Applied Machine Learning - Regression Modeling

Max Kuhn (RStudio)

Outline

- Example Data
- Regularized Linear Models
- Multivariate Adaptive Regression Splines
- Ensembles of MARS Models
- Model Comparison via Bayesian Analysis

Load Packages


```
library(tidymodels)
```

```
## ── Attaching packages ──────────────────────────────────────── tidymodels 0.0.2 ──
```

```
## ✔ broom     0.5.1     ✔ purrr     0.2.5
## ✔ dials     0.0.2     ✔ recipes   0.1.4
## ✔ dplyr     0.7.8     ✔ rsample   0.0.4
## ✔ infer     0.4.0     ✔ tibble    2.0.0
## ✔ parsnip   0.0.1     ✔ yardstick 0.0.2
```

```
## ── Conflicts ─────────────────────────────────────────── tidymodels_conflicts() ──
## ✖ purrr::accumulate() masks foreach::accumulate()
## ✖ dplyr::combine()    masks gridExtra::combine()
## ✖ purrr::discard()    masks scales::discard()
## ✖ dplyr::filter()     masks stats::filter()
## ✖ dplyr::lag()        masks stats::lag()
## ✖ recipes::step()     masks stats::step()
## ✖ purrr::when()       masks foreach::when()
```

Example Data

Expanded Car MPG Data

The data that are used here are an extended version of the ubiquitous `mtcars` data set.

`fueleconomy.gov` was used to obtain fuel efficiency data on cars from 2015-2019.

Over this time range, duplicate ratings were eliminated; these occur when the same car is sold for several years in a row. As a result, there are 4209 cars that are listed in the data. The predictors include the automaker and addition information about the cars (e.g. intake valves per cycle, aspiration method, etc).

In our analysis, the data from 2015-2018 are used for training to see if we can predict the 613 cars that were new in 2018 or 2019.

```
url <- "https://github.com/topepo/cars/raw/master/2018_12_02_city/car_data_splits.RData"
temp_save <- tempfile()
download.file(url, destfile = temp_save)
load(temp_save)

car_train %>% bind_rows(car_test) %>% group_by(year) %>% count()
```

```
## # A tibble: 5 x 2
## # Groups:   year [5]
##    year     n
##   <int> <int>
## 1  2015  1277
## 2  2016   758
## 3  2017   846
## 4  2018   715
```

Gas-Related Vehicles Only!

Non-combustion engines have their MPG estimated even though there are no real gallons. Let's get rid of these for simplicity.

```
removals <- c("CNG", "Electricity")

car_train <-
  car_train %>%
  dplyr::filter(!(fuel_type %in% removals)) %>%
  mutate(fuel_type = relevel(fuel_type, "Gasoline_or_natural_gas"))

car_test <-
  car_test %>%
  dplyr::filter(!(fuel_type %in% removals)) %>%
  mutate(fuel_type = relevel(fuel_type, "Gasoline_or_natural_gas"))
```

Hands-On: Explore the Car Data

As before, let's take 10 minutes to get familiar with the training set using numerical summaries and plots.

More information about the data can be found on the government website and in this repo.

`geom_smooth` is very helpful here to discover the nature of relationships between the outcome (`mpg`) and the potential predictors.

Question: If you had two models with competing RMSE values (in mpg), how big of a difference would be big enough to be *practical*?

Linear Models

Linear Regression Analysis

We'll start by fitting linear regression models to these data.

As a reminder, the "linear" part means that the model is linear in the *parameters*; we can add nonlinear terms to the model (e.g. `x^2` or `log(x)`) without causing issues.

We could start by using `lm` and the formula method using what we've learned so far:

```
library(splines)
lm(mpg ~ . -model + ns(eng_displ, 4) + ns(cylinders, 4), data = car_train)
```

However...


```
car_train %>%
  group_by(make) %>%
  count() %>%
  arrange(n) %>%
  head(6)
```

```
## # A tibble: 6 x 2
## # Groups:   make [6]
##   make                      n
##   <fct>                 <int>
## 1 Karma                     1
## 2 Koenigsegg                1
## 3 Mobility_Ventures_LLC     1
## 4 Bugatti                   2
## 5 Lotus                     2
## 6 Pagani                    2
```

If one of these low occurrence car makes is only in the assessment set, it may cause issues (or errors) in some models.

A basic recipe will be created to account for this that collapses these data into an "other" category and will create dummy variables.

Linear Regression Analysis for the Car Data

With recipes, variables can have different roles, such as case-weights, cluster, censoring indicator, etc.

We should keep `model` in the data so that we can use it to diagnose issues but we don't want it as a predictor.

The role of this variable will be changed to `"model"`.

```
basic_rec <- recipe(mpg ~ ., data = car_train) %>%
  # keep the car name but don't use as a predictor
  update_role(model, new_role = "model") %>%
  # collapse some makes into "other"
  step_other(make, car_class, threshold = 0.005) %>%
  step_other(fuel_type, threshold = 0.01) %>%
  step_dummy(all_nominal(), -model) %>%
  step_zv(all_predictors())
```

Potential Issues with Linear Regression

We'll look at the car data and examine a few different models to illustrate some more complex models and approaches to optimizing them. We'll start with linear models.

