Class 6: Visualising statistical and machine learning model output.

Andrew Parnell
andrew.parnell@mu.ie

Maynooth University
National University of Ireland Maynooth

PRESS RECORD

https://andrewcparnell.github.io/dataviz_course

Learning outcomes

- Quick reminder on (generalised) linear models and machine learning
- Learn how to visualise output from (generalised) linear models using `ggfortify`
- Run some machine learning models using `tidymodels` and `mlr3`
- Plot some output from machine learning models using `iml` and `DALEX`

Generalised linear models (GLMs) in one slide

- In all univariate statistical models we have one variable we are trying to predict (the *response*), and multiple variables upon which to create that prediction (*features*)
- If the response is continuous and unbounded, most people use linear regression
- If the response is restricted in some way then people use a generalised linear model which models the transformed mean of the response as a linear regression

Machine learning in one slide

- Statistical models usually assume a linear relationship between the features and the response
- Machine learning models by contrast usually assume a non-linear relationship with interactions between the features
- The fitted values are usually a better fit to the data compared to those of a statistical regression model at the expense of model interpretability and uncertainty calibration
- Machine learning has its own jargon and techniques; for example models are usually compared on data that has been left out of the fitting process

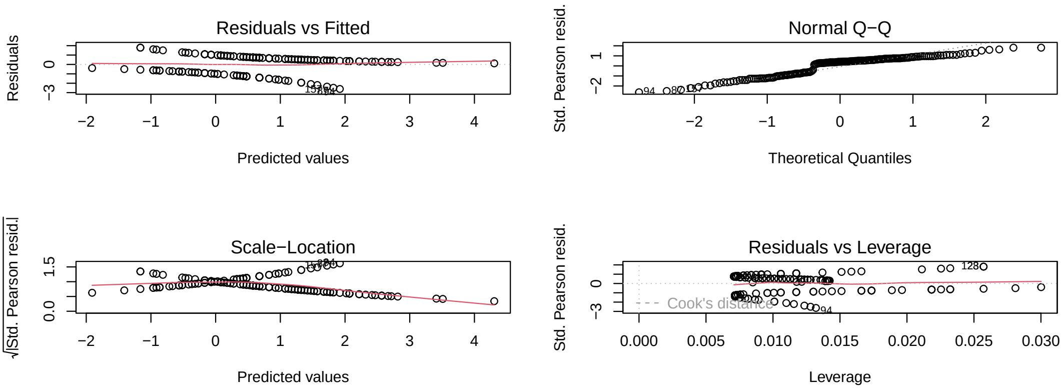
An example of a GLM fit

```r
horseshoe <- readRDS("../data/horseshoe.rds")
model <- glm(I(satell > 0) ~ width,
             family = binomial(link = 'logit'),
             data = horseshoe)
summary(model)
```

```
##
## Call:
## glm(formula = I(satell > 0) ~ width, family = binomial(link = "logit"),
##     data = horseshoe)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0281  -1.0458   0.5480   0.9066   1.6942
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -12.3508     2.6287  -4.698 2.62e-06 ***
## width         0.4972     0.1017   4.887 1.02e-06 ***
##
```

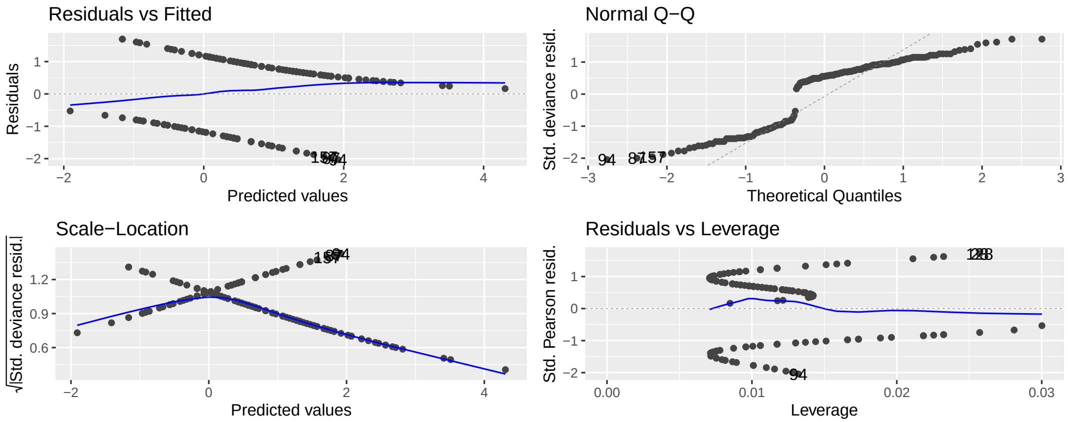
Default glm plots

```r
par(mfrow=c(2,2))
plot(model)
```



ggfortify again

```r
library(ggfortify)
autoplot(model)
```



Options for fitting a machine learning model in R

Lots of packages for fitting machine learning models in R. Some choices:

- `caret` is the original. Hundreds of different methods. Getting a bit old fashioned
- `tidymodels` in a tidyverse style set of packages for fitting machine learning models. Links well with `ggplot2`
- `mlr3` very nice extendible package with a large number of different modelling strategies and output plots

Most of these packages use **other packages** to perform the machine learning in the background

Once the model has been fitted...

- It is common to plot the feature importances, interactions and misclassification/error rates
- Plot individual variable performance using individual conditional expectation (ICE) curves and partial dependence plots (PDPs)
- (These can sometimes be tricky as the importance is conditional on other features)
- Once you have fitted the machine learning model there are lots of packages to compare the fit
- We will cover `tidymodels`, `mlr3`, `iml` and `DALEX` all briefly

Fitting a machine learning model using tidymodels

```r
library(tidymodels); library(ranger); library(palmerpenguins)

# Split the data into training and testing sets
set.seed(123)
penguins_split <- initial_split(penguins %>%
                                  na.omit(),
                                prop = 0.8)
penguins_train <- training(penguins_split)
penguins_test <- testing(penguins_split)

# Define the model specification
rf_spec <-
  rand_forest(trees = 1000) %>%
  set_engine("ranger") %>%
  set_mode("classification")
```

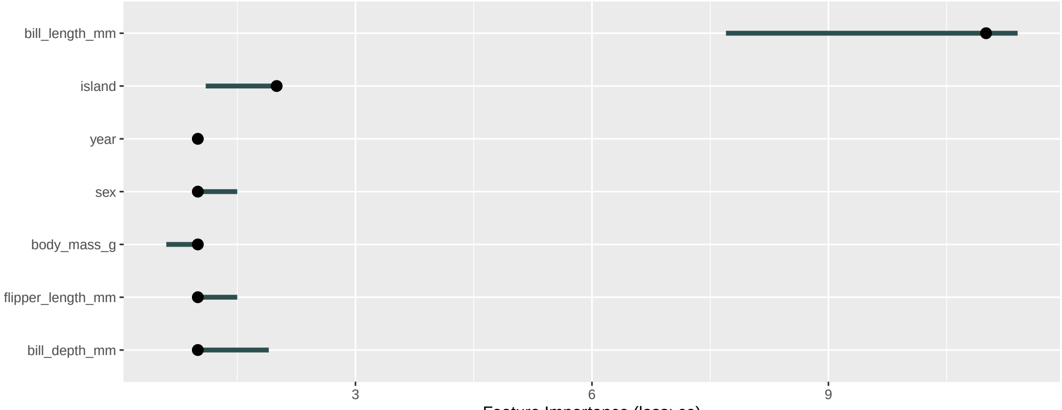
tidymodels part 2

```r
# Fit the model to the training data
rf_fit <- rf_spec %>% fit(species ~ .,
                          data = penguins_train)

# Make predictions on the test data
rf_preds <- rf_fit %>% predict(penguins_test)

# Evaluate the model performance
rf_preds %>%
  bind_cols(penguins_test) %>%
  dplyr::select(.pred_class, species) %>%
  table
```

```
##             species
## .pred_class Adelie Chinstrap Gentoo
##   Adelie        26         0      0
##   Chinstrap      2        15      0
##   Gentoo         0         0     24
```

iml - feature importance

```r
library(iml)
predictor <- Predictor$new(rf_fit, data = penguins_test[,-1],
                                   y = penguins_test[,1])
imp <- FeatureImp$new(predictor, loss = "ce") # Classification Error
plot(imp)
```

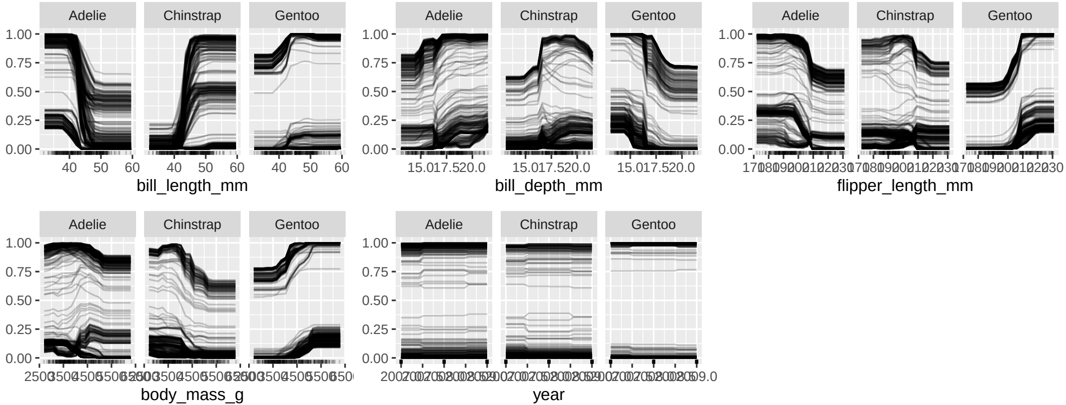


Another example - using mlr3

```r
library(mlr3)
library(mlr3learners)

# Create a task
penguins2 = na.omit(penguins)
task_peng = as_task_classif(species ~ .,
                            data = penguins2)
# Choose learner
learner = lrn("classif.ranger",
              predict_type = "prob")
# Split into training/test
split = partition(task_peng, ratio = 0.8)
# Train the learner
learner$train(task_peng, split$train_set)
# Predict on the test set
prediction = learner$predict(task_peng, split$test_set)
```

Feature effects: ICE and PDPs

```
num_features = c("bill_length_mm", "bill_depth_mm", "flipper_length_mm", "
model = Predictor$new(learner, data = penguins2[,-1],
                      y = penguins2$species)
effect = FeatureEffects$new(model, method = 'ice')
plot(effect, features = num_features)
```

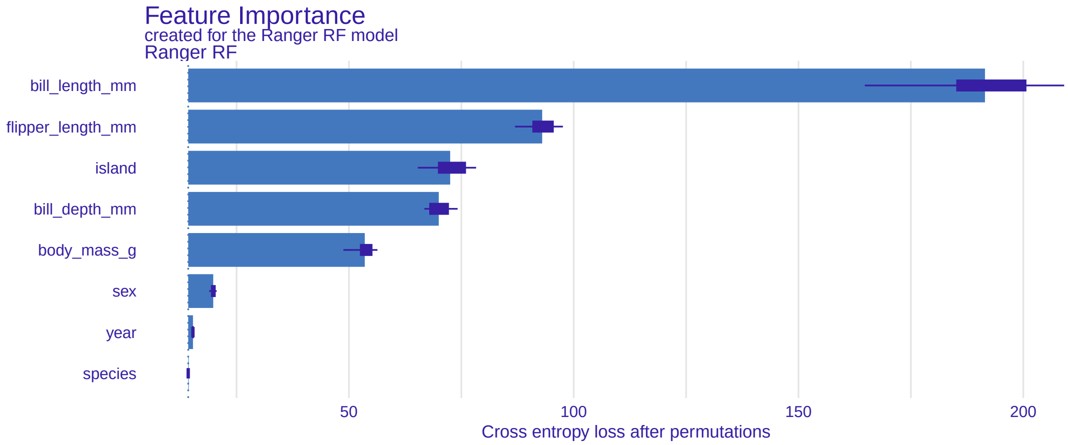


DALEX

```
library(DALEX)
library(DALEXtra)
ranger_exp = explain_mlr3(learner,
  data     = penguins2,
  y        = penguins2$species,
  label    = "Ranger RF",
  colorize = FALSE)
```

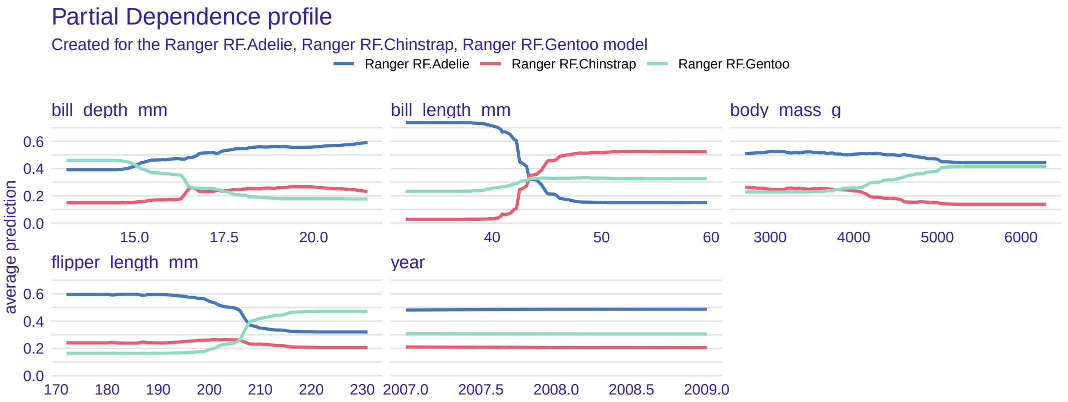
```
## Preparation of a new explainer is initiated
##   -> model label       :  Ranger RF
##   -> data              :  333  rows  8  cols
##   -> data              :  tibble converted into a data.frame
##   -> target variable   :  333  values
##   -> predict function  :  yhat.LearnerClassif  will be used (  default
##   -> predicted values  :  No value for predict function target column.
##   -> model_info        :  package mlr3 , ver. 0.14.1 , task multiclass
##   -> predicted values  :  predict function returns multiple columns:  3
```

DALEX (cont)


Feature Importance
created for the Ranger RF model
Ranger RF
bill_length_mm
flipper_length_mm
island
bill_depth_mm
body_mass_g
sex
year
species
50
100
150
200
Cross entropy loss after permutations

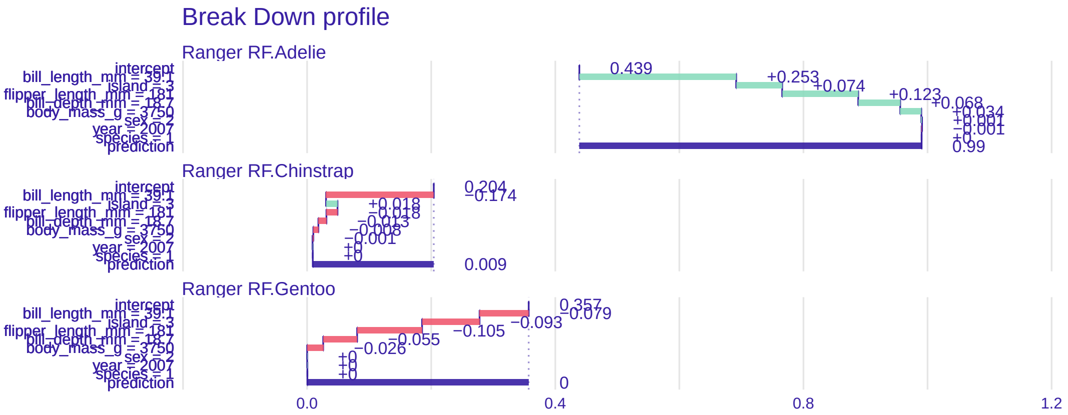
DALEX (cont 2)

```
num_features = c("bill_length_mm", "bill_depth_mm", "flipper_length_mm", "
penguins_pd <- model_profile(ranger_exp,
  variables = num_features)$agr_profiles
plot(penguins_pd)
```



Instance level explanations

```
penguin1 = penguins2[1, ]
ile_ranger = predict_parts(ranger_exp,
  new_observation = penguin1)
plot(ile_ranger)
```



Summary

- So many choices for machine learning approaches and visualisation
- `tidymodels` and `mlr3` seem to be best supported for fitting lots of machine learning models
- `DALEX` has wealth of useful plots you can use to understand your model