Interpreting & Predicting from GAMs

Fitting Models to Data, Not Data to Models
Model Fitting Series - With Applications in R

Jesse Ghashti
Source code by Stefano Mezzini

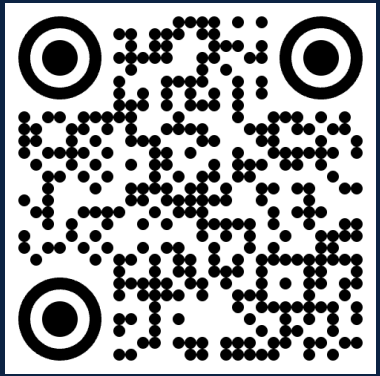
October 29, 2025

Centre for Scholarly Communication
The University of British Columbia | Okanagan Campus | Syilx Okanagan Nation Territory

Workshop Series Overview

Session	Topic	Date/Time
1	Simple Linear Regression	Oct 7, 9:00 AM
2	Fitting Linear Models in R	Oct 8, 10:30 AM
3	Multiple Linear Regression in R	Oct 16, 4:00 PM
4	Interaction Terms & Hierarchical Linear Models	Oct 21, 11:00 AM
5	Generalized Linear Models	Oct 23, 4:00 PM
6	Generalized Additive Models (GAMs)	Oct 28, 11:00 AM
7	Interpreting & Predicting from GAMs	Oct 29, 10:30 AM
8	Hierarchical GAMs	Nov 4, 12:00 PM
9	Penalized Models	Nov 18, 11:00 AM
10	Survival Models	Nov 25, 11:00 AM
11	Nonparametric Models	Dec 2, 11:00 AM

New Here?

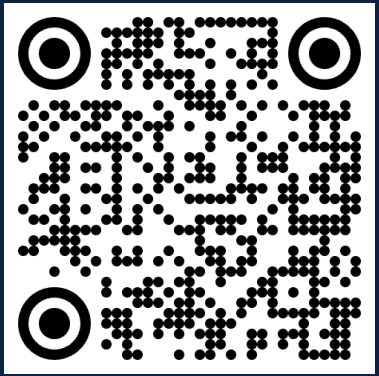


<———————-

New to R? Check out the Fundamentals of R series!

GitHub code and slides for today's workshop (and previous workshops)

———————————>



Alternatively, code/slides available at the bottom of `https://csc-ubc-okanagan.github.io/workshops/`

Last Time (Workshop 6) — Quick Recap

Key Concepts

- Learned about GAMs: replacing linear terms with smooth functions
- Used `s(x)` for smooths, `s(x, bs = 'fs')` for factor smooths
- Built hierarchical GAMs: common smooth + group deviations
- Visualized smooths with `gratia::draw()`
- Compared LM, GLM, and GAM predictions

Today: Interpreting & Predicting from GAMs

Today we will...

- Compare different GAM families (Gamma vs Tweedie)
- Interpret smooths on link vs response scales
- Make predictions for population vs individuals
- Compare models using AIC, BIC, and deviance explained
- Understand when to use `discrete = TRUE` vs `FALSE`
- Visualize uncertainty in predictions

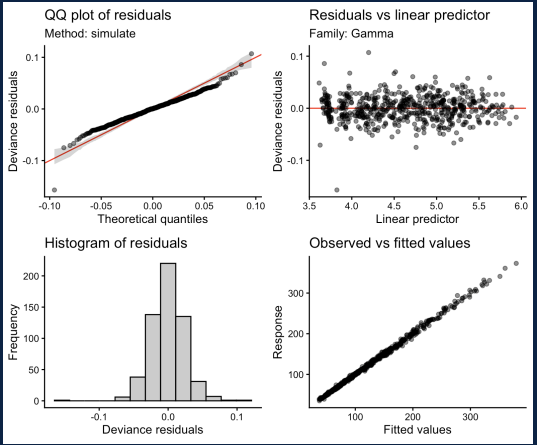
Today we require...

```
library('dplyr')    # for data wrangling
library('tidyr')    # for expand_grid
library('mgcv')     # for modeling
library('ggplot2')  # for fancy plots
library('gratia')   # for ggplot-based model graphics
theme_set(theme_classic(base_size = 15))
```

Review: Gamma GAM from Last Week

```r
# Gamma GAM (fast)
m_gamma <- bam(
  formula = weight ~
    Diet +
    s(Time, by = Diet, k = 5) +
    s(Time, Chick, bs = 'fs', k = 5),
  family = Gamma(link = 'log'),
  data = ChickWeight,
  method = 'fREML',
  discrete = TRUE,
  control = gam.control(trace = TRUE))

appraise(m_gamma, method = 'simulate')
```

Note that `by = Diet` creates diet-specific smooths (no shared common smooth)



What is the Tweedie Distribution?

