MATH5743M: Statistical Learning

Dr Seppo Virtanen, School of Mathematics, University of Leeds

Semester 2: 2022

Week 10: Further Topics

This week we will be investigating in greater depth a few topics that we have skimmed over before, and also taking a wider look at the world of statistical learning.

Standard errors - where do they come from?

In weeks 3-4, we were looking at linear regression models. We looked at how the maximum-likelihood model parameters could be derived in detail, before learning how to use the **glm** command to fit the model efficiently. This command also gives us the standard errors on the parameters, which are a measure of our *uncertainty* about the true parameter values, and used to calculate confidence intervals. But we didn't look at exactly where these standard errors come from. Now is our chance!

Consider the simple one-dimensional linear regression model:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2)$$

Lets focus on the parameter β_1. From week 3 we know that the MLE for this parameter is:

$$\hat{\beta_1} = \frac{\text{COV}(x, y)}{\text{VAR}(x)}$$

This gives us the best single estimate of β_1, but we'd also like to know how much this estimate would vary if we were to 'repeat the experiment', i.e. if we were to select data points with the same input values, but which would have different values of y because of the random variation of the residuals, ϵ.

Since our model tells us how the value of y_i is determined, we can substitute this back into our estimate:

$$\hat{\beta_1} = \frac{\text{COV}(x, \beta_0 + \beta_1 x + \epsilon)}{\text{VAR}(x)}$$

Note here that the parameter values on the right hand side are the *true* (and unknown) values. Expanding this out gives:

$$\begin{aligned} \hat{\beta_1} &= \frac{\text{COV}(x, \beta_0) + \text{COV}(x, \beta_1 x) + \text{COV}(x, \epsilon)}{\text{VAR}(x)} \\ &= \frac{0 + \beta_1 \text{VAR}(x) + \text{COV}(x, \epsilon)}{\text{VAR}(x)} \\ &= \beta_1 + \frac{\text{COV}(x, \epsilon)}{\text{VAR}(x)} \end{aligned} \tag{1}$$

Therefore:

$$\hat{\beta_1} - \beta_1 = \frac{\text{COV}(x, \epsilon)}{\text{VAR}(x)}$$

So the estimate will differ from the true value by some amount that depends on the (sample) covariance of the residuals and the input values, as well as the variance of the input values. What are the expectation and variance of this difference? We can treat $\text{VAR}(x)$ as a constant because the input values are fixed, only the values of ϵ are random, so we need to focus on $\text{COV}(x, \epsilon)$. First, the expectation: since ϵ and x are independent, this must be zero. That means that the expected value of $\hat{\beta_1}$ is simply β_1, i.e. the estimate is *unbiased* – on average we will get the true value. What about the variance? First we expand out $\text{COV}(x, \epsilon)$):

$$\begin{aligned}
\text{COV}(x, \epsilon) &= \frac{n}{n-1}\left[\frac{1}{n}\sum_{i=1}^{n} x_i\epsilon_i - \bar{x}\bar{\epsilon}\right] \\
&= \frac{n}{n-1}\left[\frac{1}{n}\sum_{i=1}^{n} x_i\epsilon_i - \bar{x}\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\right] \\
&= \frac{1}{n-1}\left[\sum_{i=1}^{n}(x_i - \bar{x})\epsilon_i\right]
\end{aligned} \tag{2}$$

Now we find the (non-sample) variance of this quantity by applying the rules for the variance of a sum of independent variables: $\text{VAR}(aX + bY) = a^2\text{VAR}(X) + b^2\text{VAR}(Y)$. Therefore:

$$\begin{aligned}
\text{VAR}(\text{COV}(x, \epsilon)) &= \text{VAR}(\frac{1}{n-1}\left[\sum_{i=1}^{n}(x_i - \bar{x})\epsilon_i\right]) \\
&= \frac{1}{(n-1)^2}\text{VAR}(\epsilon)\sum_{i=1}^{n}(x_i - \bar{x})^2 \\
&= \frac{1}{(n-1)}\text{VAR}(\epsilon)\text{VAR}(x) \\
&= \frac{1}{(n-1)}\sigma^2\text{VAR}(x)
\end{aligned} \tag{3}$$

Finally we can use this result to calculate the variance of the error $\hat{\beta_1} - \beta_1$:

$$\begin{aligned}
\text{VAR}(\hat{\beta_1} - \beta_1) &= \text{VAR}(\frac{\text{COV}(x, \epsilon)}{\text{VAR}(x)}) \\
&= \frac{\text{VAR}(\text{COV}(x, \epsilon)}{\text{VAR}(x)^2} \\
&= \frac{1}{(n-1)}\frac{\sigma^2}{\text{VAR}(x)} \\
&= \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}
\end{aligned} \tag{4}$$

Phew! So finally we have derived this expression for the variance of our estimate error. What does this tell us? It tells us how this error will move around as we repeat our experiment multiple times. The square root

of this variance gives us the standard deviation of the error, or what is generally known as the *standard error* – this is what we have been using to calculate our confidence intervals.

