https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/06/lecture-06.pdf

#

1 Recapitulation

We introduced the method of maximum likelihood for simple linear regression in the notes for two lectures ago. Let's review.

We start with the statistical model, which is the Gaussian-noise simple linear regression model, defined as follows:

1. The distribution of X is arbitrary (and perhaps X is even non-random).
2. If $X = x$, then $Y = \beta_0 + \beta_1 x + \epsilon$, for some constants ("coefficients", "parameters") β_0 and β_1, and some random noise variable ϵ.
3. $\epsilon \sim N(0, \sigma^2)$, and is independent of X.
4. ϵ is independent across observations.

A consequence of these assumptions is that the response variable Y is independent across observations, conditional on the predictor X, i.e., Y_1 and Y_2 are independent given X_1 and X_2 (Exercise 1).

As you'll recall, this is a special case of the simple linear regression model: the first two assumptions are the same, but we are now assuming much more about the noise variable ϵ: it's not just mean zero with constant variance, but it has a particular distribution (Gaussian), and everything we said was uncorrelated before we now strengthen to independence[1].

Because of these stronger assumptions, the model tells us the conditional pdf of Y for each x, $p(y|X = x; \beta_0, \beta_1, \sigma^2)$. (This notation separates the random variables from the parameters.) Given any data set $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$, we can now write down the probability density, under the model, of seeing that data:

$$\prod_{i=1}^{n} p(y_i|x_i; \beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}}$$

[1] See the notes for lecture 1 for a reminder, with an explicit example, of how uncorrelated random variables can nonetheless be strongly statistically dependent.

In multiplying together the probabilities like this, we are using the independence of the Y_i.

When we see the data, we do not *known* the true parameters, but any guess at them, say (b_0, b_1, s^2), gives us a probability density:

$$\prod_{i=1}^{n} p(y_i|x_i; b_0, b_1, s^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi s^2}} e^{-\frac{(y_i-(b_0+b_1 x_i))^2}{2s^2}}$$

This is the **likelihood**, a function of the parameter values. It's just as informative, and much more convenient, to work with the **log-likelihood**,

$$\begin{aligned} L(b_0, b_1, s^2) &= \log \prod_{i=1}^{n} p(y_i|x_i; b_0, b_1, s^2) && (1)\\ &= \sum_{i=1}^{n} \log p(y_i|x_i; b_0, b_1, s^2) && (2)\\ &= -\frac{n}{2}\log 2\pi - n\log s - \frac{1}{2s^2}\sum_{i=1}^{n}(y_i - (b_0 + b_1 x_i))^2 && (3) \end{aligned}$$

In the **method of maximum likelihood**, we p[ick the parameter values which maximize the likelihood, or, equivalently, maximize the log-likelihood. After some calculus (see notes for lecture 5), this gives us the following estimators:

$$\begin{aligned} \hat{\beta}_1 &= \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} = \frac{c_{XY}}{s_X^2} && (4)\\ \hat{\beta}_0 &= \overline{y} - \hat{beta}_1 \overline{x} && (5)\\ \hat{\sigma}^2 &= \frac{1}{n}\sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2 && (6) \end{aligned}$$

$$(y_i - (b_0 + b_1 x_i)$$

Note that is y - yhat = residual where yhat is the expected value of Y given X

https://www.r-bloggers.com/2013/08/fitting-a-model-by-maximum-likelihood/

> Maximum-Likelihood Estimation (MLE) is a statistical technique for estimating model parameters. It basically sets out to answer the question: what model parameters are most likely to characterise a given set of data? First you need to select a model for the data. And the model must have one or more (unknown) parameters. As the name implies, MLE proceeds to maximise a likelihood function, which in turn maximises the agreement between the model and the data.
>
> Most illustrative examples of MLE aim to derive the parameters for a probability density function (PDF) of a particular distribution. In this case the likelihood function is obtained by considering the PDF not as a function of the sample variable, but as a function of distribution's parameters. For each data point one then has a function of the distribution's parameters. The joint likelihood of the full data set is the product of these functions. This product is generally very small indeed, so the likelihood function is normally replaced by a log-likelihood function. Maximising either the likelihood or log-likelihood function yields the same results, but the latter is just a little more tractable!

In [1]:

```
#Let's illustrate with a simple example: fitting a normal distribution. First we
generate some data.

set.seed(1001)

N <- 100

x <- rnorm(N, mean = 3, sd = 2)
```

In [3]:

```
x <- runif(N)
y <- 5 * x + 3 + rnorm(N)
lm(formula = y ~ x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      3.228        4.583
```

Pushing on to the MLE for the linear model parameters. First we need a likelihood function. The model is not a PDF, so we can't proceed in precisely the same way that we did with the normal distribution. However, if you fit a linear model then you want the residuals to be normally distributed. So the likelihood function fits a normal distribution to the residuals.