Tweedie Family Properties

Flexibility:

- Generalization of distributions via power parameter p
- $1 < p < 2$: Compound Poisson-Gamma (allows zeros)
- $p = 1$: is Poisson, and $p = 2$: Gamma

Why use Tweedie?

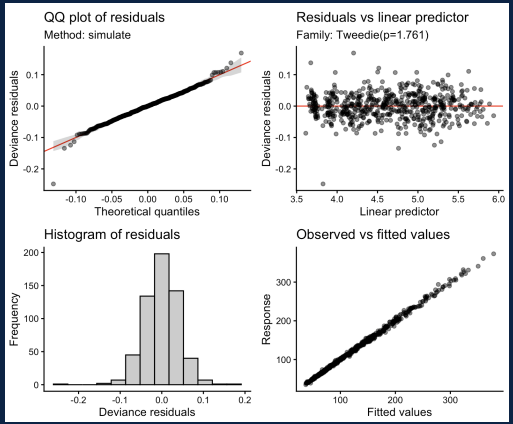
- More flexible variance-mean relationship
- `tw()` estimates p from the data
- Often fits better than Gamma for positive continuous data

Trade-off is more flexibility but computationally slower than Gamma

Fitting a Tweedie GAM

```r
# Tweedie GAM (slower but more flexible)
m_tw <- bam(
  formula = weight ~
    Diet +
    s(Time, by = Diet, k = 5) +
    s(Time, Chick, bs = 'fs', k = 5),
  family = tw(link = 'log'),  # Tweedie family with power parameter
  data = ChickWeight,
  method = 'fREML',
  discrete = TRUE,
  control = gam.control(trace = TRUE))

appraise(m_tw, method = 'simulate')
```

`tw()` automatically estimates the power parameter p to best fit the variance structure



Two Model Summary

```
weight ~ Diet + s(Time, by = Diet, k = 5) + s(Time, Chick, bs = "fs",
    k = 5)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.44861    0.03085 144.223  < 2e-16 ***
Diet2        0.16484    0.05246   3.142  0.00182 **
Diet3        0.28367    0.05246   5.408 1.17e-07 ***
Diet4        0.28533    0.05247   5.438 9.99e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
                   edf  Ref.df      F p-value
s(Time):Diet1    3.472   3.555  32.37  <2e-16 ***
s(Time):Diet2    3.261   3.360  24.53  <2e-16 ***
s(Time):Diet3    2.955   3.072  35.21  <2e-16 ***
s(Time):Diet4    3.709   3.760  29.78  <2e-16 ***
s(Time,Chick) 203.070 239.000 111.07  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.997   Deviance explained = 99.8%
fREML =   -724  Scale est. = 0.0009371  n = 578
```

```
weight ~ Diet + s(Time, by = Diet, k = 5) + s(Time, Chick, bs = "fs",
    k = 5)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.44764    0.03098 143.570  < 2e-16 ***
Diet2        0.16589    0.05270   3.148  0.00179 **
Diet3        0.28470    0.05270   5.402 1.22e-07 ***
Diet4        0.28630    0.05271   5.432 1.05e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
                   edf  Ref.df      F p-value
s(Time):Diet1    3.447   3.502  33.24  <2e-16 ***
s(Time):Diet2    3.224   3.286  24.85  <2e-16 ***
s(Time):Diet3    2.946   3.018  35.44  <2e-16 ***
s(Time):Diet4    3.703   3.735  30.00  <2e-16 ***
s(Time,Chick) 211.964 240.000 200.69  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.997   Deviance explained = 99.8%
fREML = -379.4  Scale est. = 0.0016683  n = 578
```

Comparing GAMs: Which Model is Best?