However, some potential issues with linear methods:

- They do not automatically do *feature selection* and including irrelevant predictors may degrade performance.
- Linear models are sensitive to situations where the predictors are *highly correlated* (aka collinearity). This isn't too big of an issue for these data though.

To mitigate these two scenarios, *regularization* will be used. This approach adds a penalty to the regression parameters.

- In order to have a large slope in the model, the predictor will need to have a large impact on the model.

There are different types of regularization methods.

Effect of Collinearity

As an example of collinearity, our data set has two predictors that have a correlation above 0.95: `two_door_lug_vol` and `two_door_pass_vol`.

What happens when we fit models with both predictors versus one-at-a-time?

Term	Coefficients			
	2 Door Lugg Vol	**2 Door Pass Vol**	**Both Predictors**	**Variance Inflation**
2 Door Lugg Vol	-0.173	---	0.125	14.1
2 Door Pass Vol	---	-0.023	-0.038	14.1

The coefficients can drastically change depending on what is in the model and their corresponding variances can also be artificially large.

Regularized Linear Regression

Now suppose we want to see if *regularizing* the regression coefficients will result in better fits.

The `glmnet` model can be used to build a linear model using L_1 or L_2 regularization (or a mixture of the two).

- The general formulation minimizes: $\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + penalty$.
- An L_1 penalty (penalty is $\lambda_1 \sum |\beta_j|$) can have the effect of setting coefficients to zero.
- L_2 regularization ($\lambda_2 \sum \beta_j^2$) is basically ridge regression where the magnitude of the coefficients are dampened to avoid overfitting.

For a `glmnet` model, we need to determine the total amount regularization (called `lambda`) and the mixture of L_1 and L_2 (called `alpha`).

- `alpha` = 1 is a *lasso model* while `alpha` = 0 is *ridge regression* (aka weight decay).

Predictors require centering/scaling before being used in a `glmnet`, lasso, or ridge regression model.

Technical bits can be found in Statistical Learning with Sparsity.

Tuning the `glmnet` Model

We have two tuning parameters now (`alpha` and `lambda`). We can extend our previous grid search approach by creating a 2D grid of parameters to test.

- `alpha` must be between zero and one. A small grid is used for this parameter.
- `lambda` is not as clear-cut. We consider values on the $\log_{10}$ scale. Usually values less than one are sufficient but this is not always true.

We can create combinations of these parameters and store them in a data frame:

```
glmn_grid <- expand.grid(alpha = seq(0, 1, by = .25), lambda = 10^seq(-3, -1, length = 20))
nrow(glmn_grid)
```

```
## [1] 100
```

Instead of using `rsample` to tune the model, the `train` function in the `caret` package will be introduced.

caret

`caret` was developed to:

- create a unified interface for modeling and prediction to 238 models
- streamline model tuning using resampling
- provide a variety of "helper" functions and classes for day-to-day model building tasks
- increase computational efficiency using parallel processing
- enable several feature selection frameworks

It was originally developed in 2005 and is still very active.

There is an extensive `github.io` page and an article in JSS.

caret Basics

`train()` can take a formula method, a recipe object, or a non-formula approach (`x` / `y`) to specify the model.

```
train(recipe, data = dataset)
# or
train(y ~ ., data = dataset)
# or
train(x = predictors, y = outcome)
```

Another argument, `method`, is used to specify the type of model to fit.

This is *usually* named after the fitting function.

We will need to use `method = "glmnet"` for that model. `?models` has a list of all possibilities.

One way of listing the submodels that should be evaluated is to used the `tuneGrid` parameter:

```
train(recipe,
      data = car_train,
      method = "glmnet",
      tuneGrid = glmn_grid)
```

Alternatively, the `tuneLength` argument will let `train` determine a grid sequence for each parameter.

The Resampling Scheme

How much (and how) should we resample these data to create the model?

Previously, cross-validation was discussed. If 10-fold CV was used, the assessment set would consist of about 352 cars (on average).

That seems like an acceptable amount of data to determine the RMSE for each submodel.

`train()` has a control function that can be used to define parameters related to the numerical aspects of the search:

```
library(caret)
ctrl <- trainControl(
  method = "cv",
  # Save the assessment predictions from the best model
  savePredictions = "final",
  # Log the progress of the tuning process
  verboseIter = TRUE
  )
```

Fitting the Model via `caret::train()`

Let's add some nonlinearity and centering/scaling to the preprocessing and run the search:

```
glmn_rec <-
  basic_rec %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_ns(eng_displ, cylinders, options = list(df = 4))


set.seed(92598)
glmn_mod <- train(
  glmn_rec,
  data = car_train,
  method = "glmnet",
  trControl = ctrl,
  tuneGrid = glmn_grid
  )
```

```
## Preparing recipe
## + Fold01: alpha=0.00, lambda=0.1
## - Fold01: alpha=0.00, lambda=0.1
## + Fold01: alpha=0.25, lambda=0.1
## - Fold01: alpha=0.25, lambda=0.1
```

Aside: Submodel Trick

Our grid contained 5 values for `alpha` and 20 values of `lambda`.

Although it might seem like we are fitting 100 models *per* resample, we are not.

For many models, including `glmnet`, there are some computational shortcuts that can be used.

In this case, for a fixed value of `alpha`, the `glmnet` model computes the results for all possible values of `lambda`. Predictions from any of these models can be obtained from the same object.

This means that we only need to fit 5 models per resample.

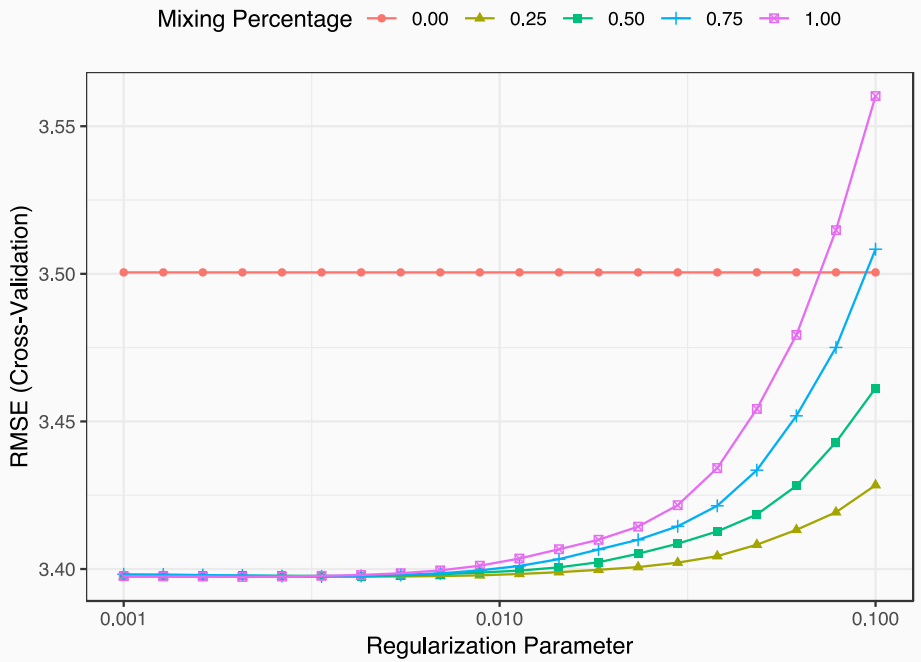
Trees and other models can often exploit this *submodel trick* and `caret` automatically does this whenever possible.

Resampling Profile for `lambda`

```
glmn_mod$bestTune
```

```
##    alpha  lambda
## 84     1 0.00207
```

```
ggplot(glmn_mod) + scale_x_log10() + theme(legend.posit
```



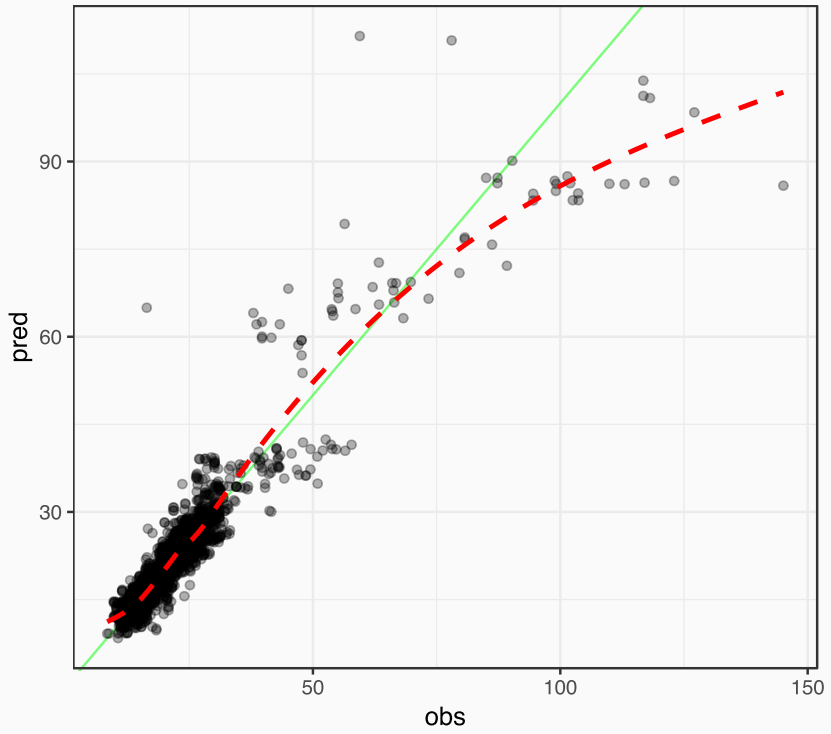
Model Checking Using the Assessment Set

Since we used `savePredictions = "final"`, the predictions on the assessment sets are contained in the sub-object `glmn_mod$pred`. This can be used to plot the data:

```
glmn_mod$pred %>% head(4)
```

```
##   alpha  lambda  obs rowIndex pred Resample
## 1     1 0.00207 19.0     3501 17.2   Fold03
## 2     1 0.00207 22.3     3374 22.2   Fold03
## 3     1 0.00207 23.8     3494 27.1   Fold03
## 4     1 0.00207 30.5     2848 34.9   Fold06
```

```
ggplot(glmn_mod$pred, aes(x = obs, y = pred)) +
  geom_abline(col = "green", alpha = .5) +
  geom_point(alpha = .3) +
  geom_smooth(se = FALSE, col = "red",
              lty = 2, lwd = 1, alpha = .5)
```



Better Plots

The `caret` object `glmn_mod$pred` doesn't contain any information about the cars. Let's add some columns and also use the `ggrepel` package to identify locations of poor fit.

The function `add_columns()` will add columns to the `pred` data in the `train` object.

`geom_text_repel()` can be used with a subset of the data to locate large residuals.

The `ggiraph` and `plotly` packages are also good options for interactivity.

```
add_columns <- function(x, dat, ...) {
  # capture any selectors and filter the data
  dots <- quos(...)
  if (!is_empty(dots))
    dat <- dplyr::select(dat, year, model, !!!dots)

  dat <-
    x %>%
    pluck("pred") %>%
    arrange(rowIndex) %>%
    dplyr::select(-rowIndex) %>%
    bind_cols(dat)

  # create a label column when possible
  if (all(c("model", "year") %in% names(dat)))
    dat <-
    dat %>%
    mutate(plot_label = paste(year, model))
  dat
}
```

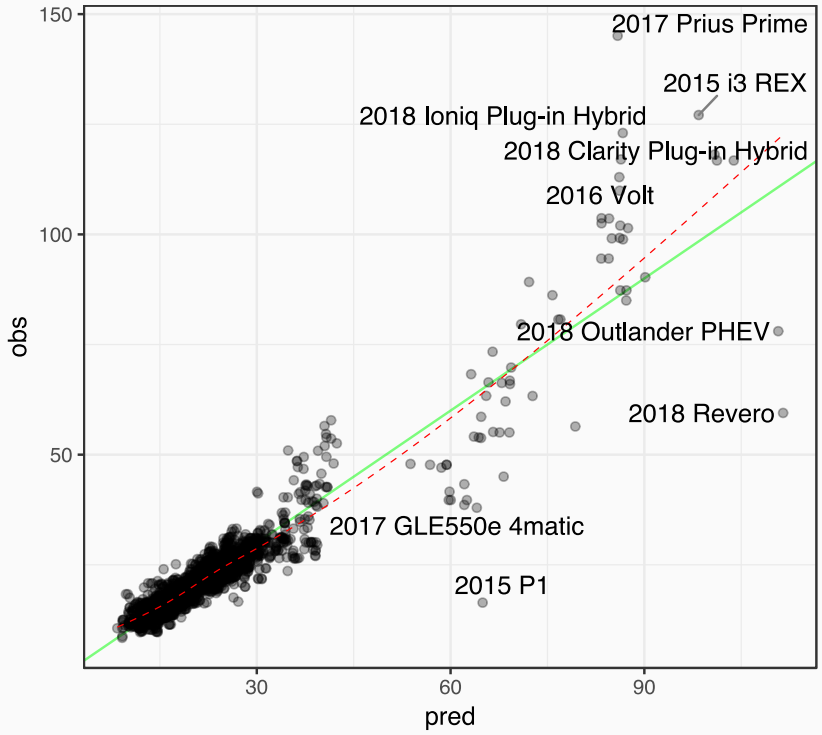
Convenience Functions


```
obs_pred_plot <- function(x, dat, cutoff = 25, ...) {

    pred_dat <- x %>%
      add_columns(dat, model, year) %>%
      mutate(residuals = obs - pred)

    ggplot(pred_dat, aes(x = pred, y = obs)) +

      geom_abline(col = "green", alpha = .5) +

      geom_point(alpha = .3) +

      geom_smooth(
        se = FALSE, col = "red",
        lty = 2, lwd = .25, alpha = .5
      ) +

      geom_text_repel(
        data = dplyr::filter(pred_dat, abs(residuals) > cutoff)
        aes(label = plot_label),
        segment.color = "grey50"
      )
}
```