Why do we need to use the t-distribution when calculating the confidence interval? Note that the standard error, $\frac{\sigma}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}}$ contains the *true* value of σ. In practice we don't know what this is, so we have to use the estimated value instead, just as when calculating confidence intervals in your earlier modules. Using the estimated value introduces more uncertainty, which is accounted for by using the t-distribution.

Roll-your-own models

It is important to realise that all the modelling principles we have studied in this course are highly general. Many students in statistics get the impression that they are supposed to learn several different completely distinct algorithms, which must be memorised like magic potions! Instead, if you understand the real principles behind modelling, it is quite straightforward to learn new models, or even to create some of your own. Below I'm going to consider a situation which is not perfectly addressed by either linear regression or logistic regression, and create a model based on the specific problem, before applying the general principles we have learnt.

Consider a group of anglers (people who fish with a rod and line). Each angler, $i \in 1, \dots n$ goes down to the same lake and fishes for a different amount of time, t_i. During this time he or she catches a number of fish, $f_i \in 1, 2, \dots, \infty$. How could we model this? The first thing to notice is that the amount of fish caught is a numeric value - this is a regression problem. But unlike the linear regression problems we've looked at before, this numeric value will typically only take relatively small integers - catching a fish is a rare event. You have probably seen before that the standard distribution for modelling the integer number of rare events is the *Poisson distribution*, defined as below for the probability that the number of events, K is equal to some integer k:

$$P(K = k) = \frac{\exp(-\lambda)\lambda^k}{k!} \tag{5}$$

where λ is the mean or expected number of events.

How can we use this to model our anglers' success, and predict the number of fish caught by another angler? Well if we assume that the expected number of fish caught is proportional to the time spent at the lake (i.e. there is a constant probability per unit time of catching a fish), this means that each individual will have a different value for this expectation, λ_i, which is proportional to their time at the lake:

$$\lambda_i = \beta t_i \tag{6}$$

where we have assumed that β is the same for everyone. In other words, the anglers are all equally good, but those who spend longer at the lake will *on average* catch more fish.

With this assumption we can write down a formula for the probability of f_i, the number of fish caught by angler i, using the Poisson distribution formula above and substituting our expression for λ_i:

$$\begin{aligned} P(f_i \mid t_i) &= \frac{\exp(-\lambda_i)\lambda_i^{f_i}}{f_i!} \\ &= \frac{\exp(-\beta t_i)(\beta t_i)^{f_i}}{f_i!} \end{aligned} \tag{7}$$

Note that this model now has one free parameter, β, that closely resembles the coefficients seen in both linear and logistic regression models – it tells us how strongly time at the lake influences the number of fish caught. So how can we estimate this parameter from a data set? By maximising the log-likelihood of course! So first we need to define this log-likelihood. Assuming that each angler's catch is independent of the others (no crossed lines or fights over fish!), then recall that the log-likelihood is simply the sum of the log-probabilities for each data point:

$$\begin{aligned}\ln \mathcal{L} &= \sum_{i=1}^{n} \ln P(f_i \mid t_i) \\ &= \sum_{i=1}^{n} \ln \left(\frac{\exp(-\beta t_i)(\beta t_i)^{f_i}}{f_i!} \right) \\ &= \sum_{i=1}^{n} (-\beta t_i + f_i \ln(\beta t_i) - \ln(f_i!))\end{aligned} \tag{8}$$

Now that log-likelihood has some scary looking terms like $\log(f_i!)$, but remember that any terms like this that don't include β won't affect the maximum-likelihood value for β, since they don't change when β is varied. To get the MLE for β we need to maximise the log-likelihood, which we do the standard way, by differentiating and setting to zero at the MLE estimate $\hat{\beta}$

$$\begin{aligned}0 &= \frac{d \ln \mathcal{L}}{d\beta}|_{\hat{\beta}} \\ &= \sum_{i=1}^{n} \left(-t_i + \frac{f_i}{\hat{\beta}} \right) \\ &= \sum_{i=1}^{n} -t_i + \sum_{i=1}^{n} \frac{f_i}{\hat{\beta}}\end{aligned}$$

which can be rearranged to give:

$$\hat{\beta} = \frac{\sum_{i=1}^{n} f_i}{\sum_{i=1}^{n} t_i} \tag{9}$$

We see that the maximum-likelihood estimator for β is simply the total number of fish caught by all anglers, divided by the total amount of time spent by all anglers. This makes sense - all the anglers are equally good, so if they spend five hours at the lake in total, that should be the same as if any one of them had spent five hours there alone.

Assume we see the following data set for times and fish caught:

Angler	1	2	3	4	5	6	7	8	9	10
Time spent (minutes)	45	60	70	40	50	100	80	35	110	50
Fish caught	1	2	1	2	0	3	3	1	2	0

From this data set we can see that the total time spent is 640 minutes, and the total number of fish caught is 15. So our MLE for β is:

$$\hat{\beta} = \frac{\sum_{i=1}^{n} f_i}{\sum_{i=1}^{n} t_i} = \frac{15}{640} = 0.0234 \tag{10}$$

Which tells us that on average an angler can expect to catch one fish every 40 minutes (roughly). Using this estimate we can make predictions for new anglers, or for test data (for doing model comparison). For example, if another angler is heading down to the lake for 60 minutes, and wants to know the probability that he will catch at least one fish, we can calculate this based on $\hat{\beta}$

$$\begin{aligned} P(f \geq 1 \mid \hat{\beta}, t = 60) &= 1 - P(f = 0 \mid \hat{\beta}, t = 60) \\ &= 1 - \frac{\exp(-\hat{\beta} \times 60)(\hat{\beta} \times 60)^0}{0!} \\ &= 1 - \exp(-0.0234 \times 60) \\ &= 1 - 0.245 \\ &= 0.755 \end{aligned} \tag{11}$$

So our new angler has approximately a 75% chance to catch at least one fish. Similar calculations can be performed to predict the probability of different catch numbers. By combining this with cross-validation and data splitting you can also perform model selection for different models. For example, you might think that another factor influences the catch, such as your experience level (in years of fishing experience, e_i). In that case you might try an alternative model such as:

$$\lambda_i = \beta_1 t_i + \beta_2 e_i,$$

and use cross-validation to test whether this makes better predictions than using time alone.

Summary

The recap, when we are modelling, we need to consider the following:

1. What is the form of the output data? Classes or numbers? Integers or real numbers? Small or large values?
2. Based on the type of data, what is the best *distribution* to model it? Binomial (for binary classification)? Normal (for real values or large integers)? Poisson (for small integers, rare events)?
3. Using this distribution, how can we define a model that incorporates the input variables as predictors of the distribution parameters?
4. What is the *likelihood*? How can we *maximise it*.
5. How can we compare this model to others?

Some useful distributions already covered in the lecture material for modelling outputs include:

- Gaussian or normal distribution (linear regression)
- Bernoulli (logistic regression)
- Poisson (see the example above)

Other relevant distributions include:

- Beta distribution for $x \in (0, 1)$, $x \sim \text{Beta}(\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$, where $\Gamma(\cdot)$ denotes the Gamma function. Here, x takes continuous values within an interval with lower and upper values for the bounds. The parameters of the distribution include continuously-valued shape and rate parameters $\alpha > 0$ and $\beta > 0$, respectively. The expected value is given by $E[x] = \frac{\alpha}{\alpha+\beta}$

- Multinomial distribution for count/discrete data (non-negative integers) over more than two categories: The distribution generalises Bernoulli/Binomial distribution that is ssuitable for two categories. For $\mathbf{x} \sim \text{Multinomial}(n, \mathbf{p}) = \frac{n!}{\prod_{k=1}^{K} x_k!} \prod_{k=1}^{K} p_k^{x_k}$, where $n = \sum_{k=1}^{K} x_k$, K denotes the number of categories, x_k is the count for the kth category out of K categories and p_k is the probability for the kth category. Here, $x_l \in \{0, 1, 2, \ldots, n\}$, for $k \in \{1, \ldots, K\}$. The probabilities are positive and sum to one over the categories: $p_k \geq 0$, for $k = 1, \ldots, K$, and $\sum_{k=1}^{K} p_k = 1$. Similarly to a special case of the Binomial distribution with $n = 1$ we get the Bernoulli distribution, for the the multinomial distribution with $n = 1$ we get the categorical distribution that may be used for classification tasks where we have more than two classes. Note here that outputs and parameters are bold-faced as they denote K-dimensional vectors. The expected value is given by $E[x_k] = np_k$.

To construct GLM-type models for different types of outputs and tasks (regression vs classification) we need to connect the outputs y to inputs $x_1, x_2, \ldots, x_P$, that co-occur, using the expected value of the probabity distribution for the outputs. We have already seen:

- linear regression: $E[y] = \beta_0 + \sum_{p=1}^{P} \beta_p x_p$
- logistic regression: $E[y] = \phi(\beta_0 + \sum_{p=1}^{P} \beta_p x_p)$, where $\phi(x) = \frac{1}{1+\exp(-x)}$ denotes the logistic function
- Poisson regression: $E[y] = \exp(\beta_0 + \sum_{p=1}^{P} \beta_p x_p)$

Note that for linear regression we have used the identity function, $I(x) = x$. The modelling steps are similar to, though a bit more complicated for, the multinomial and Beta distributions, where we need to express $E[y]$ suitably by taking some function of the linear combination of the inputs to take into account restrictions or constraints of the probabiity distribution of the outputs.

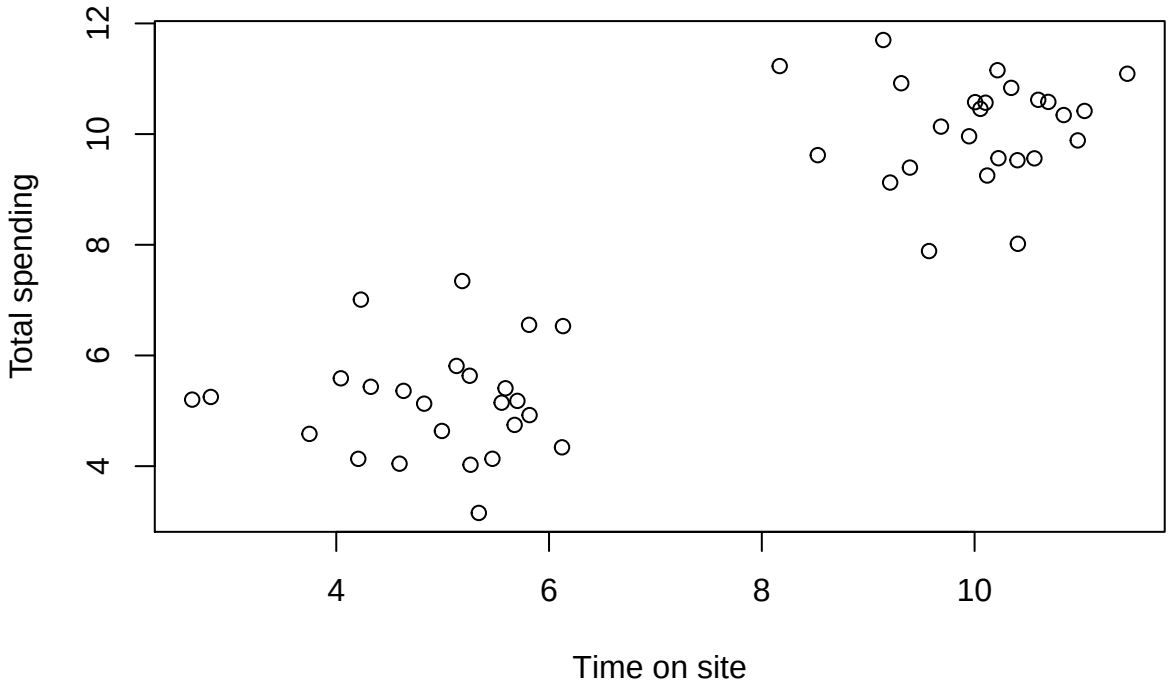
After choosing the probability distribution for the outputs, parameterising the model with a suitable function for GLM-type models, we need to write down the joint likelihood function over the observed data points and maximise the log likelihood with respect to the model parameters. This corresponds to maximum likelihood estimation that can be carried out analytically to obtain a closed form solution or by numerical optimisation when closed form expressions do not exist.

A look back at unsupervised learning

Throughout this course we have spent most of our time looking at *supervised learning* tasks – problems where we have output values to learn from. Week 1 contrasted this with unsupervised learning, where instead of learning from outputs, we look at the data and try to find intrinsic structures in there. To illustrate the difference, it is worth considering a real-world example of unsupervised learning.

