MATH5743M: Statistical Learning - Lecture 2

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

Semester 2: 31 Jan 2022

Outline

Learn the best values for unknown parameters/variables of a statistical model.

Minimise/maximise a function

- numeric optimisation
- hill-climbing algorithms
- analytic optimisation

Maximum likelihood estimation

Firing range of a cannon

Optimise the firing range of a cannon, by manipulating the angle at which it fires.

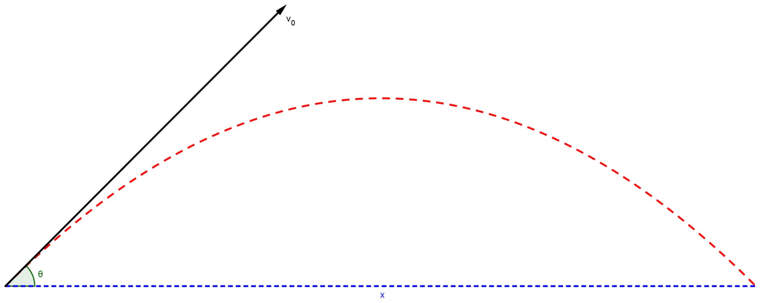


Figure 1: The path of a projectile fired from the ground depends on the initial velocity and angle.

Firing range of a cannon

The equation for the distance x travelled by a cannon ball fired at angle θ and at speed v (assuming no air resistance) is:

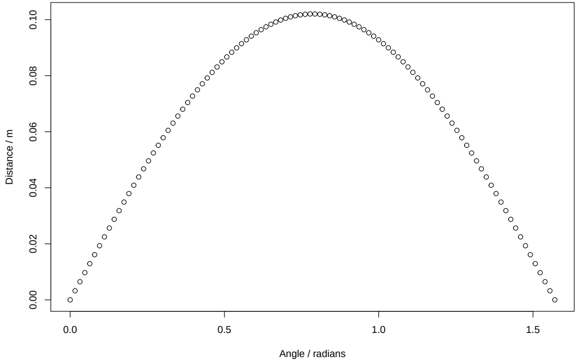
$$x = \frac{v^2}{g} sin(2\theta). \tag{1}$$

Estimate the best value for θ by trying a range of values.

Numeric optimisation

We'll set $g = 9.8ms^{-2}$ and use an initial velocity of $v = 1ms^{-1}$. We'll test angles from 0 radians (straight along the ground) to $\pi/2$ radians (straight into the air):

```r
theta = seq(0, pi/2, length.out=100)
x = (1/9.8)*sin(2*theta)
plot(theta, x, xlab="Angle / radians", ylab="Distance / m"
```



Numeric optimisation

We can use R to interrogate the values of x to find the best angle among the 100 that we tried

```r
idx = which.max(x)
print(theta[idx])
```

```
## [1] 0.7774648
```

This asks for the index of the largest value in x, and then prints the corresponding value of θ, which we see is close to $\pi/4$. We could get a better estimate simply by increasing the number of different values of θ that we tried.

Pitfalls?

What if many variables?

Prefer high degree of accuracy?

Hill-climbing algorithms

Gradient ascent/descent

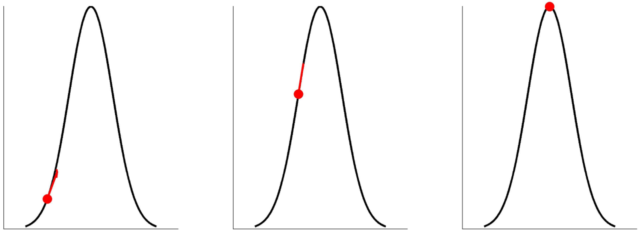


Figure 2: A gradient-ascent or 'hill climbing' algorithm improves an initial guess by following the gradient of the function, until it reaches a peak.

Use optim in R

```
#define an objective function to MINIMISE (notice the minus
obj_fn <- function(param){-(1/9.8)*sin(2*param)}
#Start randomly between 0 and pi/2
start_guess = (pi/2)*runif(1)
#Run the optimiser and store the result
opt_result = optim(par=start_guess, fn=obj_fn)
```

```
## Warning in optim(par = start_guess, fn = obj_fn): one-di
## use "Brent" or optimize() directly
```

```
#Print the optimised parameter value
print(paste("Best parameter value is: ", opt_result$par,
            collapse=""))
```

```
## [1] "Best parameter value is:  0.785408692937117"
```

optim in R

Incorporates many of the best known and widely used optimisation algorithms

Make the optimisation more efficient if you can provide a function that gives the gradient of the objective function.