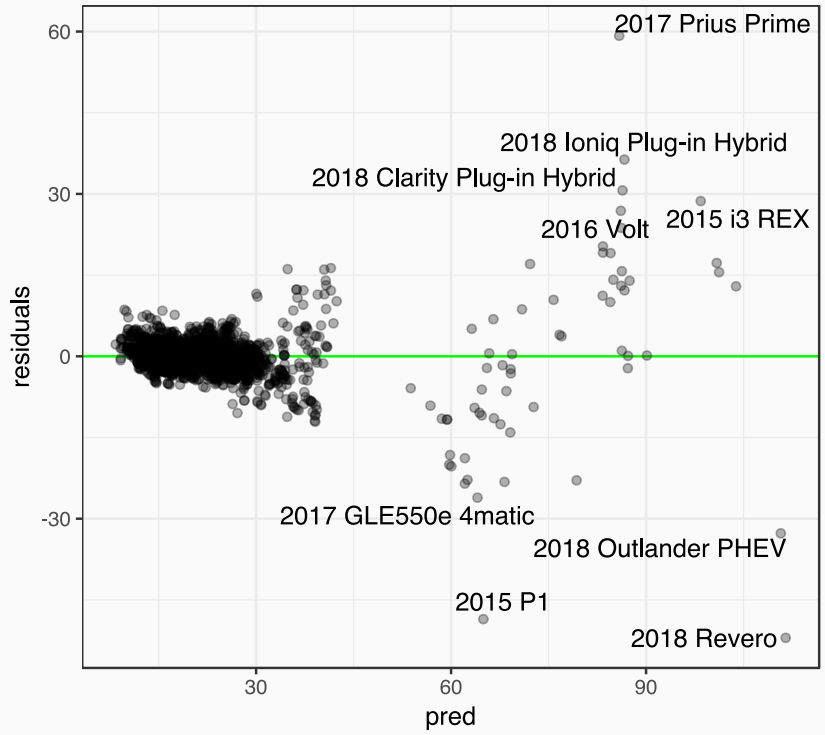
```
resid_plot <- function(x, dat, cutoff = 25, ...) {

    pred_dat <- x %>%
      add_columns(dat, model, year) %>%
      mutate(residuals = obs - pred)

    ggplot(pred_dat, aes(x = pred, y = residuals)) +

      geom_hline(col = "green", yintercept = 0) +

      geom_point(alpha = .3) +

      geom_text_repel(
        data = dplyr::filter(
          pred_dat,
          abs(residuals) > cutoff
        ),
        aes(label = plot_label),
        segment.color = "grey50"
      )
  }
```

Model Checking for glmnet Models

```
obs_pred_plot(glmn_mod, car_train)
```




```
resid_plot(glmn_mod, car_train)
```



Variable Importance Scores

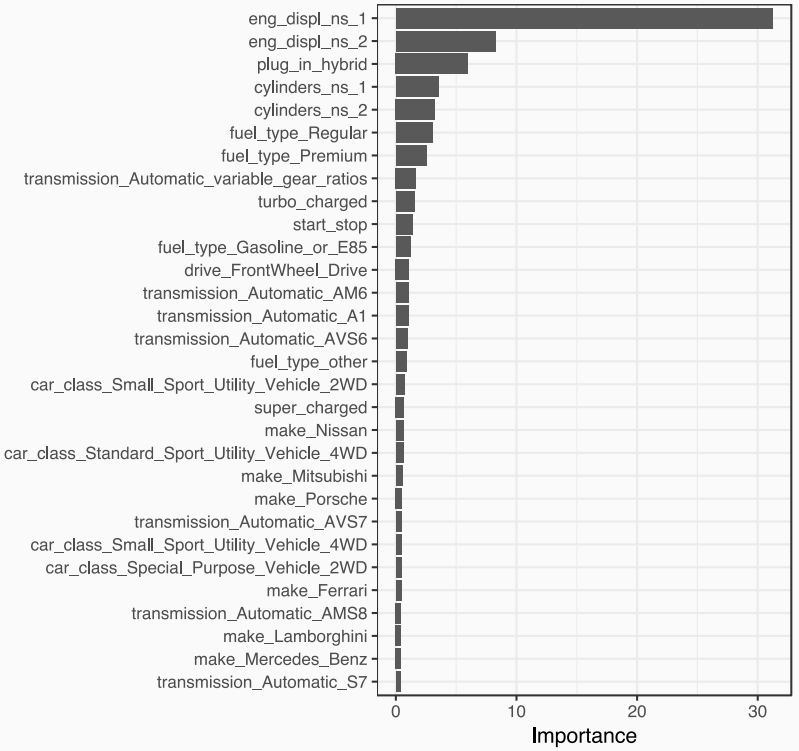
For a linear model such as `glmnet`, we can directly inspect and interpret the model coefficients to understand what is going on.

A more general approach of computing "variable importance" scores can be useful for assessing which predictors are driving the model. These are model-specific.

For this model, we can plot the absolute values of the coefficients.

This is another good reason to center and scale the predictors before the model.

```
reg_imp <- varImp(glmn_mod, scale = FALSE)
ggplot(reg_imp, top = 30) + xlab("")
```



Notes on `train()`

- Setting the seed just before calling `train()` will ensure that the same resamples are used between models. There is also a help page on reproducibility.
- `train()` calls the underlying model (e.g. `glmnet`) and arguments can be passed to the lower level functions via the `...`.
- You can write your own model code (or examine what `train()` uses) with `getModelInfo()`.
- If the formula method is used, dummy variables will *always* be generated for the model.
- If you don't like the settings that are chosen, `update()` can be used to change them without repeating all of the resampling.

Using the `glmnet` Object

The `train` object saves the optimized model that was fit to the entire training set in the slot `finalModel`.

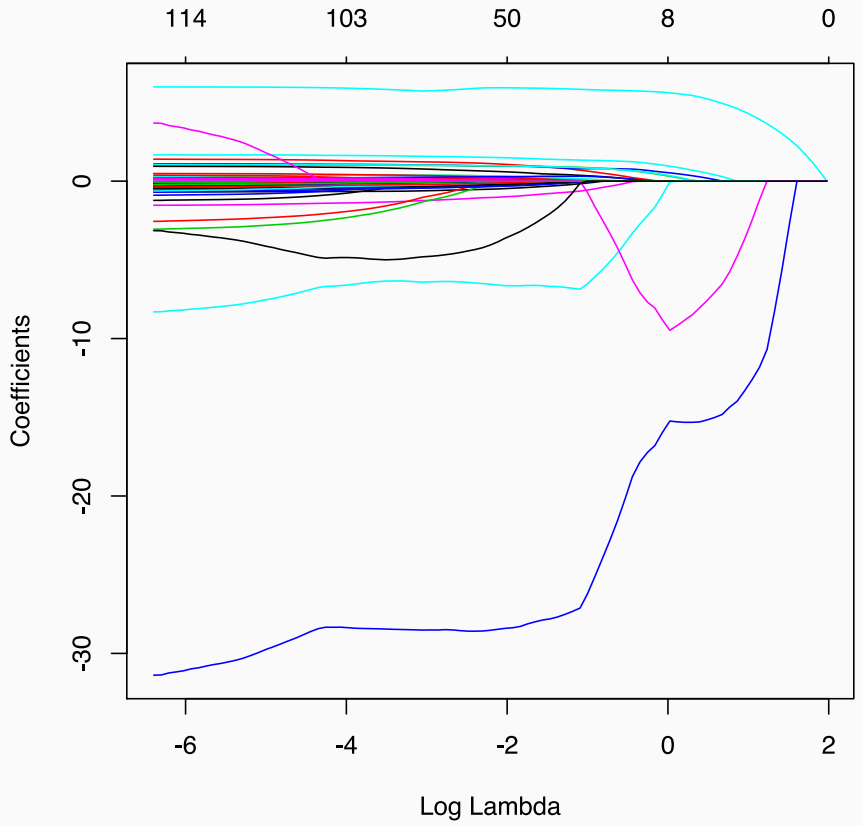
This can be used as it normally would.

The plot on the right is creating using

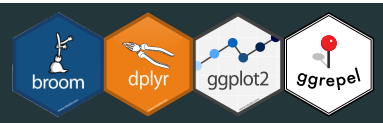
```
library(glmnet)
plot(glmn_mod$finalModel, xvar = "lambda")
```

However, **please don't predict with it**!

Use the `predict()` method on the object that is produced by `train`.

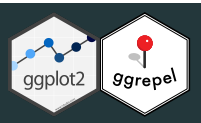


A better glmnet Plot




```
# Get the set of coefficients across penalty values
tidy_coefs <- broom::tidy(glmn_mod$finalModel) %>%
  dplyr::filter(term != "(Intercept)") %>%
  dplyr::select(-step, -dev.ratio)

# Get the lambda closest to caret's optimal choice
delta <- abs(tidy_coefs$lambda - glmn_mod$bestTune$lambda)
lambda_opt <- tidy_coefs$lambda[which.min(delta)]

# Keep the large values
label_coefs <- tidy_coefs %>%
  mutate(abs_estimate = abs(estimate)) %>%
  dplyr::filter(abs_estimate >= 3) %>%
  distinct(term) %>%
  inner_join(tidy_coefs, by = "term") %>%
  dplyr::filter(lambda == lambda_opt)

# plot the paths and highlight the large values
tidy_coefs %>%
  ggplot(aes(x = lambda, y = estimate, group = term, col = term, label = term)) +
  geom_line(alpha = .4) +
  theme(legend.position = "none") +
  scale_x_log10() +
  geom_text_repel(data = label_coefs, aes(x = .0005))
```

A better glmnet Plot



Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines (MARS)

MARS is a nonlinear machine learning model that develops sequential sets of artificial features that are used in linear models (similar to the previous spline discussion).

The features are "hinge functions" or single knot splines that use the function:

```
h(x) <- function(x) ifelse(x > 0, x, 0)
```

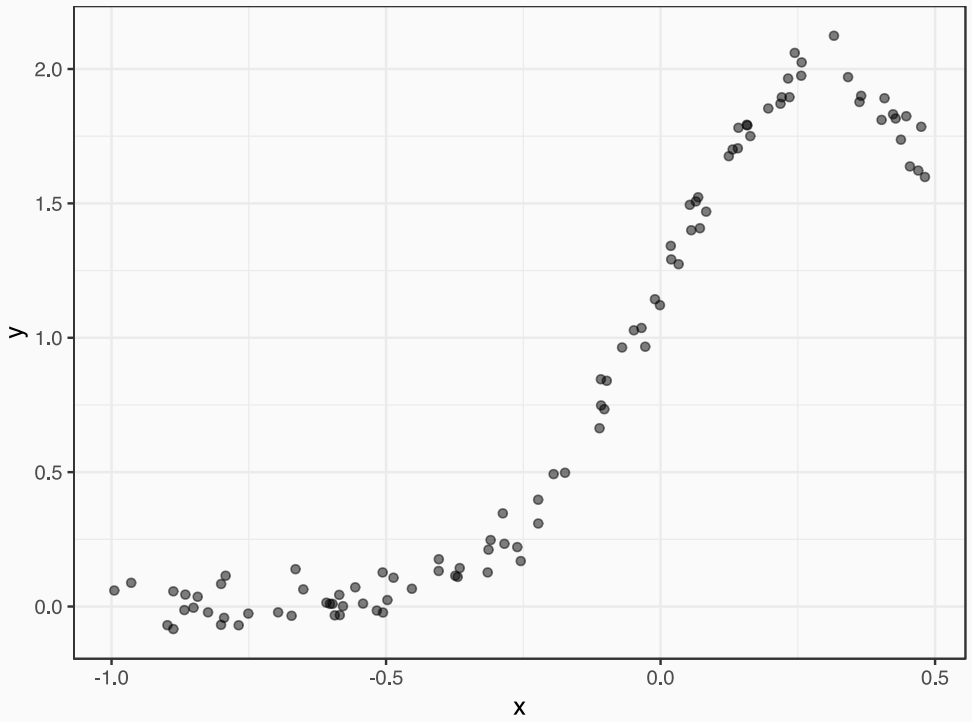
The MARS model does a fast search through every predictor and every value of each predictor to find a suitable "split" point for the predictor that results in the best features.

Suppose a value `x0` is found. The MARS model creates two model terms `h(x - x0)` and `h(x0 - x)` that are added to the intercept column. This creates a type of *segmented regression*.

These terms are the same as deep learning rectified linear units (ReLU).

Let's look at some example data...

Simulated Data: `y = 2 * exp(-6 * (x - 0.3)^2) + e`



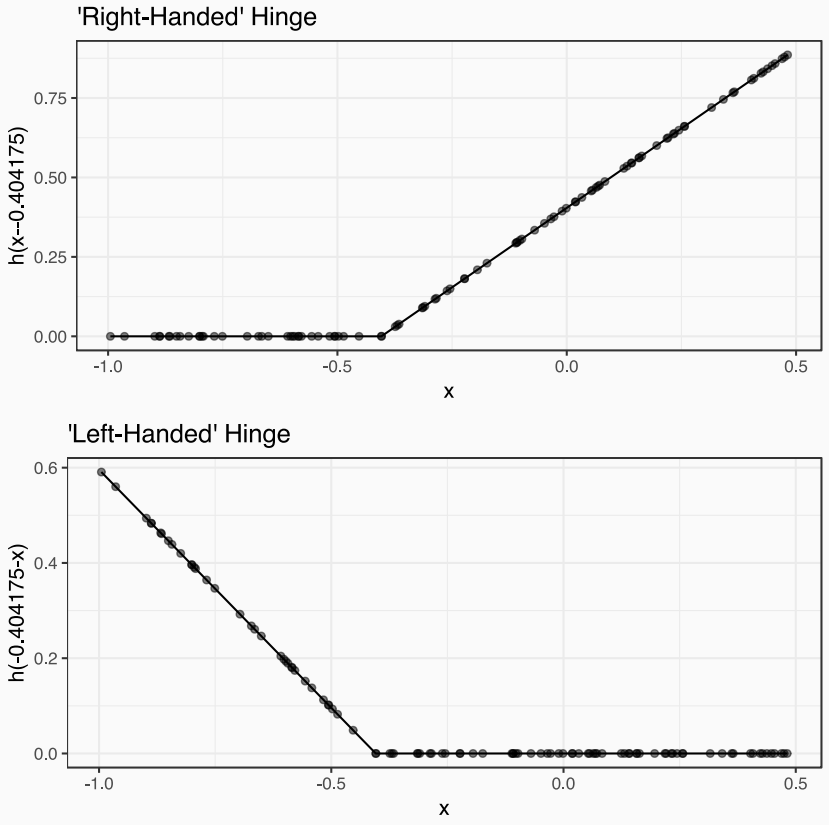
MARS Feature Creation -- Iteration #1

After searching through these data, the model evaluates all possible values of `x0` to find the best "cut" of the data. It finally chooses a value of -0.404.