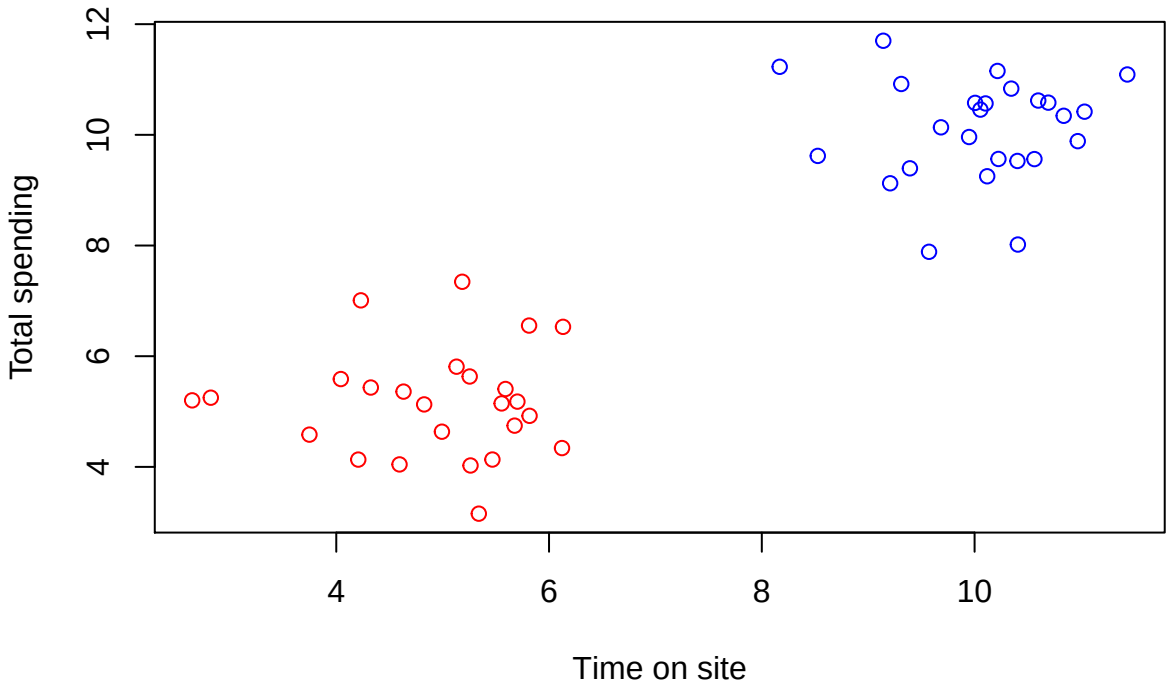
Imagine you are a data analyst for Amazon, or some other large retailer. You would like to recommend products to your customers that they would like, but how can you do this? A simple way is to recommend products that similar people have already bought.

To do this, you might look at all the behavioural data you have for each customer (time on site, previous purchases, spending, etc) and try to group all you customers into a small number of different groups. This is classic customer segmentation – finding groups such as 'new parents' or 'divorced middle-aged men' whose spending habits can be predicted. In the past this would require you to know the detailed personal information about each customer, and pick groups by hand. Now, companies can use automated methods, such as the k-means clustering algorithm we saw in week 1. Lets bring up that example again, but now I'll label the two data dimensions with plausible sounding customer behaviour metrics



Using the k-means algorithm (see week 1), we saw that we could segment this data automatically into two distinct groups. In this case those are people with high time on site and high spend, and those with low time on site and low spend. In this example the two clusters are clearly separated:

```
kmeans_object = kmeans(D, 2)
cluster_id = kmeans_object$cluster
cols = c("Red", "Blue")
plot(D[, 1], D[, 2], col=cols[cluster_id], xlab="Time on site", ylab="Total spending")
```



How is this useful to us? Well, we might imagine that individuals from each group would be interested in buying different products. Therefore, if I see someone from the high spend-high time group has bought a new iPhone, I can guess that it would be good to advertise this to the other people in this group, whereas it might be useless to advertise it to those in the low spend-low time group.

Similar principles (albeit with much more complicated data) are used to target all sorts of advertising. If you have been following the news you may be aware of the scandals concerning Cambridge Analytica (CA) and targeted political material on social media. This works in exactly the same way. CA look at data about individual's behaviour such as the websites they visit, the number of times they tweet, the words they use and the products they buy. Using this they then group them into clusters. Then, if they find that one member of a cluster responds strongly to an advert in favour of Brexit (for example) they then send similar messages to all the other members of that cluster, while sending different messages to the other clusters. Thus, each group, which is composed of people who are similar, sees different material to all the other clusters. Slowly but surely we all start to live in completely different worlds!