In [4]:

```
LL <- function(beta0, beta1, mu, sigma) {
    # Find residuals
    #
    R = y - x * beta1 - beta0
    #
    # Calculate the likelihood for the residuals (with mu and sigma as parameters)
    #
    R = suppressWarnings(dnorm(R, mu, sigma))
    #
    # Sum the log likelihoods for all of the data points
    #
    -sum(log(R))
}
```

In [5]:

```
LL <- function(beta0, beta1, mu, sigma) {
    R = y - x * beta1 - beta0
    #
    R = suppressWarnings(dnorm(R, mu, sigma, log = TRUE))
    #
    -sum(R)
}
```

In [13]:

```
Call:
mle(minuslogl = LL, start = list(beta0 = 3, beta1 = 1, mu = 0,
    sigma = 1))

Coefficients:
    beta0     beta1        mu     sigma
3.1138968 4.5834955 0.1138968 1.0458974
```

In [12]:

```
fit <- mle(LL, start = list(beta0 = 4, beta1 = 2, mu = 0, sigma=1))
fit

Error in solve.default(oout$hessian): system is computationally singular: reciprocal
condition number = 4.03531e-21
Traceback:

1. mle(LL, start = list(beta0 = 4, beta1 = 2, mu = 0, sigma = 1))
```

```
2. solve(oout$hessian)
3. solve.default(oout$hessian)
```

In [14]:

```
#But if we choose values that are reasonably close then we get a decent outcome.
library(stats4)
fit <- mle(LL, start = list(beta0 = 3, beta1 = 1, mu = 0, sigma=1))
fit
```

```
Call:
mle(minuslogl = LL, start = list(beta0 = 3, beta1 = 1, mu = 0,
    sigma = 1))

Coefficients:
    beta0     beta1        mu     sigma
3.1138968 4.5834955 0.1138968 1.0458974
```

In [15]:

```
summary(fit)
```

```
Warning message in sqrt(diag(object@vcov)):
"NaNs produced"

Maximum likelihood estimation

Call:
mle(minuslogl = LL, start = list(beta0 = 3, beta1 = 1, mu = 0,
    sigma = 1))

Coefficients:
       Estimate Std. Error
beta0 3.1138968        NaN
beta1 4.5834955 0.37941862
mu    0.1138968        NaN
sigma 1.0458974 0.07395567

-2 log L: 292.763
```

It stands to reason that we actually want to have the zero mean for the residuals. We can apply this constraint by specifying mu as a fixed parameter. Another option would be to simply replace mu with 0 in the call to dnorm(), but the alternative is just a little more flexible.

In [17]:

```
fit <- mle(LL, start = list(beta0 = 2, beta1 = 1.5, sigma=1), fixed = list(mu = 0),
             nobs = length(y))
summary(fit)
```

```
Maximum likelihood estimation

Call:
mle(minuslogl = LL, start = list(beta0 = 2, beta1 = 1.5, sigma = 1),
    fixed = list(mu = 0), nobs = length(y))

Coefficients:
      Estimate Std. Error
beta0 3.227888 0.20466245
beta1 4.583190 0.37938985
sigma 1.045818 0.07394165

-2 log L: 292.763
```

In [18]:

```
AIC(fit)
```

298.763034458593

Returning now to the errors mentioned above. Both of the cases where the call to mle() failed resulted from problems with inverting the Hessian Matrix. With the implementation of mle() in the stats4 package there is really no way to get around this problem apart from having a good initial guess. In some situations though, this is just not feasible. There are, however, alternative implementations of MLE which circumvent this problem. The bbmle package has mle2() which offers essentially the same functionality but includes the option of not inverting the Hessian Matrix.

In [19]:

```
library(bbmle)

 fit <- mle2(LL, start = list(beta0 = 3, beta1 = 1, mu = 0, sigma = 1))

Warning message:
"package 'bbmle' was built under R version 4.1.3"
```

In [20]:

```
fit

Call:
mle2(minuslogl = LL, start = list(beta0 = 3, beta1 = 1, mu = 0,
    sigma = 1))

Coefficients:
    beta0     beta1        mu     sigma
3.1138968 4.5834955 0.1138968 1.0458974

Log-likelihood: -146.38
```

In [ ]: