MTHM506/COMM511 - Statistical Data Modelling

Topic 3 - Introduction

Preliminaries

In this session, we will start to introduce Generalised Additive Models, another, more flexible class of models that are often seen as an extension to Generalised Linear Models. In this session, we need the `mgcv` package to help us fit Generalised Additive Models. We use the `install.packages()` function to download and install the most recent package and use the `library()` function to load them into the `R` library.

```
# Installing required packages
install.packages("mgcv")

# Loading required packages into the library
library(mgcv)
```

Introduction

Previously, we learned about Generalised Linear Models (GLMs), a new framework in order to build more general models and model different data types . In Topic 3, we extend the GLM framework so that we can model the mean function using smooth functions of the covariates. Let's formalise this in a similar way to GLMs. Generalised Additive Modelling (GAM) will have a response variable, Y_i which again come from the exponential family of distributions

$$Y_i \sim EF(\theta_i, \phi)$$

Examples that we have seen of exponential family distributions are Normal, Binomial, Poisson, Negative Binomial, Exponential and Gamma. Remember, θ_i is called the location parameter and ϕ is called the scale/dispersion parameter. The location parameter relates to the mean of the distributions in this family and the dispersion relates to the variance. Again we will see that the variance will not be independent of the mean (see Slides 4-5 in Topic 2 Notes). We're working within probability distributions for which there might be a potential mean-variance relationship, therefore the variance is a scaled function of the mean.

In GLMs we specified a function of the mean $E(Y_i) = \mu_i$ of the following

$$g(\mu_i) = \eta_i = \beta_0 + \beta_1 x_{1,i} + \cdots + \beta_p x_{p,i}$$

where η_i is called the linear predictor (the part of the model we relate the response y_i and the covariates x_i). It relates to the mean of the distribution μ_i through a function $g(\cdot)$, the "link-function".

Now, in GAMs, we want to replace this linear predictor with a series of unknown functions of our parameters

$$g(\mu_i) = \eta_i = \sum_i^p f_p(x_{p,i})$$

where $f_p(\cdot)$ are a series of unknown (smooth, continuous) functions of our covariates x_{p_i}. The idea of GAMs is that we want to fit these unknown functions and not individual parameters (β). The easiest way to do this

is express our functions $f_p(\cdot)$ in a linear way using basis functions. We have seen an example (see Poisson GLMs) where we fit a polynomial function as our linear predictor so a function $f_p(\cdot)$ with a polynomial basis function would look like:

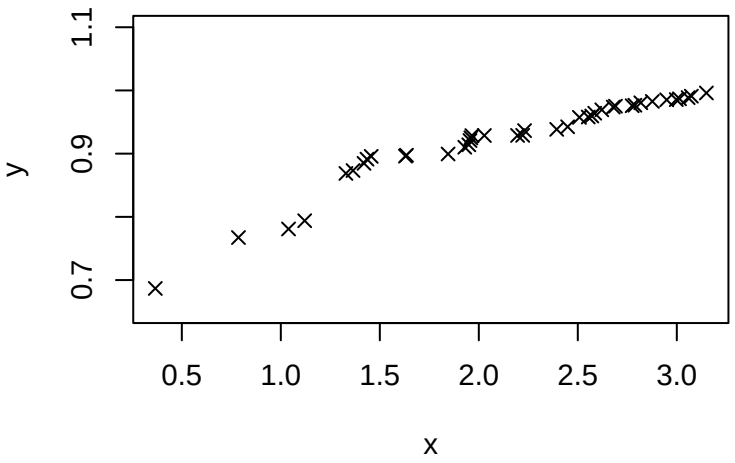
$$\begin{aligned} f_p(x_i) &= \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \ldots + \beta_q x_i^q \\ &= \beta_0 + \sum_{j=1}^{q} \beta_j x_i^j \end{aligned}$$

