Class 3: Model checking and performance

Andrew Parnell, School of Mathematics and Statistics, University College Dublin

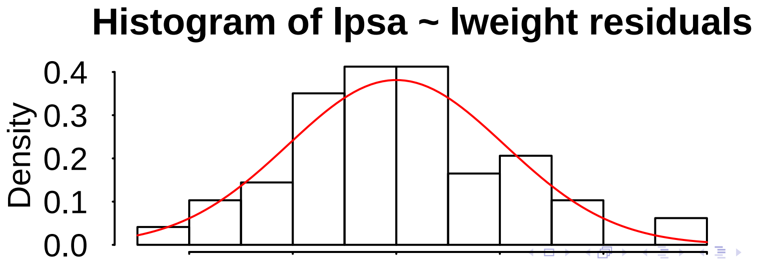
Learning outcomes

- Be able to read and understand regression diagnostic plots
- Be able to compare statistical models using information criteria and cross-validation
- Understand the different types of classification metrics
- Understand and interpret ROC curves and AUC values

Regression diagnostics

- We saw in the previous class that the vertical distances between the predictions and the observations - the *residuals* are assumed to be normally distributed. How can we check this?
- One obvious way is to produce a histogram of the residuals and see if they look 'normal':

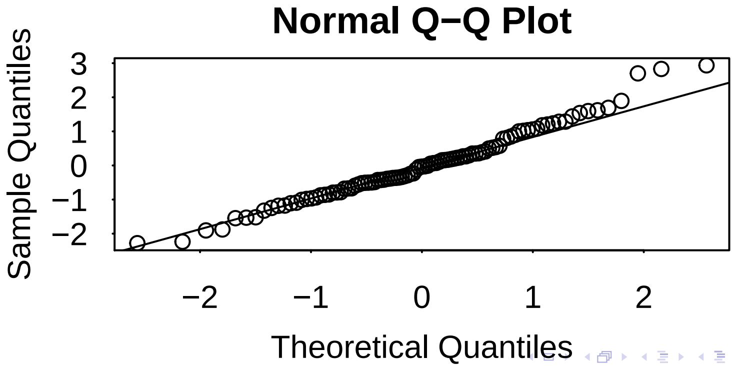
```
model_1 = lm(formula = lpsa ~ lweight, data = prostate)
hist(model_1$residuals, freq=FALSE, main = 'Histogram of lp
curve(dnorm(x, sd = 1.046), col='red', add = TRUE)
```



More on residual plots

- Histograms tend to under-weight the importance of extreme observations
- Better is a QQ-plot:

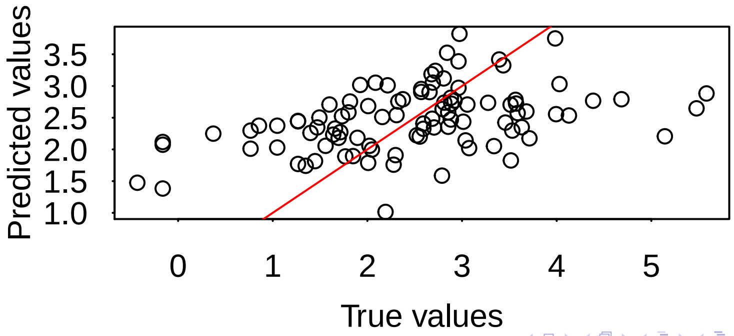
```
qqnorm(model_1$residuals)
qqline(model_1$residuals)
```



Plotting the predictions

A really good model will have the predictions looking a lot like the true values:

```
plot(prostate$lpsa, model_1$fitted.values,
     ylab = 'Predicted values', xlab = 'True values')
abline(a=0, b=1, col='red')
```



Issues with over-fitting

- As you add more terms to the regression model, the fit will generally get better, higher R-squared, lower residual standard error, closer predicted values to true values
- Regularisation and shrinkage can assist with this problem
- A useful way to judge this problem is to leave out some of the data, create the predictive model, and then create predictions of the left out data

Cross validation

A very useful recipe for evaluating model fit. For k-fold CV:

1. Break the data up into k chunks or *folds*
2. Leave fold k out and fit the model to the remaining folds
3. Predict values for the missing fold
4. Repeat k times, once for each fold
5. Plot/summarise the left out predictions with the true data

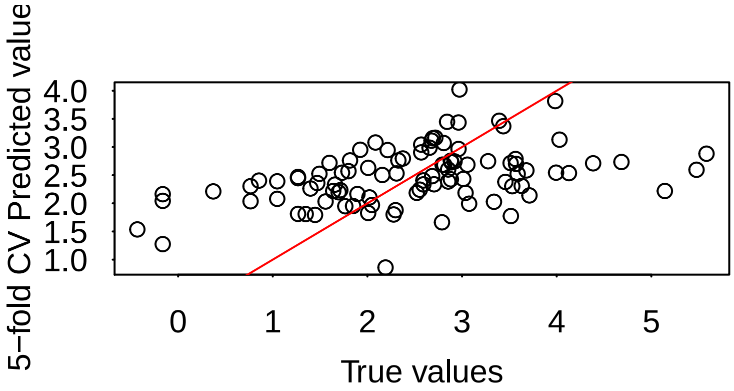
If the model is really performing then the out of sample predictions should look like the true data

