GAMs

In essence, a GAM is a GLM. What distinguishes it from the ones you know is that, unlike a standard GLM, it is composed of a sum of smooth functions of covariates instead of or in addition to the standard linear covariate effects

However, an additional aspect is that we will use penalized estimation, something that is quite common in some modeling contexts (e.g. machine learning), but not common enough in applied research.

For those new to penalized regression, again consider a standard GLM that we usually estimate with maximum likelihood, l(β)l(β), where ββ are the associated regression coefficients

Whenever we build statistical models, we face a trade-off between flexibility and interpretability. GAMs offer a middle ground between simple models, such as those we fit with linear regression, and more complex machine learning models like neural networks.

Linear models are easy to interpret and to use for inference: It is easy to understand the meaning of their parameters. However, we often need to model more complex phenomena than can be represented by linear relationships.

On the other hand, machine learning models, like boosted regression trees or neural networks, can be very good at making predictions of complex relationships. The problem is that they tend to need lots of data, are quite difficult to interpret, and one can rarely make inferences from the model results.

GAMs offer a middle ground: they can be fit to complex, nonlinear relationships and make good predictions in these cases, but we are still able to do inferential statistics and understand and explain the underlying structure of our models and why they make predictions that they do.

Diagram

Description automatically generated

GAMs let us flexibly model non-linear relationships. Here I've made a scatter plot of two variables, X and Y. We can see from the scatterplot that there is clearly some relationship between the variables, but it is not linear.

Now let us make another generalization to incorporate nonlinear forms of the predictors, via a generalized additive model. The form y∼ExpoFam(μ,etc.)E(y)=μg(μ)=b0+f(x1)+f(x2)+...+f(xp)y∼ExpoFam(μ,etc.)E(y)=μg(μ)=b0+f(x1)+f(x2)+...+f(xp)at the right gives the new setup relating our new, now nonlinear predictor to the expected value, with whatever link function may be appropriate.

So what’s the difference? In short, we are using smooth functions of our predictor variables, which can take on a great many forms, with more detail on what that means in the following section. For now, it is enough to note the observed values yy are assumed to be of some exponential family distribution, and μμ is still related to the model predictors via a link function. The key difference is that the linear predictor now incorporates smooth functions of at least some (possibly all) covariates, represented as f(x)f(x), and this will allow for nonlinear relationships between the covariates and the target variable yy.

Chart, scatter chart

Description automatically generated

If we fit a linear model to the data using the lm() function and the usual formula syntax, we can see it won't do a very good job. The model doesn't capture key aspects of this relationship.

linear\_mod <- lm(y ~ x, data = my\_data)

Chart, scatter chart

Description automatically generated

With a GAM, however, we can fit data with smooths, or splines, which are functions that can take on a wide variety of shapes. We fit a GAM using the gam() function from the mgcv package. Here, when we fit this GAM, we wrap the independent variable, x, in the s(), that is smooth function to specify that we want this relationship to be flexible. A GAM can capture the nonlinear aspects of not only this relationship, but of many nonlinear relationships, because of the flexibility of splines.

library(mgcv)

gam\_mod <- gam(y ~ s(x), data = my\_data)

Chart, line chart

Description automatically generated

The flexible smooths in GAMs are actually constructed of many smaller functions. These are called basis functions. Each smooth is the sum of a number of basis functions, and each basis function is multiplied by a coefficient, each of which is a parameter in the model. In the plot here on the left, we show the basis functions of a GAM where all the coefficients are the same. On the right, we show the same basis functions after model-fitting, where each has a coefficient fit to the data. You can see how these basis functions add up to create the overall smooth shape. So a single nonlinear relationship between a dependent and independent variable has several parameters, plus an intercept. This is different, and more complex, than a linear model, where each variable has only a single coefficient or parameter.

Chart, scatter chart

Description automatically generated

When we fit a GAM with R, we can extract the coefficients just like we can for linear models using the coef() function. Calling this function on a GAM model object will show the coefficients of each of the basis functions of the model. You can see that even a simple one-smooth model has many coefficients.

gam\_mod <- gam(y ~ s(x), data = my\_data)

coef(gam\_mod)

(Intercept) s(x2).1 s(x2).2

7.814448 5.272290 5.104941

s(x2).3 s(x2).4 s(x2).5

1.271135 1.720561 -1.180613

s(x2).6

-2.676133

mcycle <- MASS::mcycle

# Examine the mcycle data frame

head(mcycle)

plot(mcycle)

|  |  |
| --- | --- |
| # Fit a linear model  lm\_mod <- lm(accel ~ times, data = mcycle)  # Visualize the model  termplot(lm\_mod, partial.resid = TRUE, se = TRUE) | # Load mgcv  library(mgcv)  # Fit the model accel Vs. times  gam\_mod <- gam(accel ~ s(times), data = mcycle)  # Plot the results  plot(gam\_mod, residuals = TRUE, pch = 1) |

Chart, scatter chart

Description automatically generatedDiagram

Description automatically generated

# Extract the model coefficients

coef(gam\_mod)

Loading required package: nlme

This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.

(Intercept)

-25.5458646616541

s(times).1

-63.7180075250146

s(times).2

43.4756443634788

s(times).3

-110.350131585746

s(times).4

-22.1810063562354

s(times).5

35.0344227720567

s(times).6

93.1764583974439

s(times).7

-9.28301845597997

s(times).8

-111.661471743254

s(times).9

17.6037820246907

Now that you have a sense of how GAMs can fit nonlinear data, let's learn a bit more about how they work. GAMs are powerful because of their ability to take on many shapes, but this is also what makes them challenging. Their flexibility makes it easy to over-fit your data. Here we'll learn how smoothing helps us deal with this issue.

The number of basis functions in a smooth has a great impact on the shapes a model can take.

mcycle <- MASS::mcycle

library(mgcv)

# Fit a GAM with 3 basis functions

gam\_mod\_k3 <- gam(accel ~ s(times, k = 3), data = mcycle)

# Fit with 20 basis functions

gam\_mod\_k20 <- gam(accel ~ s(times, k = 20), data = mcycle)

# Visualize the GAMs

par(mfrow = c(1, 2))

plot(gam\_mod\_k3, residuals = TRUE, pch = 1)

plot(gam\_mod\_k20, residuals = TRUE, pch = 1)

Diagram

Description automatically generated

The smoothing parameter balances between likelihood and wiggliness to optimize model fit. Here, you’ll examine smoothing parameters and will fit models with different fixed smoothing parameters.

mcycle <- MASS::mcycle

library(mgcv)

# Extract the smoothing parameter

gam\_mod <- gam(accel ~ s(times), data = mcycle, method = "REML")

gam\_mod$sp

# Fix the smoothing parameter at 0.1

gam\_mod\_s1 <- gam(accel ~ s(times), data = mcycle, sp = 0.1)

# Fix the smoothing parameter at 0.0001

gam\_mod\_s2 <- gam(accel ~ s(times), data = mcycle, sp=0.0001)

# Plot both models

par(mfrow = c(2, 1))

plot(gam\_mod\_s1, residuals = TRUE, pch = 1)

plot(gam\_mod\_s2, residuals = TRUE, pch = 1)

Loading required package: nlme

This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.

**s(times):** 0.000775803587936321

Chart, diagram

Description automatically generated

The number of basis functions and the smoothing parameters interact to control the wiggliness of a smooth function. Here you will see how changing both together affects model behavior.

mcycle <- MASS::mcycle

library(mgcv)

# Fit the GAM

gam\_mod\_sk <- gam(accel ~ s(times, k=50), data = mcycle, sp = 0.1)

# Visualize the model

plot(gam\_mod\_sk, residuals = TRUE, pch = 1)

Diagram

Description automatically generated

**Multiple Regression with GAMs**

So far, all the GAMs we have seen have been univariate; they have a single predictor for the outcome. However, we can perform multiple regression with GAMs. Multiple GAMS also can contain a mixture of smooths, linear effects, and continuous or categorical variables. In this lesson, we'll learn how to use this flexibility to fit a variety of different models to data

We'll now work with the mpg data set. This is a data set of 205 models of cars, consisting of various traits like their make, model, cylinders, price and weight, and their city and highway fuel efficiency. We'll be building models that use the vehicle traits to predict fuel efficiency.

# Working Dataset: mpg

mpg

symbol loss make fuel aspir doors style drive eng.loc wb length width

1 3 NA alfa-romero gas std two convertible rwd front 88.6 168.8 64.1

2 3 NA alfa-romero gas std two convertible rwd front 88.6 168.8 64.1

3 1 NA alfa-romero gas std two hatchback rwd front 94.5 171.2 65.5

4 2 164 audi gas std four sedan fwd front 99.8 176.6 66.2

5 2 164 audi gas std four sedan 4wd front 99.4 176.6 66.4

6 2 NA audi gas std two sedan fwd front 99.8 177.3 66.3

7 1 158 audi gas std four sedan fwd front 105.8 192.7 71.4

8 1 NA audi gas std four wagon fwd front 105.8 192.7 71.4

9 1 158 audi gas turbo four sedan fwd front 105.8 192.7 71.4

10 0 NA audi gas turbo two hatchback 4wd front 99.5 178.2 67.9

...

Let's start with a very simple one variable model. Here is the code for a model that predicts highway fuel efficiency as a smooth function of automobile weight. The resulting model captures the nonlinear decreasing relationship between these two variable

model <- gam(hw.mpg ~ s(weight), data = mpg,

method = "REML")

Chart, scatter chart

Description automatically generated

To add an additional variable, such as length, we just include another s() function in our formula, separated by a plus sign. Here, we add car length as another predictor.

model <- gam(hw.mpg ~ s(weight), data = mpg,

method = "REML")

model2 <- gam(hw.mpg ~ s(weight) + s(length), data = mpg,

method = "REML")

We see from these plots that length has increasing nonlinear effect on fuel economy, and this effect is weaker than the weight effect.

Note that, in this model, both the effect of weight and price are non-linear terms, but the two are simply added together to get a final prediction. That addition is where the additive in generalized additive models comes from.

Chart, scatter chart

Description automatically generated

Not every term in a GAM has to be nonlinear. You can combine linear and nonlinear terms. To add a linear term, don't wrap the predictor in the s() function. Here I've made the length term from theprevious model linear.

In practice, we rarely make continuous variables linear in GAMs. This is because, if the relationship is really linear, or there is not enough data to show otherwise, automatic smoothing will force a linear shape.

model2 <- gam(hw.mpg ~ s(weight) + length, data = mpg,

method = "REML")

Chart, scatter chart

Description automatically generated

We can produce the same model if we set the smoothing parameter of the length term very high. Strong smoothing results in a linear model

model2b <- gam(hw.mpg ~ s(weight) + s(length, sp = 1000), data = mpg,

method = "REML")

However, linear terms are very useful when we have categorical variables as predictors. For instance, in our mpg data set, the fuel variable has two categories: diesel and gas. When we include a linear term with this categorical variable, the gam() function fits a model with a fixed effect for each level of the category. Here, you can see that having gasoline has a negative effect; diesel vehicles are more fuel efficient.

In this model the nonlinear effect of weight applies to vehicles of both gas types. This is nonlinear equivalent to the fixed slope, varying intercept models you may recall from a linear regression course.

Note that, when you use categorical variables this way, it's important that the variables are stored as factors. The mgcv package does not use character variables.

model3 <- gam(hw.mpg ~ s(weight) + fuel, data = mpg,

method = "REML")

Chart, scatter chart

Description automatically generated

We can also specify a GAM formula that will fit different smooths for different categorical variables. We call this a factor-smooth interaction. By specifying the "by" argument to the s() function, we can tell R to calculate a different smooth for each unique category.

You can see here we have different smooths for diesel and gas cars, but the diesel smooth is much more uncertain

model4 <- gam(hw.mpg ~ s(weight, by = fuel), data = mpg,

method = "REML")

Chart, histogram

Description automatically generated

Usually, when we have smooth-factor interactions, we want to also include a varying intercept, in case the different categories are different in overall means in addition to shape of their smooths. Here, you see adding this varying intercept improves the estimate of the smooth for diesel cars.

model4b <- gam(hw.mpg ~ s(weight, by = fuel) + fuel, data = mpg,

method = "REML")

Chart, histogram

Description automatically generated

library(gamair)

data("mpg", package="gamair")

library(mgcv)

# Examine the data

head(mpg)

str(mpg)

# Fit the model

mod\_city <- gam(city.mpg ~ s(weight) + s(length) + s(price),

data = mpg, method = "REML")

# Plot the model

plot(mod\_city, pages = 1)

Diagram, engineering drawing

Description automatically generated

Now you’ll include categorical variables in your model. Categories are inherently linear, so you’ll model them as linear terms.

library(gamair)

data("mpg", package="gamair")

library(mgcv)

# Fit the model

mod\_city2 <- gam(city.mpg ~ s(weight) + s(length) + s(price)+ fuel + style+ drive, data = mpg, method = "REML")

# Plot the model

plot(mod\_city2, all.terms = TRUE, pages = 1)

Diagram

Description automatically generated

library(gamair)

data("mpg", package="gamair")

library(mgcv)

# Fit the model

mod\_city3 <- gam(city.mpg ~ s(weight, by = drive) + s(length, by = drive) + s(price, by = drive) + drive,

data = mpg, method = "REML")

# Plot the model

plot(mod\_city3, pages = 1)

Diagram, engineering drawing

Description automatically generated

# Interpreting GAM outputs

As with other models in R, you can get a summary of model statistics with the summary() function. Let's look at this output for a model we fit with the mpg data.

mod\_hwy <- gam(hw.mpg ~ s(weight) + s(rpm) +

s(price) + s(comp.ratio) +

s(width) + fuel + cylinders,

data = mpg, method = "REML")

summary(mod\_hwy)

The first part of the summary describes the model we fit. The "Family" component tells us the model assumes a Gaussian or normal distribution of our errors, and the "Link" of "identity" shows that the model doesn't transform the predictions. We'll talk more about transformation in the final chapter of the course.

After that, we have the model formula.

Family: gaussian

Link function: identity

Formula:

hw.mpg ~ s(weight) + s(rpm) + s(price) +

s(comp.ratio) + s(width) + fuel

The next section describes the parametric terms of our model. Parametric means models that have a pre-determined form. In this context, it refers to the linear terms in the model.

This section may be familiar from linear modeling. It shows the coefficients for the linear terms in the model, their values, errors, test statistics, and p-values. Asterisks next to the p-values indicate statistical significance. In this case, the model intercept is significant, but the fixed effect of fuel type is only significant at the 0.1 level.

Parametric coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 23.873 3.531 6.760 1.89e-10 \*\*\*

fuelgas 7.571 3.922 1.931 0.0551 .

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 '

The next section covers smooth terms. For smooths coefficients are not printed. This is because each smooth has several coefficients - one for each basis function. Instead, the first column reads edf, which stands for effective degrees of freedom. This value represents the complexity of the smooth. An edf of 1 is equivalent to a straight line. An edf of 2 is equivalent to a quadratic curve, and so on, with higher edfs describing more wiggly curves.