More generally we write $f_p(\cdot)$ as a linearly or as a sum of basis functions $b_j(\cdot)$

$$f_p(x_i) = \beta_{p,0} + \sum_{j=1}^{q} \beta_{p,j} b_{p,j}(x_i)$$

where $b_1(x_i) = x_i$, $b_2(x_i) = x_i^2$, ..., $b_j(x_i) = x_i^j$. There are certain questions that arise from this. Is this the most sensible way of doing this? And how do you decide what q (the number of basis functions) are? Can we do this as part of the inference? Let's consider it with an example on some simulated data:

```
# We will simulate data to illustrate this so we use set.seed to ensure simulated data are same every t
set.seed(1)

# Simulate an Uniform x on [0,1] and sort the values in ascending order (to create a "relationship").
# (Don't worry about the details of simulating this data, it's just for illustration purposes)
x <- sort(runif(40)*10)^0.5

# Simulate an Uniform x on [0,1] and sort the values in ascending order
y <- sort(runif(40))^0.1

# Plot the data
par(mar = c(4, 4, 1, 1),cex=1.2)
plot(x,y,pch=4,ylim=c(0.65,1.1))
```



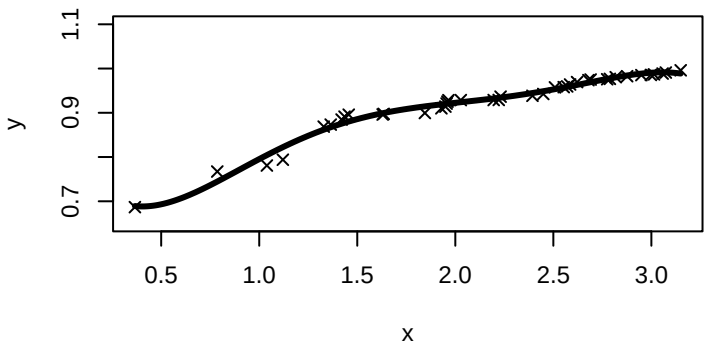
We want to fit the following model to this data:

$$Y_i \sim N(\mu_i, \sigma^2) \quad Y_i \text{ indep.}$$

$$\begin{aligned} \eta_i = \mu_i &= \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \ldots + \beta_q x_i^q \\ &= \beta_0 + \sum_{j=1}^{q} \beta_j x_i^j \end{aligned}$$

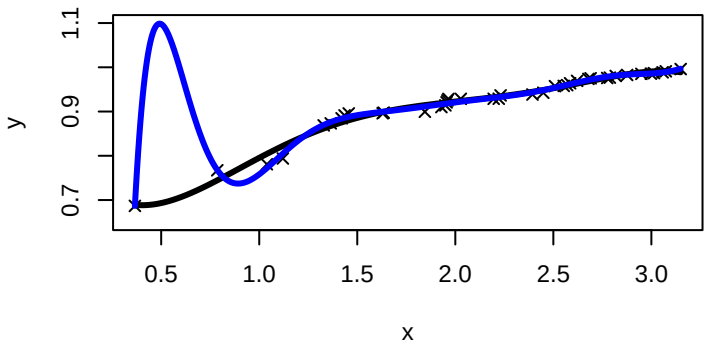
for varying levels of q. Let's start with fitting with $q = 5$ and predicting the mean over the data

```
# Fit an order 5 polynomial using Normal GLM
model <- glm(y ~ poly(x,5),
             family = gaussian(link = 'identity'))

# Create a sequence of x values for which to predict from the fitted model
xp <- seq(min(x),max(x),len=500)

# predict from model (this is the mean of our model)
preds <- predict(model,newdata=data.frame(x=xp))

# Add the predictions to the plot
plot(x,y,pch=4,ylim=c(0.65,1.1))
lines(xp,preds,lwd=3)
```



The model fits well, there is a lot of flexibility in the mean to capture all of the data. One approach discussed in the Lecture notes/recordings is to set the number of basis functions to a large number in order to give the functions too much flexibility and then think about a way of reigning it in or penalising the model for overfitting the data. Note that if you set q equal to the number of data points you end up with a saturated model which is not a good model for a number of reasons. Let's increase the maximum power q to 10.

```
# Fit the model with polynomial order up to 10
model2 <- glm(y ~ poly(x,10),
              family = gaussian(link = 'identity'))

# predict from it
preds2 <- predict(model2,newdata=data.frame(x=xp))

# and plot the predicted line
plot(x,y,pch=4,ylim=c(0.65,1.1))
lines(xp,preds,lwd=3)
lines(xp,preds2,col="blue",lwd=3)
```