Cross validation in R

```
n_folds = 5
n = nrow(prostate)
folds = sample(1:5, n, replace = TRUE)
cv_preds = rep(NA, length = n)
for(i in 1:n_folds) {
  curr_model = lm(lpsa ~ lweight,
                  data = subset(prostate, folds!=i))
  cv_preds[folds==i] = predict(curr_model,
                               newdata = subset(prostate, f
}
```

Cross-validation output

```
plot(prostate$lpsa, cv_preds,
     ylab = '5-fold CV Predicted values',
     xlab = 'True values')
abline(a=0, b=1, col='red')
```



Not a great model!

More on cross-validation

- Cross-validation can be used to choose between models. For example, if you're not sure whether to include a particular explanatory variable you could run 5-fold CV for each one and see which performs better. The advantage is that the performance will not necessarily improve as you put in more explanatory variables
- There are different versions of CV. We could run 10-fold CV, which would take longer but be more like the full model as it uses more data points
- Some people run leave-out-out CV (LOO-CV) which leaves only one data point out at a time. This can be very slow though
- CV will work with both regression and classification approaches

Model comparison

- Another way to choose between models (e.g. with different explanatory variables) is to use an *information criterion*
- This is a measure of the model fit penalised by its complexity
- A model with lots of explanatory variables is very complex and so will be given a high penalty
- However, if the new explanatory variables explain the variation in the response well then it will be worth adding in to the model
- Perhaps the most commonly used is the Akaike Information Criterion (AIC) which R gives you as part of `lm`'s output
- The usual practice is to fit a range of models and pick the one with the *smallest* AIC value. This will usually give you a better model than if you pick on p-values

Model comparison

The cheat way of doing this is via `stepAIC`:

```r
library(MASS)
stepAIC(model_4) # Recall model 4 had everything in it
```

```
...
Step:  AIC=10.66
lpsa ~ lweight

          Df Sum of Sq    RSS    AIC
<none>                 103.90 10.665
- lweight  1    24.018 127.92 28.838
```

Some notes about information criteria

- There are lots of different versions: AIC, AICc, BIC, DIC, WAIC, . . .
- It's very hard to decide whether a drop in AIC is 'statistically significant' so sometimes we are left with two or three models to choose between
- Some information criteria (AIC) aim to estimate the LOO-CV performance, but only require one fit of the model
- Information criteria, like cross-validation, will work for both regression and classification models

Classification diagnostics

Classification diagnostics

- Recall that for a classification model, such as the logistic regression model we applied to the South African Heart Rate data, we are predicting a probability value
- We might decide to assume that all observations with a probability value greater than 0.5 get classified as CHD, whilst all those with probability values less than 0.5 get classified as non-CHD
- If all the probabilities are particularly low or high it might be that we have don't have anyone classified to one of the groups
- We might thus decide on a different cut-off other than 0.5 which might improve the predictions

The misclassification table

For a probability cut-off of 0.5 we have:

```
table(SA$chd, round(model_1$fitted.values),
      dnn=c('True','Predicted'))
```

```
##      Predicted
## True   0   1
##    0 243  59
##    1  89  71
```

- The top left figure here is the number of *true negatives*, i.e. those who do not have CHD and are predicted to not have CHD
- The bottom right figure is the number of *true positives*
- The top right figure is the number of *false negatives*. They are predicted to be positive but they are not
- The bottom left figure is the number of *false positive*. They are predicted to be negative but are not

More on misclassification tables

- From the misclassification table we can calculate a huge number of different performance metrics (more later)
- Ideally we want the values on the diagonal to be large and the off-diagonals to be small
- We can change the cut-off with

```
cut_off = 0.3
table(SA$chd, as.integer(model_1$fitted.values>cut_off),
      dnn=c('True','Predicted'))
```

```
##      Predicted
## True    0   1
##     0 162 140
##     1  36 124
```

Now many more observations have been put into the right column

Sensitivity and specificity

- The two most common statistics to calculate from the misclassification table are:
  - The *sensitivity*, or the true positive rate, calculated as the number of true positives divided by the number of positives
  - The *specificity*, or the true negative rate, calculated as the number of true negatives divided by the number of negatives
- We want both of these to be high:

```
cut_off = 0.3
tab = table(SA$chd,
            as.integer(model_1$fitted.values>cut_off),
            dnn=c('True','Predicted'))
cat('Sensitivity = ',tab[2,2]/(tab[2,1] + tab[2,2]),
    'Specificity = ',tab[1,1]/(tab[1,1] + tab[1,2]))
```

```
## Sensitivity =  0.775 Specificity =  0.5364238
```

Choosing a probability cut-off: Youden's index

- If you have to choose a single probability cut-off, a popular choice is Youden's index, calculated as:

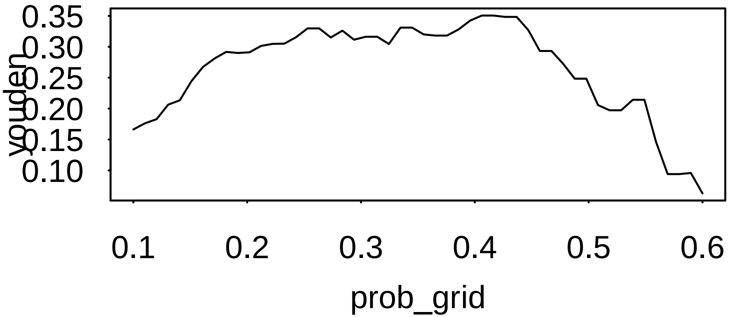
$$\textit{sensitivity} + \textit{specificity} - 1$$

- We can find the probability value which maximises Youden's index:

```
prob_grid = seq(0.1, 0.6, length = 50)
youden = rep(NA, length = 50)
for(i in 1:50) {
  tab = table(SA$chd,
              as.integer(model_1$fitted.values>prob_grid[i
  sens = tab[2,2]/(tab[2,1] + tab[2,2])
  spec = tab[1,1]/(tab[1,1] + tab[1,2])
  youden[i] = sens + spec - 1
}
```

Youden's index value

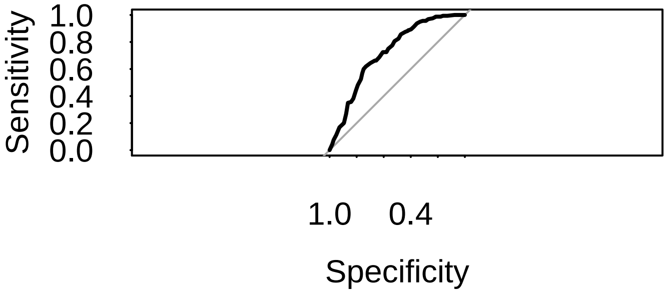
```
plot(prob_grid, youden, type = 'l')
```



The ROC curve

- It is common to plot the sensitivity and specificity values for a full range of cut-offs. This is known as the *Receiver Operator Characteristic* (ROC) curve:

```r
library(pROC)
roc(SA$chd, model_1$fitted.values, plot=TRUE)
```



AUC

- The ROC curve shows, for each probability cut-off the value of the sensitivity and specificity
- A good classification model should have the curve well away from the diagonal. This means that for every probability cut off we are good at identifying the positive and negative cases
- As a general summary, the area under the ROC curve (the *AUC*) is often calculated too:

```
auc(SA$chd, model_1$fitted.values)
```

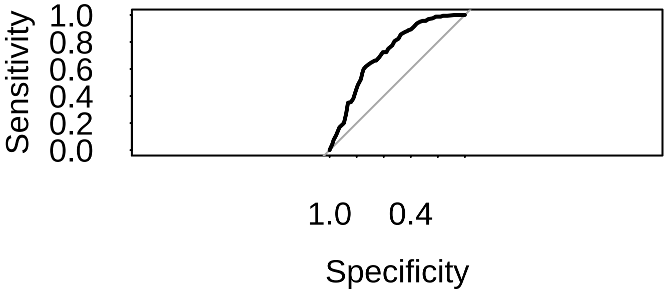
```
## Area under the curve: 0.7225
```

- The AUC measures the extend to which the classification model beats a random classifier
- A high value is desirable. A common (unjustified) cut-off is 0.7

Calibration

- The ROC curve just looks at the sensitivity and specificity for different cut-offs. It doesn't actually matter what the probability cut-offs are - you get the same plot if you divide them by 1000!

```
roc(SA$chd, model_1$fitted.values/1000, plot=TRUE)
```



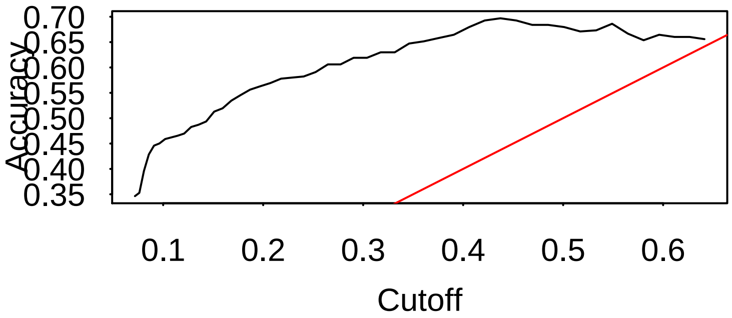
Calibration 2

- A model is mis-calibrated if the probabilities do not match the true probabilities in the data set. For example, if we say someone has an 80% chance of getting CHD, then under repeated sampling of similar individuals 80% should have CHD
- We can create a calibration plot with the R package `ROCR`:

```r
library(ROCR)
pred = prediction(model_1$fitted.values, SA$chd)
acc = performance(pred, measure = 'acc')
```

Calibration plot

```r
plot(acc)
abline(a=0, b=1, col='red')
```



Summary

- We can perform residual analysis on a regression model
- We can perform cross validation or compare models using AIC for both classification and regression models
- We have learnt about classification metrics: ROC curves, AUC values, and calibration