Model Selection Criteria for GAMs

Information Criteria (penalize complexity):

- **AIC:** Akaike Information Criterion — lower is better
- **BIC:** Bayesian Information Criterion — penalizes complexity more
- Rule: $|\Delta\text{AIC}| < 2$ suggests models are similar

Goodness of Fit:

- **Deviance Explained:** Proportion of null deviance explained (like R^2)
- **Adjusted R^2:** Accounts for effective degrees of freedom (edf)

Predictive Performance:

- **RMSE:** Root Mean Squared Error on response scale
- **MAE:** Mean Absolute Error — robust to outliers

Model Comparison: Gamma vs Tweedie

```r
AIC(m_gamma, m_tw)
BIC(m_gamma, m_tw)

summary(m_gamma)$dev.expl
summary(m_tw)$dev.expl

summary(m_gamma)$r.sq
summary(m_tw)$r.sq

# RMSE gamma
sqrt(mean((fitted(m_gamma) - ChickWeight$weight)^2))
# RMSE tw
sqrt(mean((fitted(m_tw) - ChickWeight$weight)^2))
```

```
> AIC(m_gamma, m_tw)
             df      AIC
m_gamma 222.1197 3194.142
m_tw    231.6524 3125.490
> BIC(m_gamma, m_tw)
             df      BIC
m_gamma 222.1197 4162.489
m_tw    231.6524 4135.396
> summary(m_gamma)$dev.expl
[1] 0.9982569
> summary(m_tw)$dev.expl
[1] 0.9983795
> summary(m_gamma)$r.sq
[1] 0.9968351
> summary(m_tw)$r.sq
[1] 0.996928
> # RMSE gamma
> sqrt(mean((fitted(m_gamma)
[1] 3.144859
> # RMSE tw
> sqrt(mean((fitted(m_tw) -
[1] 3.059771
```

Model Comparison Results

Metric	Gamma GAM	Tweedie GAM
AIC	3194.1	3125.5
BIC	4162.5	4135.4
Deviance Explained	99.83%	99.84%
Adjusted R^2	0.9968	0.9969
RMSE	3.14 g	3.06 g
Power parameter (p)	2 (fixed)	1.761 (estimated)

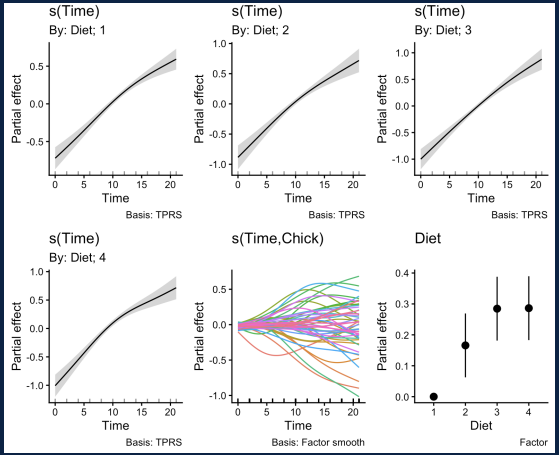
So... what do we think?

Visualizing Smooths: Link Scale

```
# Terms on LOG link scale
# Centered at 0 (average)
# Additive effects
draw(m_tw,
     scales = 'free',
     parametric = TRUE)
```

Link scale:

- y-axis: $\log(\mu)$
- Effects are additive
- Centred at o
- Easier for testing

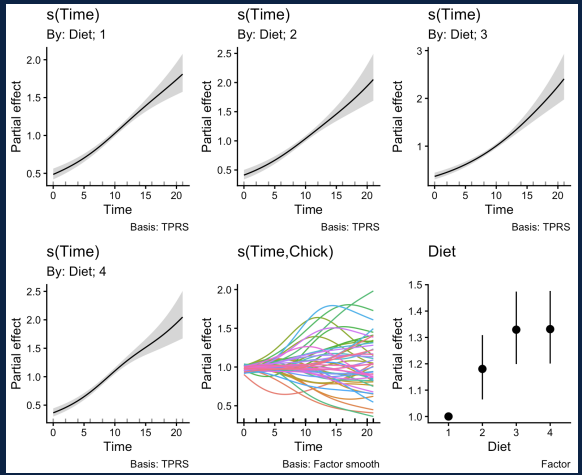


Visualizing Smooths: Response Scale

```r
# Terms on RESPONSE scale
# Centered at 1 (average)
# Multiplicative effects
draw(m_tw,
     scales = 'free',
     parametric = TRUE,
     fun = exp)
```

Response scale:

- y-axis: μ (weight in grams)
- Effects are multiplicative
- Centred at 1
- Easier interpretation



Link vs Response Scale: When to Use Each?

Link Scale

Use for:

- Statistical testing
- Model building
- Comparing effect sizes
- Constructing CIs

Interpretation:

- Additive effects
- Linear combinations
- Symmetry

Response Scale

Use for:

- Communication
- Visualization
- Subject-matter interpretation
- Practical decisions

Interpretation:

- Multiplicative effects
- Actual units (grams)
- Intuitive

Remember the best practice of working on link scale, present on response scale

Setting Up Prediction Data

```r
# new dataset for predictions
newd <- expand_grid(
  Time = seq(0, 21, length.out = 400),
  Diet = unique(ChickWeight$Diet),
  Chick = ChickWeight$Chick[1])          # Placeholder for chick
# This gives us 400 × 4 = 1600 prediction points
dim(newd)
```

Why Chick = ChickWeight$Chick[1]?