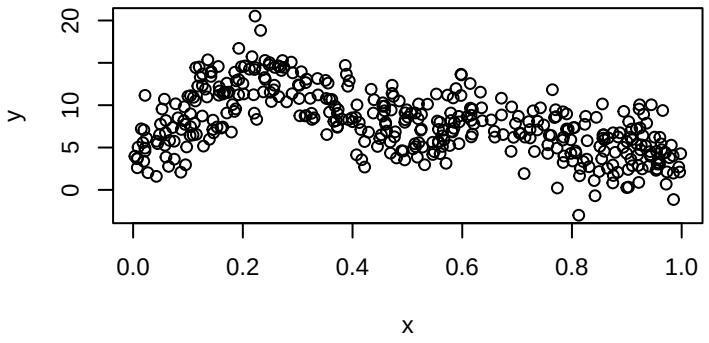
The model is doing something weird, which is a common problem with polynomial basis functions. In regions where you don't have a lot of data, the model will need to interpolate/extrapolate from the polynomial equation however there is nothing to constrain the polynomial which is why you will often get strange estimates. Therefore we can see that polynomial basis functions are unstable and not robust enough to use in a general way.

A much more stable choice for basis functions $b_{p,j}(\cdot)$ is something like cubic splines. When we use cubic splines, we specify something called knots. These knots are a bunch of points which splits up x into regions, between which we assume a cubic polynomial as our basis function. These cubic polynomials are joined together at the knot points (see Slides 15-16, Topic 3 Notes). Given knot values x_j^*, $j = 1, \ldots, q$ then we see that the basis functions are $b_{p,1}(x) = 1$, $b_{p,2}(x) = x$ and $b_{p,k}(x) = |x - x_k^*|^3$ for $k = 3, \ldots, q$ where q is called the basis dimension or rank. We still end up with a very flexible way in which complex relationships between data and covariates.

Let's simulate the data from the slides:

```
# Simulate some data:
set.seed(2)
dat <- gamSim(1,n=400,dist="normal",scale=2)
Gu & Wahba 4 term additive model

# The covariate x:
x <- dat$x2

# The response y:
y <- dat$y

# Write two functions, to compute cubic splines as in lecture notes, and create the associated model ma
# I will not expect you to understand what this are doing and I only use them here for illustration. We
# be using the function gam() to actually fit models.
rk <- function(x,z){
    abs((x-z)^3)
}
spl.X <- function(x,xk){
    q <- length(xk)+2
    n <- length(x)
    X <- matrix(1,n,q)
    X[,2] <- x
    X[,3:q] <- outer(x,xk,FUN=rk)
```

```
    X
}

# Plot x against y
plot(x, y)
```



We want to fit the following model to this data:

$$Y_i \sim N(\mu_i, \sigma^2) \quad Y_i \text{ indep.}$$

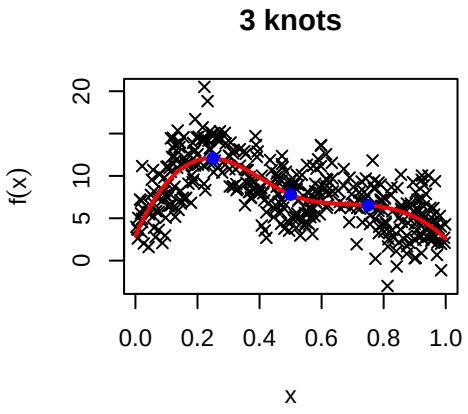
$$\eta_i = \mu_i = f(x_i)$$

$$f(x_i) = \sum_{j=1}^{q} \beta_j b_j(x_i)$$

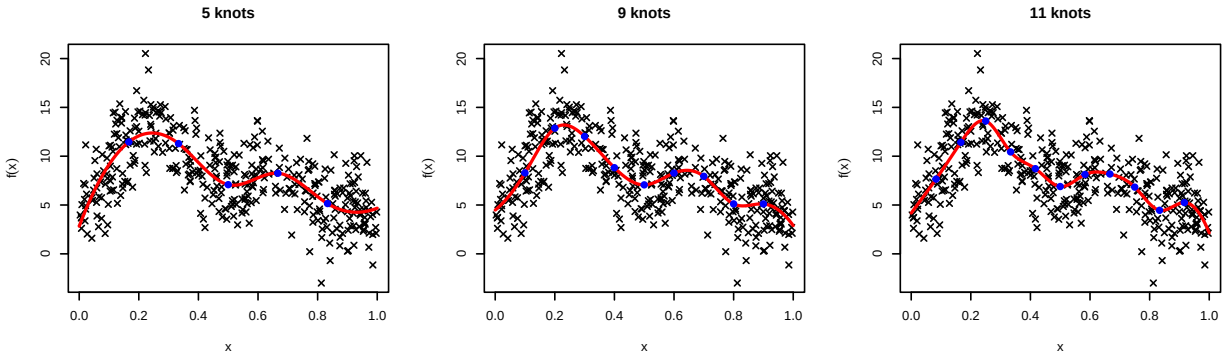
where $b_1(x) = 1$, $b_2(x) = x$ and $b_{p,k}(x) = |x - x_k^*|^3$ for $k = 3, \ldots, q$. Lets fit a model with 3 knots (i.e. rank $q = 5$) at 0.25, 0.5 and 0.75.

```
# Additive model with a cubic spline with 3 knots placed at 0.25, 0.5, 0.75
xk <- c(0.25,0.5,0.75)

# Model matrix
X <- spl.X(dat$x2,xk) # Calculate the model matrix

# Fit the model
model <- glm(y~X-1,data=dat,family=gaussian) # Fit the model (basically a linear model)

# Summarise the model
summary(model)

Call:
glm(formula = y ~ X - 1, family = gaussian, data = dat)

Coefficients:
   Estimate Std. Error t value Pr(>|t|)
X1   20.400      1.533  13.309  < 2e-16 ***
X2  -28.285      2.886  -9.801  < 2e-16 ***
```

```
X3   83.833      9.043   9.271  < 2e-16 ***
X4 -200.892     27.153  -7.399 8.34e-13 ***
X5   15.147      9.273   1.633    0.103
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 7.142478)

    Null deviance: 30013.2  on 400  degrees of freedom
Residual deviance:  2821.3  on 395  degrees of freedom
AIC: 1928.5

Number of Fisher Scoring iterations: 2

# Now estimate the mean of the model at a fine grid between 0 and 1
xp <- seq(0,1,length=500)

# Predicted mean
mu_hat <- spl.X(xp,xk)%*%coef(model) ## This is X'beta

# Predict the mean of the model
plot(dat$x2,dat$y,pch=4,lwd=1,main="3 knots",xlab="x",ylab=expression(f(x)))
lines(xp,mu_hat,col="red",lwd=2)
points(xk,spl.X(xk,xk)%*%coef(model),col="blue",pch=19) # indicate where the knots are
```



Let's also increase the number of knots a few times to 5, 7 and 9 and see the effects (code not shown) and compare the results



The more knots we have the more "wiggly" a function we predict, this is because increasing the number of knots in a model gives it more flexibility. The results are much more stable and using cubic splines means we won't get any weirdness like we did with polynomial basis functions.

Now we have a way of modelling using smooth functions of our covariates, the question then becomes is how do we choose our rank q (or number of basis functions)? Choose a rank q too small and the function is not flexible enough, but a rank q too large will lead to overfitting (See Slide 19 Topic 3 Notes). Using the above, how to we choose between 3, 5, 7 and 9 knots? You could make an argument for each. Therefore we need a way to be objective about it and let the data inform us about it so we do not need to make the decision ourselves.

The easiest way to do this in terms of a statistical framework or model is to simply choose too many of the basis functions (that computation would reasonably allow) so that in principal the function is super wiggly and then find a way to "penalise" the model for "overexplaining" the data or having a function that is too flexible. This is the fundamental idea of GAMs.