Approximate significance of smooth terms:

edf Ref.df F p-value

s(weight) 6.254 7.439 20.909 < 2e-16 \*\*\*

s(rpm) 7.499 8.285 8.534 2.07e-09 \*\*\*

s(price) 2.681 3.421 1.678 0.155

s(comp.ratio) 1.000 1.001 18.923 2.22e-05 \*\*\*

s(width) 1.001 1.001 0.357 0.551

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

We can see this by plotting. Here are partial effect plots for the weight and compression ratio smooths. You can see the weight smooth, with an edf over 6, is complex and wiggly. But the compression ratio smooth, with an edf of 1, is linear.

Chart, scatter chart

Description automatically generated The terms to the right of the EDF column have to do with significance testing for smooths. The Ref.df and F columns are test statistics used in an ANOVA test to test overall significance of the smooth. The result of this test is the p-value to the right. It's important to note that these values are approximate, and it's important to visualize your model to check them.

A good way to interpret significance for smooth terms in GAMs is this: a significant smooth term is one where you can not draw a horizontal line through the 95% confidence interval.

If we look at plots of the weight and price smooths, we see this. Clearly a horizontal line can't go through the weight smooth confidence interval, but a horizontal line just fits inside the price smooth interval.

Chart, scatter chart

Description automatically generated

Note that high EDF doesn't mean significance or vice-versa. A smooth may be linear and significant, non-linear and non-significant, or one of each.

In this model, the price term is non-linear but non-significant, meaning it has some complexity but there isn't certainty as to the shape or direction of its effect. Compression ratio is linear but significant. Width is neither.

Chart

Description automatically generated

library(gamair)

data("mpg", package="gamair")

library(mgcv)

# Fit the model

mod\_city4 <- gam(city.mpg ~ s(weight) + s(length) + s(price) + s(rpm) + s(width),

data = mpg, method = "REML")

# View the summary

summary(mod\_city4)

Loading required package: nlme

This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.

Family: gaussian

Link function: identity

Formula:

city.mpg ~ s(weight) + s(length) + s(price) + s(rpm) + s(width)

Parametric coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 25.201 0.188 134 <2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:

edf Ref.df F p-value

s(weight) 5.620 6.799 17.524 < 2e-16 \*\*\*

s(length) 2.943 3.759 0.904 0.420

s(price) 1.000 1.000 16.647 6.68e-05 \*\*\*

s(rpm) 7.751 8.499 16.486 < 2e-16 \*\*\*

s(width) 1.003 1.005 0.006 0.939

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) = 0.831 Deviance explained = 84.7%

-REML = 496.47 Scale est. = 7.0365 n = 199

The GCV, or generalized cross validation score can be taken as an estimate of the mean square prediction error based on a leave-one-out cross validation estimation process. We estimate the model for all observations except ii, then note the squared residual predicting observation ii from the model. Then we do this for all observations. However, the GCV score is an efficient measure of this concept that doesn’t actually require fitting all those models and overcomes other issuesIn this initial model the GCV can be found as:  
GCV=n∗scaledest.(n−edf−[nofparametricterms])2GCV=n∗scaledest.(n−edf−[nofparametricterms])2. It is this score that is minimized by default when determining the specific nature of the smooth. On its own it doesn’t tell us much, but we can use it similar to AIC as a comparative measure to choose among different models, with lower being better.

What are we getting here? The same thing you get from a regular linear model, because you just ran one. However, there are a couple things to look at. The coefficient is statistically significant, but serves as a reminder that it usually a good idea to scale predictor variables so that the effect is more meaningful. Here, moving one unit on Income is akin from going broke to being the richest country. But in a more meaningful sense, if we moved from say, .7 to .8, we’d expect an increase of about 35 points on the science score. We also see the deviance explained[10](https://m-clark.github.io/generalized-additive-models/application.html#fn10), which serves as a generalization of R-squared, and in this case, it actually is equivalent to the unadjusted R-squared. Likewise, there is the familiar adjusted version of it to account for small sample size and model complexity. The scale estimate is the scaled deviance, which here is equivalent to the residual sums of squares. The GCV score we will save for when we run a GAM.

After that we see that the output is separated into parametric and smooth, or nonparametric parts.As an aside, the term nonparametric has at least two general uses in the statistical world. [Wikipedia](https://en.wikipedia.org/wiki/Nonparametric_statistics) has a nice delineation. In this case, the only parametric component is the intercept, but it’s good to remember that you are not bound to smooth every effect of interest, and indeed, as we will discuss in more detail later, part of the process may involve refitting the model with terms that were found to be linear for the most part anyway

In this situation, we are still trying to minimize the residual sums of squares, but we also have a built-in penalty for ‘wiggliness’ of the fit, where in general we try to strike a balance between an undersmoothed fit and an oversmoothed fit. The default p-value for the test is based on the effective degrees of freedom and the rank rr of the covariance matrix for the coefficients for a particular smooth, so here, conceptually, it is the p-value associated with the F(r,n−edf

# Visualizing GAMs

The plots generated by mgcv's plot() function are partial effect plots. That is, they show the component effect of each of the smooth or linear terms in the model, which add up to the overall prediction.

Chart

Description automatically generated with low confidence

The first option we have when making our plots is which partial effects to show. The select argument chooses which terms we plot, with the default being all of them.

Normally, each plot gets its own page, but using the pages argument, you can decide how many total pages to spread plots across. Using pages = 1, you show all your partial effects together.