- We need *some* chick ID for the random effect structure
- We'll exclude it later for population-level predictions
- Or keep it for individual-level predictions

Population-Level Predictions

```r
# predictions with 95% credible intervals (assuming Gaussian posterior on log scale)
preds <-
  bind_cols(
    newd,
    predict(object = m_tw,
            newdata = newd,
            type = 'link',              # Predict on log scale
            se.fit = TRUE,              # Include standard errors
            discrete = TRUE,            # Fast (only for levels in data)
            exclude = c('s(Time,Chick)')) %>%  # EXCLUDE chick effects
      as.data.frame() %>%
      mutate(mu_hat = exp(fit),
             lwr_95 = exp(fit - 1.96 * se.fit),  # CI on link, transform
             upr_95 = exp(fit + 1.96 * se.fit)))
preds
```

`exclude = c('s(Time,Chick)')` gives predictions for an "average" chick

Understanding `discrete = TRUE` vs `FALSE`

When to Use Each

`discrete = TRUE` **(fast):**

- Use when predicting for random effect levels *in the dataset*
- Example: Chick '1', Chick '2', etc.
- Much faster computation
- Required if you used `discrete = TRUE` in `bam()`

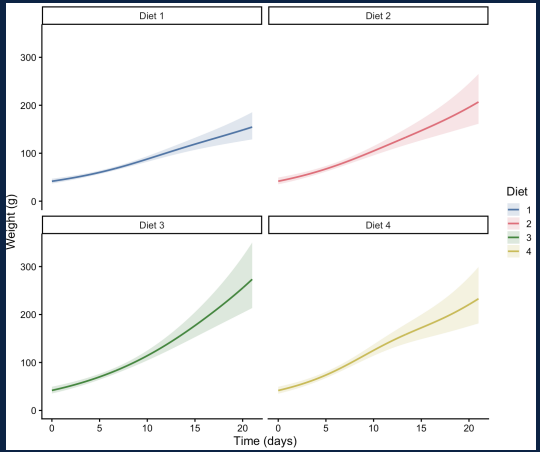
`discrete = FALSE` **(slower):**

- Use when predicting for *new* random effect levels
- Example: Chick 'new chick' (not in original data)
- Slower but necessary for new levels
- Also use when `exclude` parameter is needed with new levels

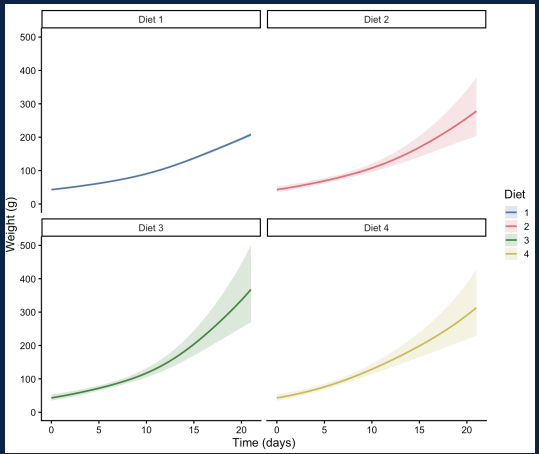
If using `exclude` with a new random effect level, must use `discrete = FALSE`

Visualizing Population-Level Predictions

```
# Plot together
p_mean <- ggplot(preds) +
  geom_ribbon(aes(Time,
                  ymin = lwr_95,
                  ymax = upr_95,
                  fill = Diet),
              alpha = 0.2) +
  geom_line(aes(Time, mu_hat,
                color = Diet),
            lwd = 1) +
  xlab('Time (days)') +
  scale_y_continuous('Weight (g)',
                     limits = c(0, NA)) +
  khroma::scale_color_bright() +
  khroma::scale_fill_bright()
p_mean

# Faceted by diet
p_mean +
  facet_wrap(~ paste('Diet', Diet))
```



Population Predictions by Diet



These represent the expected growth trajectory for an "average" chick on each diet

Individual-Level Predictions (Chick 1)

```
# Check which diet Chick 1 ate
ChickWeight[1, ]  # Chick 1 ate Diet 1

# Predictions for Chick 1 specifically
preds_1 <-
  bind_cols(
    mutate(newd, Chick = '1'),
    predict(object = m_tw,
            newdata = mutate(newd, Chick = '1'),
            type = 'link',
            se.fit = TRUE,
            discrete = TRUE) %>%  # DON'T exclude s(Time,Chick) now!
      as.data.frame() %>%
      mutate(mu_hat = exp(fit),
             lwr_95 = exp(fit - 1.96 * se.fit),
             upr_95 = exp(fit + 1.96 * se.fit)))
preds_1
```

Notice no `exclude` — we *want* Chick 1's individual effect

Why Individual Predictions Differ

Population vs Individual

Population-level (exclude random effects):

- Represents the "average" individual
- Wider confidence intervals (between-individual variation)
- Shows the fixed effect structure only

Individual-level (include random effects):

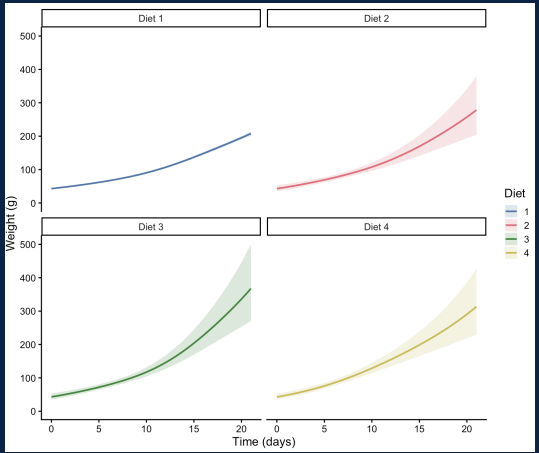
- Represents a specific individual's trajectory
- Narrower CIs (within-individual variation only)
- Incorporates individual deviations from average
- CIs narrower for observed diet (e.g., Chick 1 ate Diet 1)

Interpreting the difference: if we know which chick we're measuring, we can predict more precisely.

Visualizing Individual Predictions (Chick 1)

```
# Plot for Chick 1
p_1 <- ggplot(preds_1) +
  geom_ribbon(aes(Time,
                  ymin = lwr_95,
                  ymax = upr_95,
                  fill = Diet),
              alpha = 0.2) +
  geom_line(aes(Time, mu_hat,
                color = Diet),
            lwd = 1) +
  xlab('Time (days)') +
  scale_y_continuous('Weight (g)',
                     limits = c(0, NA)) +
  khroma::scale_color_bright() +
  khroma::scale_fill_bright()
p_1 + facet_wrap(~ paste('Diet', Diet))
```

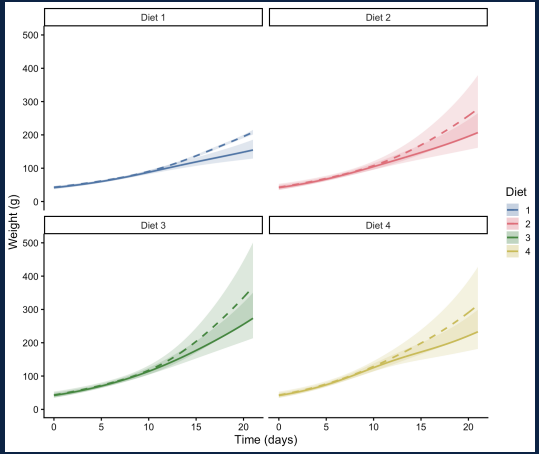
Notice that CIs are narrower for Diet 1 (Chick 1's actual diet)



Comparing Population vs Individual Predictions

```
# Overlay population and individual
p_mean +
  facet_wrap(~ paste('Diet', Diet)) +
  # Add Chick 1 predictions
  geom_ribbon(aes(Time,
                  ymin = lwr_95,
                  ymax = upr_95,
                  fill = Diet),
              preds_1, alpha = 0.2) +
  geom_line(aes(Time, mu_hat,
                color = Diet),
            preds_1, lwd = 1, linetype = "dashed")
```

Population (wider bands) vs Individual (narrower bands)



Understanding Prediction Uncertainty

Sources of Uncertainty

In our confidence intervals:

- **Parameter uncertainty:** Uncertainty in βs and smooth functions
- Captured by `se.fit` from `predict()`
- Assumes Gaussian distribution on link scale

Not included (but could be):

- **Residual variance:** Natural variation around fitted mean
- Would create *prediction intervals* (much wider than CIs)
- Use `type = 'response'` with simulation for full uncertainty

We showed uncertainty in the *mean* trajectory, not individual observations

Confidence Intervals vs Prediction Intervals

Aspect	Confidence Interval	Prediction Interval
What it captures	Uncertainty in the *mean*	Uncertainty for a *new observation*
Width	Narrower	Wider
Includes	Parameter uncertainty	Parameter + residual variance
Interpretation	"Where is the true mean?"	"Where will a new value fall?"
Use case	Model comparison, inference	Forecasting, planning

Best Practices for GAM Predictions

Do's

- Always predict on the link scale, then transform
- Compute confidence intervals on link scale before transforming
- Use `discrete = TRUE` when possible for speed
- Visualize predictions with uncertainty bands
- Check if predictions are in reasonable range
- Use `exclude` for population-level, omit for individual-level

Don'ts

- Don't transform standard errors directly
- Don't extrapolate far beyond observed data range
- Don't forget about the random effect structure
- Don't ignore diagnostic plots before predicting

When Can GAM Predictions Mislead?

Common Pitfalls

Extrapolation:

- GAMs are data-driven — behaviour outside data range is unpredictable
- Smooths can curve wildly beyond observed values

Boundary effects:

- Predictions near edges of data can be unreliable
- Wider uncertainty at boundaries

Sparse regions:

- Few observations in some regions $\Rightarrow$ wider CIs
- Check data density before trusting predictions

Always check by plotting your data alongside predictions to see coverage.

Always Check Diagnostics Before Predicting

```
# diagnostic checks
# appraise(m_tw, method = 'simulate')  # Overall model fit
# gam.check(m_tw)                      # Basis dimensions adequate?
# summary(m_tw)                        # Significant terms?
# concurvity(m_tw)                     # Smooths too correlated?

# Comparative diagnostics
# AIC(m_gamma, m_tw)                   # Which model is better?
```

Bad fit = unreliable predictions

REVISITING TRAINING AND TESTING SETS

We compare Interaction GLMs with Gamma GAM

Validating Models: Train/Test Split

Why Split the Data?

The Problem:

- AIC/BIC computed on the same data used to fit the model
- Models can overfit — perform well on training data, poorly on new data
- Need to test predictive performance on *unseen* observations

The Solution:

- **Training set:** Fit the model (typically 70-80% of data)
- **Test set:** Evaluate predictions (remaining 20-30%)
- More honest assessment of model performance

Always remember that a good in-sample fit doesn't guarantee good out-of-sample predictions.

Creating Train/Test Split

```r
set.seed(2025)

uniqueCH <- unique(ChickWeight$Chick)
numCH <- length(uniqueCH)

# random assign 80% of chicks to training, 20% to test
trainCH <- sample(uniqueCH, size = floor(0.8 * numCH))
testCH <- setdiff(uniqueCH, trainCH)

# train and test datasets
trainDATA <- ChickWeight %>% filter(Chick %in% trainCH)
testDATA <- ChickWeight %>% filter(Chick %in% testCH)
```

Notice how we are splitting by *chick*, not by observation.

Fitting Models on Training Data

```r
# notice we got rid of s(Time, Chick, bs = 'fs', k = 5) when we predict on testDATA, some Chick values would have never been seen during training.
# Our goal is to predict new chicks (not seen in training), so we must exclude the Chick-specific smooth
gammaTRAIN <- bam(formula = weight ~ Diet + s(Time, by = Diet, k = 5), family = Gamma(link = 'log'), data = trainDATA, method = 'fREML', discrete = TRUE)
tweedieTRAIN <- bam(formula = weight ~ Diet + s(Time, by = Diet, k = 5), family = tw(link = 'log'), data = trainDATA, method = 'fREML', discrete = TRUE)
summary(gammaTRAIN)
summary(tweedieTRAIN)
```

```
Family: Gamma
Link function: log

Formula:
weight ~ Diet + Time:Diet

Parametric coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.755336   0.029844 125.834   <2e-16 ***
Diet2       0.049355   0.049068   1.006   0.3150
Diet3       0.036770   0.049068   0.749   0.4540
Diet4       0.106304   0.056269   1.889   0.0595 .
Diet1:Time  0.071762   0.002403  29.862   <2e-16 ***
Diet2:Time  0.081027   0.003031  26.729   <2e-16 ***
Diet3:Time  0.089897   0.003031  29.655   <2e-16 ***
Diet4:Time  0.083279   0.003713  22.431   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


R-sq.(adj) =   0.76   Deviance explained = 85.9%
-REML = -30.348  Scale est. = 0.045565  n = 461
```

```
Formula:
weight ~ Diet + s(Time, by = Diet, k = 5)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.51805    0.01561 289.412  < 2e-16 ***
Diet2        0.14791    0.02520   5.870 8.49e-09 ***
Diet3        0.22995    0.02519   9.130  < 2e-16 ***
Diet4        0.22771    0.02883   7.899 2.20e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
                edf Ref.df     F p-value
s(Time):Diet1 2.307  2.792 348.7  <2e-16 ***
s(Time):Diet2 2.466  2.967 265.8  <2e-16 ***
s(Time):Diet3 2.321  2.809 343.3  <2e-16 ***
s(Time):Diet4 2.556  3.062 183.5  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.776   Deviance explained = 87.1%
fREML = -35.077  Scale est. = 0.041849  n = 461
```

Making Predictions on Test Data

```r
predGLM <- predict(glmTRAIN, newdata = testDATA, type = 'response', discrete = FALSE)
predGAM <- predict(gamTRAIN, newdata = testDATA, type = 'response', discrete = FALSE)
# Calculate prediction errors
testDATA <- testDATA %>% mutate(pred_glm = predGLM,
         pred_gam = predGAM,
         error_glm = weight - pred_glm,
         error_gam = weight - pred_gam,
         abs_error_glm = abs(error_glm),
         abs_error_gam = abs(error_gam),
         sq_error_glm = error_glm^2,
         sq_error_gam = error_gam^2)
# Root Mean Squared Error (RMSE)
glmRMSE <- sqrt(mean(testDATA$sq_error_glm))
gamRMSE <- sqrt(mean(testDATA$sq_error_gam))
# Mean Absolute Error (MAE)
glmMAE <- mean(testDATA$abs_error_glm)
gamMAE <- mean(testDATA$abs_error_gam)
# Mean Absolute Percentage Error (MAPE)
glmMAPE <- mean(abs(testDATA$error_glm/testDATA$weight))*100
gamMAPE <- mean(abs(testDATA$error_gam/testDATA$weight))*100
tibble(
  Model = c("GLM", "GAM"),
  RMSE = c(glmRMSE, gamRMSE),
  MAE = c(glmMAE, tweedieMAE),
  MAPE = c(gammaMAPE, tweedieMAPE),
  AIC = c(AIC(glmTRAIN), AIC(gamTRAIN)),
  BIC = c(BIC(glmTRAIN), BIC(gamTRAIN)),
  ADJR = c(summary(glmTRAIN)$r.sq, summary(gamTRAIN)$r.sq))
```

Test Set Performance: GLM vs GAM

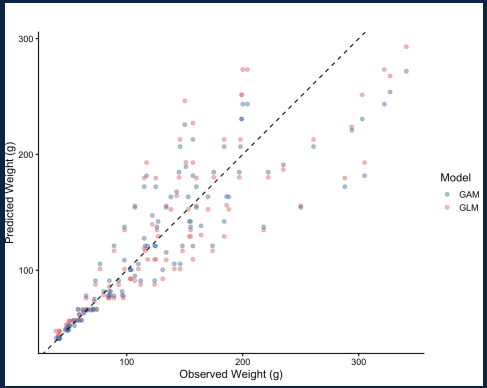
Model	RMSE (g)	MAE (g)	MAPE (%)	AIC	BIC	Adj. R^2
Interact. GLM	35.6	24.3	16.4%	4193	4230	0.760
Gamma GAM	34.0	22.0	14.1%	4167	4233	0.776
Best?						

GAM outperforms GLM on unseen data — lower prediction errors across all metrics

The more flexible variance structure of GAM generalizes better to new chicks.

Visualizing Test Set Predictions

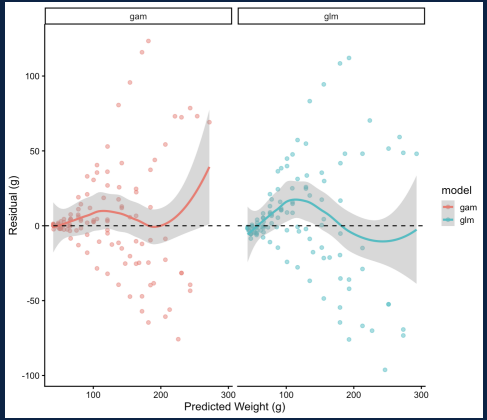
```
ggplot(testDATA) +
  geom_point(aes(weight, pred_glm,
                 color = "GLM"),
             alpha = 0.5) +
  geom_point(aes(weight, pred_gam,
                 color = "GAM"),
             alpha = 0.5) +
  geom_abline(slope = 1,
              intercept = 0,
              linetype = "dashed") +
  coord_equal() +
  labs(x = "Observed Weight (g)",
       y = "Predicted Weight (g)",
       color = "Model") +
  khroma::scale_color_bright()
```