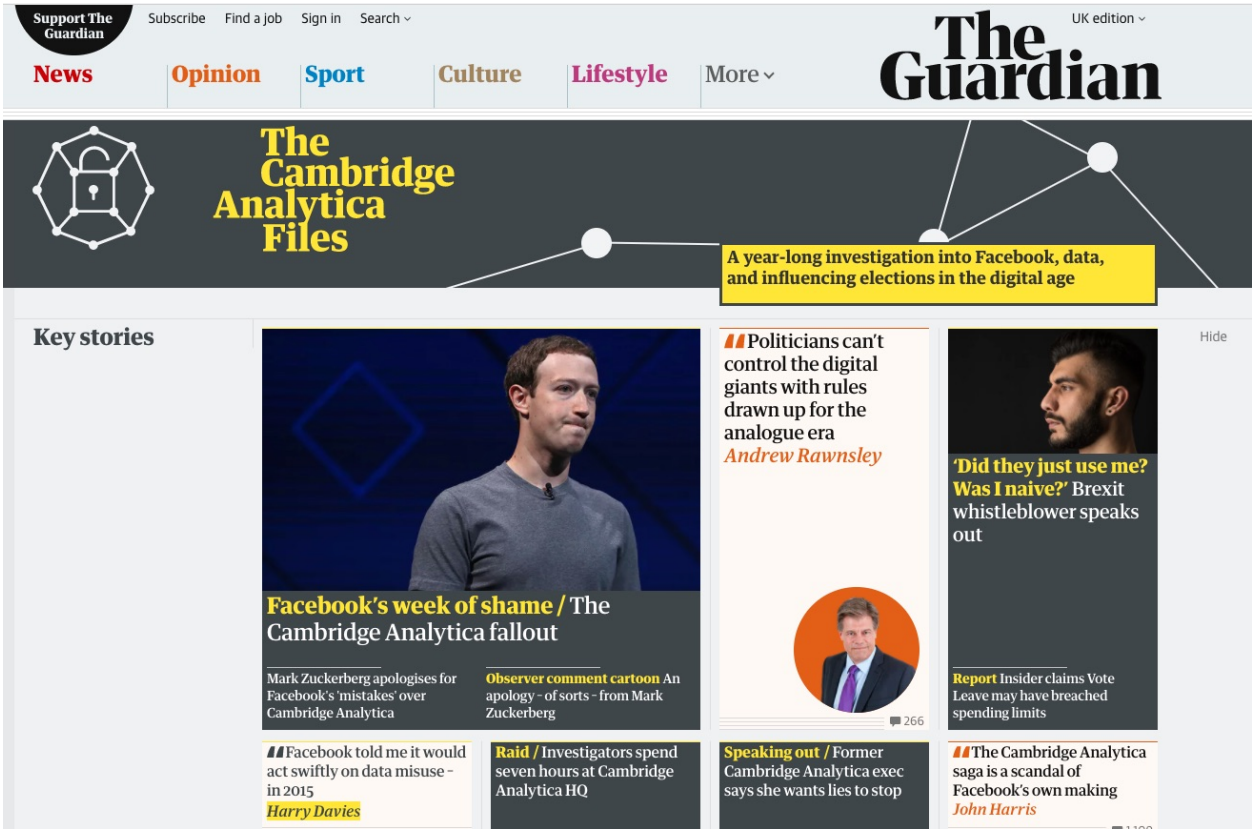


Figure 1: Find out more about how Cambridge Analytica and social media companies have been targeting political advertising at the Guardian: https://www.theguardian.com/news/series/cambridge-analytica-files