Well this leads us into penalised likelihood, takes the log-likelihood $\ell(\boldsymbol{\theta}, \phi, \boldsymbol{y})$ and adds on an extra term

$$\ell(\boldsymbol{\theta}, \phi, \boldsymbol{y}) - \lambda \int_x [f''(x)]^2 dx$$

the second term is a penalty for smooth functions $f(\cdot)$ which is controlled through a smoothing parameter λ. Note that in these models in above we have chosen to write our functions in the form

$$\eta_i = f(x_i) = \sum_{j=1}^{q} \beta_j b_j(x_i)$$

$$\boldsymbol{\eta} = \boldsymbol{X}\boldsymbol{\beta}$$

The models are still LINEAR in terms of the parameters β_i's. The more β's we have in our model, the more likely we are to overfit. So adding this penalty term, stops this from happening. We penalise the model using the second derivative of $f(\cdot)$, $f''(\cdot)$, as the second derivative measures the rate of change. The more wiggly or flexible our function is the larger $f''(\cdot)$ will be. So the the integral measures how wiggly or flexible the model is; if $f(\cdot)$ is smooth the integral will be small, and if $f(\cdot)$ is wiggly the integral will be large. The smoothing parameter λ then controls how much we penalise the model/function for being too wiggly. If $\lambda \to 0$ then we end up with unpenalised splines (like above), and if $\lambda \to \infty$ then we end up with a straight line.

Now we have a new penalised likelihood to work with, the question then is how do we estimate λ? Do we treat it as another parameter and use MLE? Well no, because if we if we try to maximise the likelihood then the model will always opt for maximum flexibility. Remember that the likelihood is the joint probability of observing the data given the model parameters, and models with more parameters will always have a larger likelihood. So by having a term that tries to take away some of that flexibility means that MLE will set $\lambda = 0$.

Unfortunately, we must find out a different way of estimating λ. From Slide 22 Topic 3 Notes, we see a mathematical argument for a method, through the use of ordinary cross validation scores, to choose the optimum λ. The ordinary cross validation score

$$V = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}^{-i}(x_i))^2$$

where $\hat{f}^{-i}(x_i)$ is the prediction from the model fitted to the data with i removed. In essence, we use the predictability of our function to choose the correct λ. It is the mean square error of predictions using data that the model hasn't seen. A model that overfits the data, will be very good at predicting data that is used to predict it, but not those predicting outside the range of the data. Conversely, having a function that is too restrictive means that we wont identify "signals" or patterns in the data and therefore will not be particularly good at predicting either. So by using V, which is a measure of predictability of a model makes sense to help us estimate λ.

One way to do this is to choose a few values of λ and find the one that minimises V (i.e. the best predictability). However, leave-one-out cross validation can be very computationally expensive, especially if you are having to conduct it may times to find an optimum λ. Fortunately, the `mgcv` package in `R`, which we will use to fit these models, does this for us. But instead of using leave-one-out cross validation it does Generalised Crossed Validation, hence the name `mgcv`.

The final thing we need to consider is how do we fit this model? When we were in the GLM framework we used the Iterative Reweighted Least Squares (IRLS) algorithms to maximise the likelihood (MLE). Instead, as we are now in the GAM framework we have a penalised likelihood and we must use Penalised Iterative Reweighted Least Squares (P-IRLS) algorithms. Again, the `mgcv` package in `R` does this for us.

Once the model is fit how do we perform inference on our models (parameters, LRTs, AIC etc.)? Well we use a trick where we condition on the estimated smoothing parameter treating it as "known", and ignore any uncertainty on it. Then once we have done this, inference is done in the same way as if we had fit this as a GLM. Let's assume that we know λ,

$$\ell(\boldsymbol{\theta}, \phi, \boldsymbol{y}) - \lambda \int_x [f''(x)]^2 dx$$

then we would just be maximising the likelihood as in GLMs as the second term here is a constant. So we're just fitting our GAM, ignore that λ exists and perform exactly the same inference as we learned in Topic 2.

When doing t-tests on model parameters or performing LRTs, these tests need the degrees of freedom of our model. We can't simply set it to $n - p - 1$ anymore as we are now penalising the model parameters as we know there are too many. In effect, we are constraining them. Instead we use something called the effective degrees of freedom (EDF), which is the number of parameters minus the number of "constraints" within our tests. Therefore a GAMs degrees of freedom becomes $n - EDF$.

This is a lot of theory, and it sounds more complicated than it is to do it so let's consider an example.