MATH5743M: Statistical Learning - Lecture 4

Dr Seppo Virtanen, s.virtanen@leeds.ac.uk

Semester 2: 14 Feb 2022

Recap

Last week we looked at linear regression models in one dimension. We started by

- defining the likelihood, the probability of observing a given set of outputs,
- conditioned on known input values and a particular choice of model parameters: the intercept and gradient values.

We used optimisation techniques to infer the maximum-likelihood estimates for the model parameters.

We then learnt about using the **glm** command in R to efficiently perform model fits and predictions.

Outline

Multiple linear regression, where there may be more than one input value used to predict the output.

We will also see how this can be used to model non-linear patterns in the data, by using transformations of the inputs known as *basis functions* as new inputs.

We will conclude with overfitting and model selection.

Multiple linear model

In multiple linear regression the output, y, is assumed to be determined by a weighted, linear sum of several inputs, $x_1, x_2, \ldots, x_n$, and a normally-distributed random residual, ϵ

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_n x_{in} + \epsilon_i,\ \epsilon_i \sim \mathcal{N}(0, \sigma) \qquad (1)$$

The likelihood

Assuming that we have some collected data to analyse, in the form of some inputs and outputs, we can write down the log-likelihood for the model parameters (the coefficient values) in a manner analogous to the one-dimensional case. I'll use both the summation form and the matrix form:

$$\begin{aligned}\log \mathcal{L}(\beta_0, \beta_1, \ldots, \beta_n, \sigma) &= \log P(\mathrm{Data} \mid \beta_0, \beta_1, \ldots, \beta_n \sigma) \\ &= \sum_{i=1}^{N} \log \mathcal{N}(y_i; \beta_0 + \sum_{j=1}^{n} \beta_j x_{ij}, \sigma^2)\end{aligned} \tag{2}$$

Multiple linear regression in R

Create some simulated data where an output y is a linear function of three different inputs, x_1, x_2, x_3, with some random normally distributed residuals. The values of the inputs are drawn randomly between zero and one

```r
x1 = runif(100)
x2 = runif(100)
x3 = runif(100)
y = 1*x1 + 2*x2 + 3*x3 + rnorm(100, 0, 0.1)
```

glm in R

```
my_multiple_data = data.frame(y, x1, x2,x3)
my_multiple_model = glm(y ~ x1 + x2 + x3, data=my_multiple_
summary(my_multiple_model)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + x3, data = my_multiple_data
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.210606  -0.058321  -0.006042   0.068937   0.248191
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.03443    0.03598  -0.957    0.341
## x1           1.03981    0.03854  26.982   <2e-16 ***
## x2           2.02769    0.03666  55.309   <2e-16 ***
## x3           3.02346    0.03511  86.113   <2e-16 ***
```

Confidence intervals

Last week we learnt how to construct a 95% confidence interval for a regression parameter value in the one-dimensional case. To do so for multiple dimensions we follow a very similar procedure. However, we must take to obtain the right number of degrees of freedom. Recall that for one-dimensional regression with n observations we had $n - 2$ degrees of freedom, because that data had been used to estimate two parameters before the standard error.

In the case of multiple linear regression we have fewer degrees of freedom, one fewer for each input. So if we have a model with 3 inputs as above, we will have $n - 4$ degrees of freedom (the extra one is for the intercept).

Confidence intervals

As an example, lets construct the 95% confidence interval for the x1 coefficient. We can extract the summary table of coefficients and standard errors like this:

```
summary_table = summary(my_multiple_model)$coefficients
print(summary_table)
```

```
##                Estimate Std. Error    t value     Pr(>|t
## (Intercept) -0.03442773 0.03597593 -0.9569656 3.409884e-
## x1           1.03981225 0.03853705 26.9821417 1.317015e-
## x2           2.02768640 0.03666097 55.3091385 1.296069e-
## x3           3.02345947 0.03511054 86.1125812 1.063875e-
```

Confidence intervals

We use R to find the appropriate critical t value via the **qt** function (which gives quantiles of the t distribution). Here we had 100 observations, so $n - 4 = 96$ is the number of degree of freedom. The quantile we need is 0.975 (i.e. 2.5% lies beyond this)

```r
t_critical = qt(0.975, 96)
```

Confidence intervals

The estimate of the coefficient and the standard error are extracted from the table:

```
estimate = summary_table[2, 1]
sterr = summary_table[2,2]
```

We now construct the interval as: estimate $\pm$ $t_{2.5\%,96} \times$ standard error

```
interval_min = estimate - t_critical*sterr
interval_max = estimate + t_critical*sterr
print(paste(c('Min: ', interval_min), collapse=""))
```

```
## [1] "Min: 0.963316806794049"
```

```
print(paste(c('Max: ', interval_max), collapse=""))
```

```
## [1] "Max: 1.11630770237561"
```

Basis functions for non-linear regression

One reason we learn about and use linear regression is that the process of fitting this model is extremely well understood, and the results are readily interpretable. We also do so because we can use the framework of linear regression to build non-linear models.

Basis functions: These are transformations of the original inputs through some non-linear functions, to create new inputs for use in a (multiple) linear regression problem.

Basis functions for non-linear regression

Radioactive decay: Model the amount of radiation, $r(t)$, that we record as a function of time, t, like this:

$$r(t) = r_0 \exp(-t/\tau) + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma). \tag{3}$$

Polynomial regression:

$$y = \sum_{i=0}^{p} \beta_i x^i + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma) \tag{4}$$

Difficulties and dangers

- Highly correlated inputs: Difficult to interpret the coefficients.
- Overfitting: model has found all the useful information contained in some data and starts to fit not to the general patterns but to the noise that is always present in a real data set.

We will learn more about overfitting and how to avoid it in Week 5 when we discuss Model Selection.