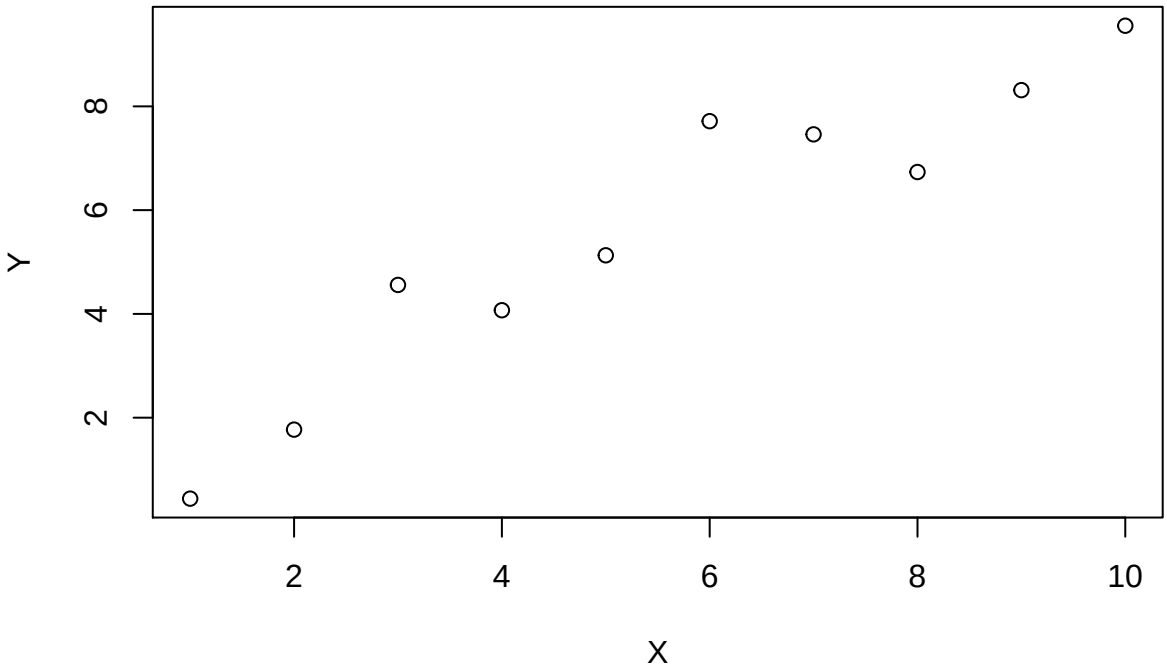
A demonstration of overfitting

Model overfitting is an important topic that we have looked at several times so far. We looked at how we can use multiple linear regression to fit non-linear polynomial models, and we also looked at using model selection methods to choose which input factors are best for making predictions. We saw that sometimes including another factor, or polynomial term, can make our predictions worse even as it makes the model fit

better. Here I want to quickly give a very clear demonstration of this effect.

First, I will generate some data using a simple linear model. By doing this I know that the data really does obey a simple linear relationship between input X and output Y, so any model that is more complex than this will be overfitting

```
X = 1:10
Y = X + rnorm(10, 0, 1)
plot(X, Y)
```



Now I will fit both a linear model and a complex polynomial model (of order 5) to these data, and look at the model fit. First I will need to set up a data frame with columns corresponding to the different polynomial powers of X

```
mydata = data.frame(X=X, X2=X^2, X3=X^3, X4=X^4, X5=X^5, Y=Y)
linear_model = glm(Y ~ X, data=mydata)
polynomial_model = glm(Y ~X+X2+X3+X4+X5, data =mydata)
print(logLik(linear_model))
```

```
## 'log Lik.' -12.83886 (df=3)
```

```
print(logLik(polynomial_model))
```

```
## 'log Lik.' -9.492867 (df=7)
```

```
print(extractAIC(linear_model)[2])
```

```
## [1] 31.67772
```

```
print(extractAIC(polynomial_model)[2])
```

