

```
> roc(y, regr.hrt$fitted.values)
```

```
Call:
roc.default(response = y, predictor = regr.hrt$fitted.values)

Data: regr.hrt$fitted.values in 3262 controls (y 0) < 1327 cases (y 1).
Area under the curve: 0.5216
```

An AUC of 0.52 is not good: this model is not very different from a random model at discriminating cases from controls. The purpose of this model was not to explain hip fractures based on HRT (or indeed to predict them!), but simply to understand if HRT has an effect on the odds of hip fractures.

To plot the ROC curve, the same command can be used with a few extra options documented in `?plot.roc`.

```
> roc(y, regr.hrt$fitted.values, plot=TRUE, legacy.axes=TRUE)
```

As expected from the AUC, the curve is very flat.

Making predictions

Let's consider a dataset which relates to subarachnoid hemorrhage (SAH), which is part of the `pROC` package, and let's start by fitting a simple model containing only age and gender on the entire dataset.

```
> data(aSAH)      # copy the dataset from the package into the workspace
> asah.all <- glm(outcome ~ age + gender, data=aSAH, family="binomial")
```

The `predict()` function allows us to use a fitted regression model (either linear or logistic regression) to make a prediction. The function takes two main arguments: a fitted prediction object (the output of `lm()` or `glm()`) and a dataframe (specified by option `newdata`) containing the covariate values for which we want to make a prediction of the outcome. By default, for a logistic regression model `predict()` will return the linear predictors, that is the log-odds. If we are interested in predicted probabilities, we need to specify `type="response"`.

```
> y.linpred <- predict(asah.all, newdata=aSAH)  # predict the data used for fitting
> y.pred <- predict(asah.all, newdata=aSAH, type="response")
```

Predicting the same data that was used for fitting is not very interesting: on one hand we are overestimating the predictive performance of the model, and on the other hand this produces what is already available in the `linear.predictors` and `fitted.values` vectors of the regression object.

What makes more sense is predicting the outcome for new data. This needs to be a dataframe containing columns that correspond to the predictors used in the model. For example, to predict the response for a 30 years old male patient, we could create a dataframe to contain these data and ask the model make a prediction for it.

```
> data.30M <- data.frame(age=30, gender="Male")
> predict(asah.all, newdata=data.30M, type="response")
        1
0.2990627
```

In general, predictions will be made on dataframes coming from different cohorts, or for subsets of the original dataset that were withdrawn, as in a cross-validation setting.

Data partitioning and cross-validation

Some convenient functions related to predictive models are implemented in the `caret` package.

```
> library(caret)
```

This allows us to call function `createDataPartition()` which creates a single training/test split. This function (like similar ones in the same package) requires us to pass the vector of outcomes, so that the partition will contain a balanced proportion of cases and controls in the training and test sets. Before creating the partition, we must set the seed of the random number generation, so that results will be reproducible.

```
> set.seed(1)
> train.idx <- createDataPartition(aSAH$outcome, p=0.7)$Resample1      # 70-30 split
```

This creates a list of indices corresponding to the observations in the training set: we can use this to fit the model only on this subset of data. The `subset` option of `lm()` and `glm()` allows to control the observations that are used to fit the model coefficients without making us create new dataframes to store the subsets.

```
> asah.train <- glm(outcome ~ age + gender, data=aSAH, subset=train.idx,
+                   family="binomial")
```

Having fitted the same model on just a part of the data, we can see that the coefficients are different from those we obtained when we used all data: in this case they appear to be smaller than before. Also p-values are less extreme, but this is mainly due to the reduced size of the dataset and the consequent increase in the standard errors. Note that we cannot use the deviance or the AIC to compare models fitted on different datasets (or, as in this case, on datasets of different sizes): these quantities depend on the number of observations used in fitting, so a model fitted to a larger dataset will in general have higher deviance and AIC, but this is not necessarily an indication of bad quality of fit.

What really matters is how well we predict the outcome for the withdrawn observations.

```
> # option 1
> pred.prob <- predict(asah.train, newdata=aSAH, type="response")[-train.idx]
>
> # option 2
> pred.prob <- predict(asah.train, newdata=aSAH[-train.idx, ], type="response")
```

Let's plot the ROC curve for the prediction of the test data and compute the corresponding test AUC.

```
> roc(outcome ~ pred.prob, data=aSAH[-train.idx, ], plot=TRUE)
```

The AUC for this model (0.736) is quite good. Unfortunately, given that we created only one partition, it's hard to tell if a similar performance would be obtained on a different random training/test split.

In this sense, cross-validation is a more informative approach, as it allows us make a prediction for all observations of the dataset. The folds can be generated with function `createFolds()`. By default the function returns a list of indices corresponding to the test set of each fold.

```
> set.seed(1)
> num.folds <- 10
> folds <- createFolds(aSAH$outcome, k=num.folds)  # get indices of the test sets
```

Note that the `folds` object is a list: this datatype allows to store different classes of objects (say vectors and dataframes) within the same variable. To access an element of a list, use the `[[ ]]` operator.

```
> class(folds)
[1] "list"
> length(folds)
[1] 10
> folds[[1]]                                          # this stores a vector
 [1]  20  30  37  41  43  47  53  55  91 100 106
```

Now it's a matter of fitting a model in each of the training sets and predict the outcome for observations in the corresponding test sets. We need to store each fitted model so that they can be used afterwards for prediction or inspection: by assigning the output from `glm()` to `res.folds[[fold]]` we are effectively adding an element to a list of results.

```
> regr.cv <- NULL
> for (fold in 1:num.folds) {
+   train.idx <- setdiff(1:nrow(aSAH), folds[[fold]])
+   regr.cv[[fold]] <- glm(outcome ~ age + gender, data=aSAH, subset=train.idx,
+                          family="binomial")
+ }
```

We will need another loop to produce the predicted responses for each fold. For convenience we create a dataframe to store the observed outcome and what we predict from the model.

```
> pred.cv <- NULL
> auc.cv <- numeric(num.folds)
> for (fold in 1:num.folds) {
+   test.idx <- folds[[fold]]
+   pred.cv[[fold]] <- data.frame(obs=aSAH$outcome[test.idx],
+                                 pred=predict(regr.cv[[fold]], newdata=aSAH,
+                                              type="response")[test.idx])
+   auc.cv[fold] <- roc(obs ~ pred, data=pred.cv[[fold]])$auc
+ }
```

After cross-validation, we can report the expected performance of the model on withdrawn data. This is slightly smaller than what we obtained when using a single partition, but we can attach more confidence to this estimate.

```
> round(mean(auc.cv), 3)
[1] 0.704
```