Perfect predictions fall on the diagonal line

Test Set Residuals: Checking for Bias

```r
# residuals vs fitted
testDATA_long <- testDATA %>%
  pivot_longer(
    cols = c(error_glm, error_gam),
    names_to = "model",
    values_to = "residual",
    names_prefix = "error_") %>%
  pivot_longer(
    cols = c(pred_glm, pred_gam),
    names_to = "pred_model",
    values_to = "predicted",
    names_prefix = "pred_") %>%
  filter(substr(model, 1, 2) ==
           substr(pred_model, 1, 2))
ggplot(testDATA_long,
       aes(predicted, residual,
           color = model)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "loess") +
  geom_hline(yintercept = 0,
             linetype = "dashed") +
  facet_wrap(~ model) +
  labs(x = "Predicted Weight (g)",
       y = "Residual (g)")
```



Beyond Simple Train/Test: Cross-Validation

K-Fold Cross-Validation

Limitation of single split:

- Results depend on which chicks happened to be in test set
- Single estimate of performance — could be lucky or unlucky

Cross-validation solution:

- Split data into K folds (e.g., $K = 5$ or $K = 10$)
- Train on $K - 1$ folds, test on remaining fold
- Repeat K times, rotating which fold is the test set
- Average performance across all folds

Better estimate of out-of-sample performance, uses all data for both training and testing

GAM Prediction Workflow Summary

1. **Fit model:** Choose appropriate family and smooth structure
2. **Check diagnostics:** `appraise()`, `gam.check()`, `summary()`
3. **Compare models:** AIC, BIC, deviance explained, RMSE
4. **Create prediction data:** `expand_grid()` with appropriate structure
5. **Decide level:** Population (exclude RE) vs Individual (include RE)
6. **Set parameters:** `type = 'link'`, `se.fit = TRUE`, `discrete = TRUE/FALSE`
7. **Transform:** Apply inverse link function to fitted values and CIs
8. **Visualize:** Plot with uncertainty bands
9. **Interpret:** Consider what the predictions mean scientifically
10. **Validate:** Check predictions against held-out data if possible

You Try: Making and Comparing Predictions

Exercise: Predict and compare

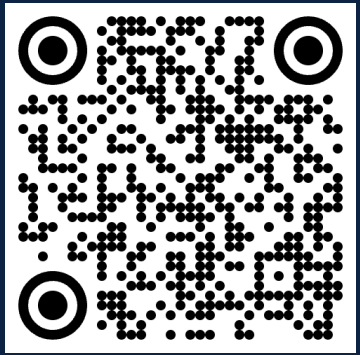
```r
# Use the Wage data from ISLR package
library(ISLR)
?Wage

# 1. Fit two models with different families
m1 <- gam(wage ~ s(age) + s(year) + education,
          data = Wage, family = Gamma(link = 'log'))
m2 <- gam(wage ~ s(age) + s(year) + education,
          data = Wage, family = tw(link = 'log'))

# 2. Compare models
# - Which has lower AIC/BIC?
# - What's the deviance explained?
# - Compute RMSE for both

# 3. Make predictions for education = "HS Grad", varying age
# - Create prediction grid
# - Get both population-level predictions
# - Plot with confidence intervals
# - Compare the two models visually

# 4. Interpret
# - Do the models give similar predictions?
# - Where do they differ most?
# - Which would you trust more and why?
```

Key Takeaways

- Tweedie family more flexible than Gamma, estimates power parameter
- Compare models with AIC, BIC, deviance explained, and RMSE
- Visualize smooths on link scale (testing) or response scale (interpretation)
- Always predict on link scale, then transform to response scale
- Population-level: `exclude` random effects for average trajectory
- Individual-level: Include random effects for specific predictions
- Use `discrete = TRUE` for speed when predicting existing levels
- Confidence intervals show uncertainty in *mean*, not individual observations
- Always check diagnostics before making predictions!
- Plot predictions with data to verify they're reasonable

What's Next?

Additional Questions?
Book an Appointment!



Next Workshop:

Hierarchical GAMs
–> November 4, 12:00 PM
–> More GAMs.

Thank You!

Questions?

Workshop Materials:
`https://github.com/csc-ubc-okanagan/ubco-csc-modeling-workshop`

Contact:
jesse.ghashti@ubc.ca