Finally, by default we only see the smooth plots, but by setting all.terms = TRUE, we can display partial effects of linear or categorical terms, as wel

plot(gam\_model, select = c(2, 3))

plot(gam\_model, pages = 1)

plot(gam\_model, pages = 1, all.terms = TRUEChart, histogram

Description automatically generated

We often want to show data alongside model predictions. There are two ways to do this. First, the rug argument puts X-values along the bottom of the plot.

plot(gam\_model, rug = TRUE)

Chart, histogram

Description automatically generated

The residuals argument puts partial residuals on the plots. Partial residuals are the difference between the partial effect and the data, after all other partial effects have been accounted for.

plot(gam\_model, residuals = TRUE)

Chart, line chart

Description automatically generated

It's often helpful to make these more visible with the pch argument, which changes the shape of the residuals points and the cex argument, which changes the size.

plot(gam\_model, rug = TRUE, residuals = TRUE,

pch = 1, cex = 1)

Chart, scatter chart

Description automatically generated

By default, plot will put standard errors on your plots. These show the 95% confidence interval for the mean shape of the effect.

plot(gam\_model, se = TRUE)-

Chart, histogram

Description automatically generated

It's often preferable to use shading rather than lines to show these intervals, and you can do so with the shade argument.

plot(gam\_model, shade = TRUE)

Chart, histogram

Description automatically generated

You can also change the color of shading with the shade.col argument.

plot(gam\_model, shade = TRUE, shade.col = "lightblue")

Chart, line chart, histogram

Description automatically generated

It's often useful to plot the standard errors of a partial effect term combined with the standard errors of the model intercept. This is because confidence intervals at the mean value of a variable can be very tiny, and don't reflect overall uncertainty in our model. Using the seWithMean argument adds in this uncertainty.

plot(gam\_model, seWithMean = TRUE)

Chart, histogram

Description automatically generated

To make the plots even more interpretable, it's useful to shift the scale so that the intercept is included. Using the shift argument, we can shift the scale by value of the intercept, which is the first coefficient of the model. Note how the y-axis has changed. Now, the partial effect plot has a more natural interpretation - it shows us the prediction of the output, assuming other variables are at their average value. For instance, this plot shows that the miles per gallon of a 2000 pound car is about 30, all else being equal.

plot(gam\_model, seWithMean = TRUE, shift = coef(gam\_model)[1])

Chart, histogram

Description automatically generated

mcycle <- MASS::mcycle

library(mgcv)

# Fit the model

mod <- gam(accel ~ s(times), data = mcycle, method = "REML")

# Make the plot with residuals

plot(mod, residuals = TRUE)

# Change shape of residuals

plot(mod, residuals = TRUE, pch = 1, cex = 1)

Chart, histogram

Description automatically generated Diagram

Description automatically generated

library(gamair)

data("mpg", package="gamair")

library(mgcv)

# Fit the model

mod <- gam(hw.mpg ~ s(weight) + s(rpm) + s(price) + comp.ratio, data = mpg, method = "REML")

# Plot the price effect

plot(mod, select = 3)

# Plot all effects

plot(mod, pages = 1, all.terms = TRUE)

Chart

Description automatically generated

Diagram

Description automatically generated

library(gamair)

data("mpg", package="gamair")

library(mgcv)

# Fit the model

mod <- gam(hw.mpg ~ s(weight) + s(rpm) + s(price) + comp.ratio,

data = mpg, method = "REML")

# Plot the weight effect

plot(mod, select = 1, shade = TRUE, shade.col = "hotpink")

# Make another plot adding the intercept value and uncertainty

plot(mod, select = 1, shade = TRUE, shade.col = "hotpink",

shift = coef(mod)[1], seWithMean = TRUE)

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

We've learned that the number of basis functions determines how wiggly a smooth can be. If there are not enough basis functions, it may not be wiggly enough to capture the relationships in data. Here is a model where smooths have 4 basis functions. As we see in this partial effect plot, this is not enough to capture the pattern.

It's not always obvious visually whether we have enough basis functions. We can test for this, though, via the gam.check() function.

mod <- gam(y ~ s(x1, k = 4) + s(x2, k = 4),

data = check\_data, method = "REML")

Chart, scatter chart

Description automatically generated

Running gam.check() on a model provides several outputs, in both the console and as plots. We'll start with the console output.

First, gam.check() reports on model convergence. Here, it reports full convergence. R has found a best solution. If the model has not converged, results are likely not correct. This can happen when there are too many parameters in the model for not enough data.

Below, we see a table of basis checking results. This shows a statistical test for patterns in model residuals, which should be random. Each line reports the test results for one smooth. It shows the k value or number of basis functions, the effective degrees of freedom, a test statistic, and p-value.

Here, small p-values indicate that residuals are not randomly distributed. This often means there are not enough basis functions.

This is an approximate test. Always visualize your results too, and compare the k and edf values in addition to looking at the p-value.

gam.check(mod)

Method: REML Optimizer: outer newton

full convergence after 9 iterations.

Gradient range [-0.0001467222,0.00171085]

(score 784.6012 & scale 2.868607).

Hessian positive definite, eigenvalue range [0.00014,198.5]

Model rank = 7 / 7

Basis dimension (k) checking results. Low p-value

(k-index<1) may indicate that k is too low, especially

if edf is close to k'.

k' edf k-index p-value

s(x1) 3.00 1.00 0.35 <2e-16 \*\*\*

s(x2) 3.00 2.88 1.00 0.52

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1

If we re-fit our model with higher k , we see that this test is no longer significant. However, now we see a problem with the second smooth - the p-value for its test is now significant. Fixing one problem can reveal another. So it is always important to re-run gam.check after changing models.

mod <- gam(y ~ s(x1, k = 12) + s(x2, k = 4),data = dat, method = "REML")

gam.check(mod)

...

k' edf k-index p-value

s(x1) 11.00 10.85 1.05 0.830

s(x2) 3.00 2.98 0.89 0.015 \*

If we increase the k value for the second smooth, and run gam.check() again, now both smooths pass the test. Neither has significant patterns in their residuals and both have enough basis functions.

mod <- gam(y ~ s(x1, k = 12) + s(x2, k = 12),data = dat, method = "REML")

gam.check(mod)

...

k' edf k-index p-value

s(x1) 11.00 10.86 1.08 0.94

s(x2) 11.00 7.78 0.94 0.12

gam.check() will also generate four plots. Each of these gives a different way of looking at your model residuals. These plots show the results from the original, poorly fit model. On the top-left is a Q-Q plot, which compares the model residuals to a normal distribution. A well-fit model's residuals will be close to a straight line. On bottom left is a histogram of residuals. We would expect this to have a symmetrical bell shape. On top-right is a plot of residual values. These should be evenly distributed around zero. Finally, on the bottom-right is plot of response against fitted values. A perfect model would form a straight line. We don't expect a perfect model, but we do expect the pattern to cluster around the 1-to-1 line.

A picture containing diagram

Description automatically generated

Now, here is the output of our final model, with larger k values. See that the Q-Q plot no longer curves, the histogram is bell shaped, and the comparison of response vs. fitted values clusters around a 1-to-1 line. These all indicate a much better model fit.

A picture containing chart

Description automatically generated

You may recall the concept of collinearity from a linear modeling course. When two variables or covariates in a model are strongly correlated, it's difficult to fit the model, because the outcome variable could be responding to either one. We call this phenomenon collinearity, and it can result in poorly fit models with large confidence intervals. In general, we avoid putting multiple collinear variables into the same model.

Here, we look at correlation between three covariates in our mpg data set: the length, width, and height of cars. We can see that there is strong collinearity between car length and width. It would be hard to distinguish between their effects on car efficiency.

Scatter chart

Description automatically generated

With GAMs, we have an additional potential pitfall. Even if two variables aren't collinear, they may have concurvity, that is, one may be a smooth curve of another. For instance, on the left, we have two covariates, X1 and X2, that are not linearly related but form a perfect parabola. If we use both X1 and X2 as predictors in a model, we get smooths with wild confidence intervals, as shown in the middle and right plots.

Diagram, histogram

Description automatically generated

mgcv's concurvity() function measures concurvity in model variables. Like gam.check(), we run this function on a model object to examine the quality of our model.

concurvity() has two modes.

The first mode, full = TRUE, reports overall concurvity for each smooth. Specifically, it shows how much each smooth is predetermined by all the other smooths.

Since concurvity is complex, the function reports three different ways of measuring concurvity. Each is better in some situations. What is important is that you should always look at the worst case, and if the value is high (say, over 0.8), inspect your model more carefully.

Here I show the output of running the concurvity() function on a model with variables that are related but not perfectly. The concurvity of the terms is high in the worst case, so we'll want to inspect the plots of our model closely and be careful in making interpretations.

Chart, scatter chart

Description automatically generated

concurvity(m1, full = TRUE)

para s(X1) s(X2)

worst 0 0.84 0.84

observed 0 0.22 0.57

estimate 0 0.28 0.60

If any of these values from the full = TRUE mode is high, we will want to also use the second mode, setting full = FALSE. With full = FALSE, the function returns matrices of pairwise concurvities. These show the degree to which each variable is predetermined by each other variable, rather than all the other variables. This can be used to pinpoint which variables have a close relationship. Once again, the function returns three measures, this time as three matrices. Look for the worst-case scenario and see if variables with high values have problematic shapes or confidence intervals.

concurvity(model, full = FALSE)

$worst

para s(X1) s(X2)

para 1 0.00 0.00

s(X1) 0 1.00 0.84

s(X2) 0 0.84 1.00

$observed | $estimate

para s(X1) s(X2) | para s(X1) s(X2)

para 1 0.00 0.00 | para 1 0.00 0.0

s(X1) 0 1.00 0.57 | s(X1) 0 1.00 0.6

s(X2) 0 0.22 1.00 | s(X2) 0 0.28 1.0

library(gamair)

set.seed(0)

data("mpg", package="gamair", verbose=FALSE)

library(mgcv)

# Fit the model

mod <- gam(hw.mpg ~ s(length) + s(width) + s(height) + s(weight), data = mpg, method = "REML")

# Check overall concurvity

concurvity(mod, full=TRUE)

Loading required package: nlme

This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.

|  | **para** | **s(length)** | **s(width)** | **s(height)** | **s(weight)** |
| --- | --- | --- | --- | --- | --- |
| **worst** | 1.075259e-20 | 0.9303404 | 0.9322887 | 0.6723705 | 0.9603887 |
| **observed** | 1.075259e-20 | 0.7534619 | 0.8757513 | 0.4869308 | 0.8793300 |
| **estimate** | 1.075259e-20 | 0.8353324 | 0.7943374 | 0.4452676 | 0.8567519 |

height has relatively low concurvity. It isn’t too similar to any of the other variables.

# Check pairwise concurvity

concurvity(mod, full=FALSE)

Loading required package: nlme

This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.

$worst

|  | **para** | **s(length)** | **s(width)** | **s(height)** | **s(weight)** |
| --- | --- | --- | --- | --- | --- |
| **para** | 1.000000e+00 | 8.371907e-26 | 5.426323e-21 | 4.917497e-23 | 4.885837e-25 |
| **s(length)** | 8.379039e-26 | 1.000000e+00 | 8.336513e-01 | 6.058015e-01 | 8.797217e-01 |
| **s(width)** | 5.426335e-21 | 8.336513e-01 | 1.000000e+00 | 4.099837e-01 | 8.953662e-01 |
| **s(height)** | 4.919296e-23 | 6.058015e-01 | 4.099837e-01 | 1.000000e+00 | 3.665831e-01 |
| **s(weight)** | 4.908155e-25 | 8.797217e-01 | 8.953662e-01 | 3.665831e-01 | 1.000000e+00 |

$observed

|  | **para** | **s(length)** | **s(width)** | **s(height)** | **s(weight)** |
| --- | --- | --- | --- | --- | --- |
| **para** | 1.000000e+00 | 1.332911e-30 | 1.207487e-33 | 9.421332e-33 | 4.025643e-31 |
| **s(length)** | 8.379039e-26 | 1.000000e+00 | 7.511142e-01 | 2.827977e-01 | 8.232449e-01 |
| **s(width)** | 5.426335e-21 | 5.077384e-01 | 1.000000e+00 | 1.186126e-01 | 7.813743e-01 |
| **s(height)** | 4.919296e-23 | 2.284116e-01 | 3.313152e-01 | 1.000000e+00 | 2.900361e-01 |
| **s(weight)** | 4.908155e-25 | 6.052819e-01 | 7.863555e-01 | 1.494913e-01 | 1.000000e+00 |

$estimate

|  | **para** | **s(length)** | **s(width)** | **s(height)** | **s(weight)** |
| --- | --- | --- | --- | --- | --- |
| **para** | 1.000000e+00 | 3.264434e-28 | 1.729333e-23 | 3.441292e-25 | 2.185441e-27 |
| **s(length)** | 8.379039e-26 | 1.000000e+00 | 6.415191e-01 | 2.271285e-01 | 7.209033e-01 |
| **s(width)** | 5.426335e-21 | 6.477497e-01 | 1.000000e+00 | 1.054762e-01 | 7.241891e-01 |
| **s(height)** | 4.919296e-23 | 3.303484e-01 | 2.644827e-01 | 1.000000e+00 | 2.669300e-01 |
| **s(weight)** | 4.908155e-25 | 7.235198e-01 | 6.913221e-01 | 1.390568e-01 | 1.000000e+00 |

weight and width have worst-case concurvity of about 0.895.

# Dimensional Smooths and Spatial Data

Up until now, we have been working with models made up of one or several smooths, each of a single variable. Now, we will expand our models to include smooths of multiple variables and their interactions. This will allow us to look at new kinds of data, especially geospatial data, which are best represented by complex surfaces rather than single smooth lines.

You may be familiar with the concept of interactions from linear modeling. Interactions in models represent the fact that outcomes depend on non-independent relationships of multiple variables. In a linear model, they are generally represented by adding a term multiplying two variables. This can result in the outcome being higher or lower than what would be predicted by the sum of the two values alone.

Text, letter

Description automatically generated

In a GAM, the relationship between a variable and an outcome changes across the range of the smooth. Similarly, interactions are different across all the values of two or more variables. We represent interactions between variables as a smooth surface, so any combination of variables can take a different value. This is also a natural way to represent spatial data.

Chart, surface chart

Description automatically generated

The syntax for interactions in GAMs is straightforward. To model the interaction between two variables, we put two variables inside the s() function in a GAM formula, as shown here.

gam(y ~ s(x1, x2), data = dat, method = "REML") # <-- 2 variables

ou can mix interactions with other terms, which can be linear or nonlinear. For instance, the first formula here has an additional nonlinear term, x3, which is separate from the interaction of terms x1 and x2. The second formula has linear terms x3 and x4. Just as in our previous GAMs, you can include discrete, categorical terms along with interactions and linear terms.

A common way to model geospatial data is to use an interaction term of x and y coordinates, along with individual terms for other predictors. The interaction term then accounts for the spatial structure of the data.

gam(y ~ s(x1, x2) + s(x3), data = dat, method = "REML")

gam(y ~ s(x1, x2) + x3 + x4,data = dat, method = "REML")

When you look at the summary outputs of a model with interactions, you'll see that the interaction is a single smooth term. This combines the effects of x1, x2, and their combination in a single smooth. This differs from what you may expect in a linear model, where terms for x1, x2, and their combination are separate. We will discuss how to fit a model that separates these components later in this chapter.

Also note the high EDF, that is, effective degrees of freedom for this term. It takes many more basis functions, and therefore more data, to build a two-dimensional surface rather than a one-dimensional line

Family: gaussian

Link function: identity

Formula:

y ~ s(x1, x2)

Parametric coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 0.34256 0.01646 20.82 <2e-16 \*\*\*

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:

edf Ref.df F p-value

s(x1,x2) 10.82 14.9 14.37 <2e-16 \*\*\* #<-- Interaction

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) = 0.519 Deviance explained = 54.5%

GCV = 0.057564 Scale est. = 0.054161 n = 200

Interactions in GAMs are powerful tools for modeling complex or spatial data. However, their complexity makes them challenging to understand. Visualizing interactions helps us unlock understanding of these complexities. In this lesson we will learn mgcv's tools for visualizing interactions in two and three dimensions.

In mgcv's plot() command, interactions are represented with contour plots. A single plot represents both variables and their interaction. In this plot the axes represent values of our predictor variables, x1 and x2. The interior is a topographic map of predicted values. The contour lines represent points of equal predicted values, and they are labeled. The dotted lines show uncertainty in prediction; they represent how contour lines would move if predictions were one standard error higher or lower.

Chart, scatter chart

Description automatically generated

A contour plot is not always the most intuitive way of plotting interactions, so mgcv has a couple of more options. Setting the scheme parameter in the plot function to 1, we get a 3D perspective plot instead.

plot(mod\_2d, scheme = 1)

A picture containing chart

Description automatically generated

Setting scheme to 2 generates a heat map. This is a simplified contour map on top of which colors are added. The yellow colors represent larger predictions and the red colors smaller ones.

plot(mod\_2d, scheme = 2)

Chart

Description automatically generated

These pre-defined schemes are a nice way to view your model interactions quickly. However, we often want to customize these plots. For this, we can use mgcv's vis.gam(). vis.gam() has MANY options. We will focus on a few important ones.

vis.gam(x,

view = NULL,

cond = list(),

n.grid = 30,

too.far = 0,

col = NA,

color = "heat",

contour.col = NULL,

se = -1,

type = "link",

plot.type = "persp",

zlim = NULL,

nCol = 50,

...)

The first argument to vis.gam(), x, is the GAM model. The second, view, is where you list which variables in the model you want to visualize jointly.

Setting the plot.type argument to "persp" will produce a 3D perspective plot as shown here.

In this case, x1 and x2 are interacting variables, but they do not need to be. You can view a perspective plot of any two variables in the model to see their combined effects.

vis.gam(x = mod, # GAM object

view = c("x1", "x2"), # variables

plot.type = "persp") # kind of plot

Chart, surface chart

Description automatically generated

Or you can set plot.type to "contour", which will produce a contour plot or heat map.

vis.gam(x = mod, # GAM object

view = c("x1", "x2"), # variables

plot.type = "contour") # kind of plot

Chart

Description automatically generated

The too.far argument is an important one in using these plots to inspect your model. too.far indicates what predictions should not be plotted because they are too far from the actual data. In other words, how far is too far to extrapolate? Setting this value lets you see what combinations of variables are not represented in your data and therefore might not yield good predictions in your model.

too.far is scaled from zero to one along the range of the variables. Here, we set it at 0.1 for the left plot and 0.05 on the right. On the left, 10% extrapolation fills in most of the surface. On the right, 5% extrapolation shows more areas not supported by data.

vis.gam(mod, view = c("x1", "x2"), plot.type = "contour", too.far = 0.1)

vis.gam(mod, view = c("x1", "x2"), plot.type = "contour", too.far = 0.05)

Chart

Description automatically generated

For perspective plots, one can display confidence intervals of the predictions with the "se" argument. se takes a number which is the number of standard errors away from the average prediction to plot high- and low-prediction surfaces.

vis.gam(x = mod, view = c("x1", "x2"),

plot.type = "persp", se = 2)

Chart, diagram

Description automatically generated

You can also control the rotation angle of your perspective plots. The theta parameter controls horizontal rotation, the phi parameter controls vertical rotation, and the r parameter controls how zoomed in the plot is. Plots with low r values have lots of distortion or parallax, while plots with high r values have little sense of perspective.

vis.gam(g, view = c("x1", "x2"), plot.type = "persp", theta = 220)

vis.gam(g, view = c("x1", "x2"), plot.type = "persp", phi = 55)

vis.gam(g, view = c("x1", "x2"), plot.type = "persp", r = 0.1)

A picture containing shape

Description automatically generated

When making contour plots, color and contrast are important for clarity. Three important additional options for contours shown here are the color parameter, which selects the background color palette, the contour.col parameter, which selects the color of lines, and the nlevels option, which lets you control the number of contour lines - important for showing details and subtleties of interactions.

vis.gam(g, view = c("x1", "x2"), plot.type = "contour", color = "gray")

vis.gam(g, view = c("x1", "x2"), plot.type = "contour", contour.col = "blue")

vis.gam(g, view = c("x1", "x2"), plot.type = "contour", nlevels = 20)

Chart, histogram

Description automatically generated