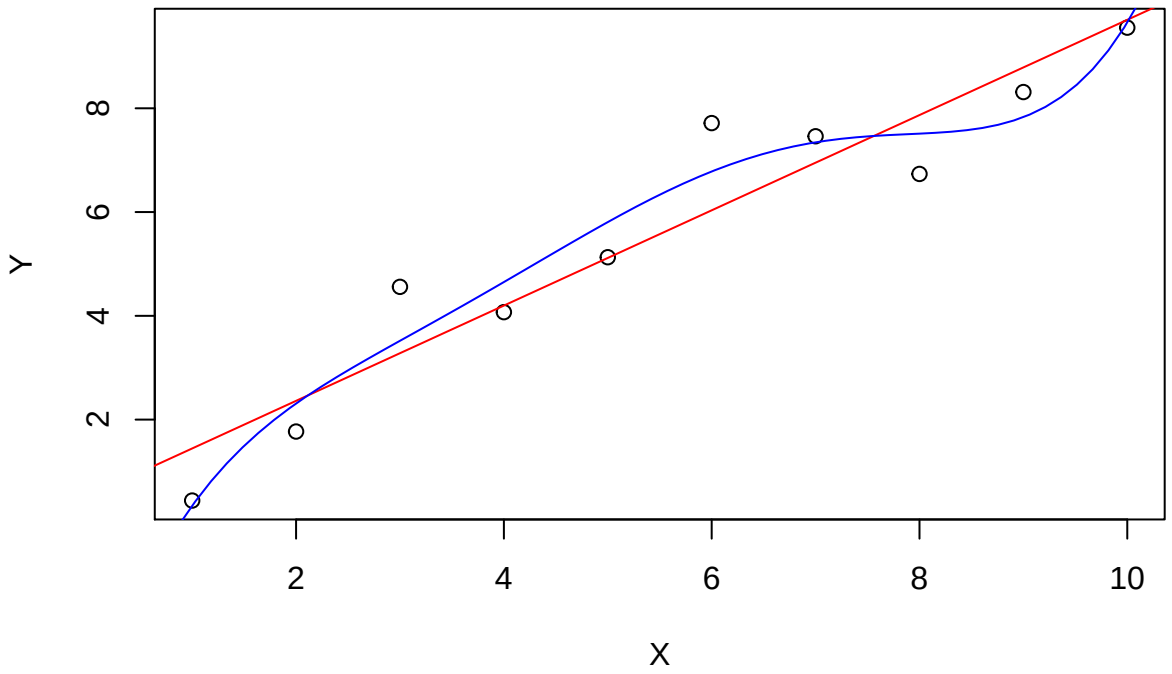
```
## [1] 32.98573
```

We can see that the polynomial model has a closer fit (higher log-likelihood), but the higher AIC suggests it will make worse predictions. Lets see what that looks like graphically.

```
fit_X = seq(-2, 13, length.out=100)
mynewdata = data.frame(X=fit_X, X2=fit_X^2, X3=fit_X^3, X4=fit_X^4, X5=fit_X^5)
predict_Y_linear = predict(linear_model, newdata=mynewdata)
predict_Y_polynomial = predict(polynomial_model, newdata=mynewdata)
```

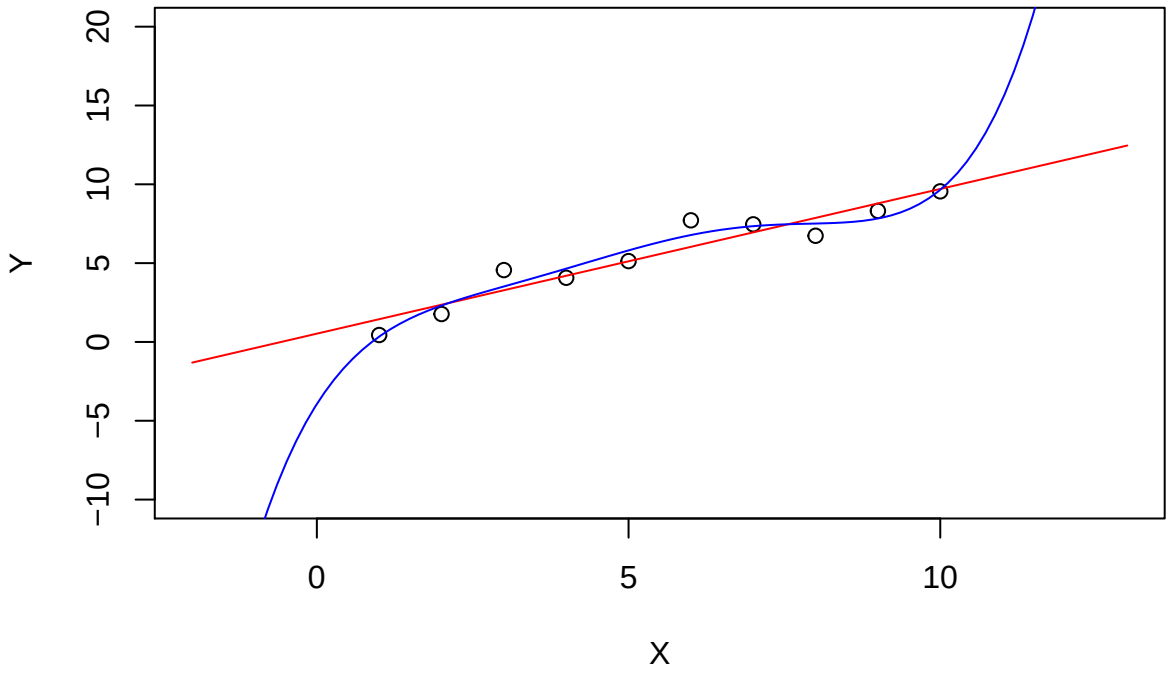
First I'll plot these predicted values on the same graph scale as above:

```
plot(X, Y, xlim = c(1, 10))
lines(fit_X, predict_Y_linear, col='red')
lines(fit_X, predict_Y_polynomial, col='blue')
```



We can see clearly here that the polynomial fit in blue is closer on average to the data points than the red linear model. So what is the problem with this, if the fit is better? The clearest way to demonstrate the problem is to zoom out a little in the graph scale

```
plot(X, Y, xlim=c(-2, 13), ylim=c(-10, 20))
lines(fit_X, predict_Y_linear, col='red')
lines(fit_X, predict_Y_polynomial, col='blue')
```



Notice how quickly the predictions of the polynomial model move away from the red line when we move outside the range of the original data. Interpolated predictions (inside the original data) are always more accurate than extrapolation (outside the original data) for any model. We don't know if the relationship between X and Y remains consistent outside of the data we have seen, and small errors in parameters can be magnified, even for a simple linear model. But in the case of overfitted models this problem is much much worse, especially for non-linear models. While the overfitted model must stay near the data within the original range, outside of this it can go rapidly up or down because of the effect of the higher order terms. This makes extrapolated predictions very poor.