Problem: Existence of local maxima of the function being optimised.

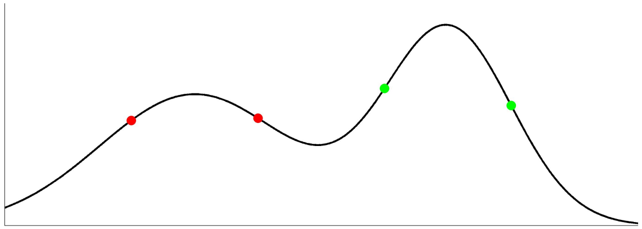


Figure 3: Local maxima can lead to optimsation answers that depend on the initial guess. Here the red initial guesses result in finding the left hand peak, while the green guesses result in finding the right hand peak.

Analytic optimisation

Obtain the exact solution using basic calculus when analytic (closed-form) solution exists.

Key: *Gradient* of the function at its maximum is zero.

$$\begin{aligned}
\left.\frac{dx}{d\theta}\right|_{\hat{\theta}} &= 0 = \frac{2v^2}{g}\cos(2\hat{\theta}) \\
&\Rightarrow \cos(2\hat{\theta}) = 0 \\
&\Rightarrow 2\hat{\theta} = \cos^{-1}(0) = \pi/2 \\
&\Rightarrow \hat{\theta} = \pi/4 = 0.7853982\ldots
\end{aligned} \tag{2}$$

Maximum-likelihood estimation

How we use statistical models to learn from data. Alternatively, fit statistical models to data.

Recall:

$$P(\text{OUTPUT} = y \mid \text{INPUT} = x, \theta) = f(y, x, \theta). \tag{3}$$

The function $f()$ specifies the probability that the output value(s) will be y, given input value(s) of x, and the model parameters θ.

In the *learning* phase of statistical modelling, we first select an appropriate function $f()$, then we try to identify the best possible set of parameters θ so that we can make accurate predictions of the OUTPUT from the INPUT.

Maximum-likelihood estimation

Maximum-likelihood estimation is a technique for choosing the best value of θ. It works by considering which value of θ would have made the data we have already seen most predictable. In other words, we look for the value of θ that maximises a quantity known as the likelihood $\mathcal{L}(\theta)$, which is the probability of all observed outputs, conditioned on all observed inputs and the value of θ

$$\mathcal{L}(\theta) = P(\textsc{output} = \{y_1, \ldots, y_n\} \mid \textsc{input} = \{x_1, \ldots, x_n\}, \theta) \quad (4)$$

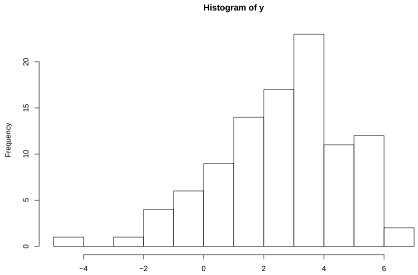
Example

The normal or Gaussian distribution has a probability density function defined as:

$$p(X = x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \tag{5}$$

Generate the data:

```
y = rnorm(100, m = 3, sd = 2)
hist(y)
```



Example

The likelihood function for the model parameters μ and σ^2 is the probability of observing (or generating) these data from a normal distribution with those parameters. Since the samples are generated independently, this is the product of the probability for each data point

$$\mathcal{L}(\mu, \sigma) = \prod_{i=1}^{100} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y_i - \mu)^2}{2\sigma^2}\right) \tag{6}$$

Use the *log-likelihood* instead; the maximum value of $f(x)$ and $\log(f(x))$ occur at the same value of x. This turns the product over probabilities into a sum over log-probabilities:

$$\log \mathcal{L}(\mu, \sigma^2) = -100 \log(\sqrt{2\pi}\sigma) - \sum_{i=1}^{100} \frac{(y_i - \mu)^2}{2\sigma^2} \tag{7}$$

Example

```
lognorm<-function(param,Y){-100*log(sqrt(2*pi)*param[2])-
    sum((Y-param[1])^2/(2*param[2]^2))}
neg_LL <- function(param) -lognorm(param, Y=y)
start_guess = c(0, 1) #first guess of mean of 0, stand.dev
mle_param = optim(par=start_guess, fn=neg_LL)
print(paste("MLE mean = ", mle_param$par[1], collapse=""))
```

```
## [1] "MLE mean =  2.63212171482323"
```

```
print(paste("MLE variance = ", mle_param$par[2]^2, collapse
```

```
## [1] "MLE variance =  4.40883703469466"
```