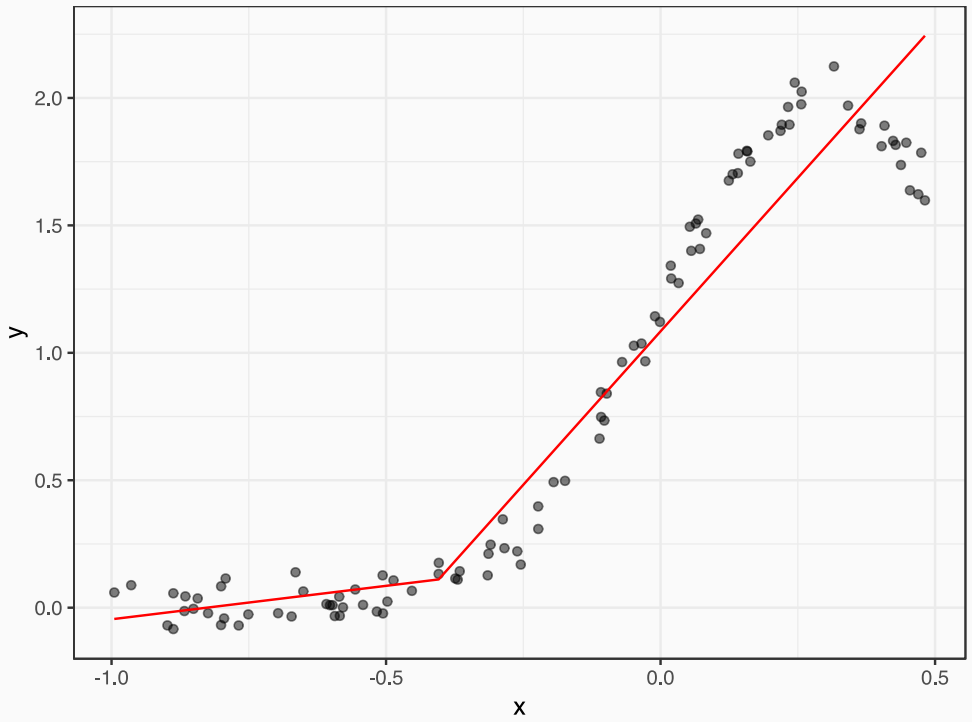
To do this, it creates these two new predictors that isolate different regions of `x`.

If we stop there, these two terms would be added into a linear regression model, yielding:

```
## y =
##   0.111
##   - 0.262 * h(-0.404175 - x)
##   +  2.41 * h(x - -0.404175)
```



Fitted Model with Two Features



Growing and Pruning

Similar to tree-based models, MARS starts off with a "growing" stage where it keeps adding new features until it reaches a pre-defined limit.

After the first pair is created, the next cut-point is found using another exhaustive search to see which split of a predictor is best *conditional on the existing features*.

Once all the features are created, a *pruning phase* starts where model selection tools are used to eliminate terms that do not contribute meaningfully to the model.

Generalized cross-validation (GCV) is used to efficiently remove model terms while still providing some protection from overfitting.

Model Size

There are two approaches:

1. Use the internal CGV to prune the model to the best subset size. This is fast but you don't learn much and it may underselect terms.

2. Use the external resampling (10-fold CV here) to tune the model as you would any other.

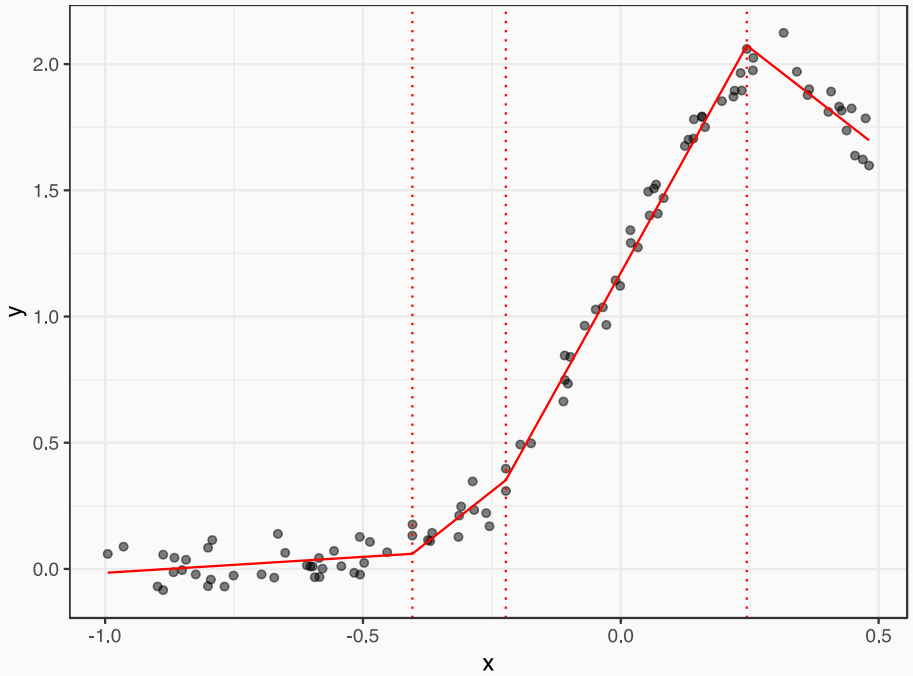
I usually don't start with GCV. Instead use method #2 above to understand the trends.

The Final Model

For the simulated data, the mars model only requires 4 features to model the data (via GCV).

```
## y =
##   0.0599
##   - 0.126 * h(-0.404175 - x)
##   +  1.61 * h(x - -0.404175)
##   +  2.08 * h(x - -0.222918)
##   -  5.27 * h(x - 0.244406)
```

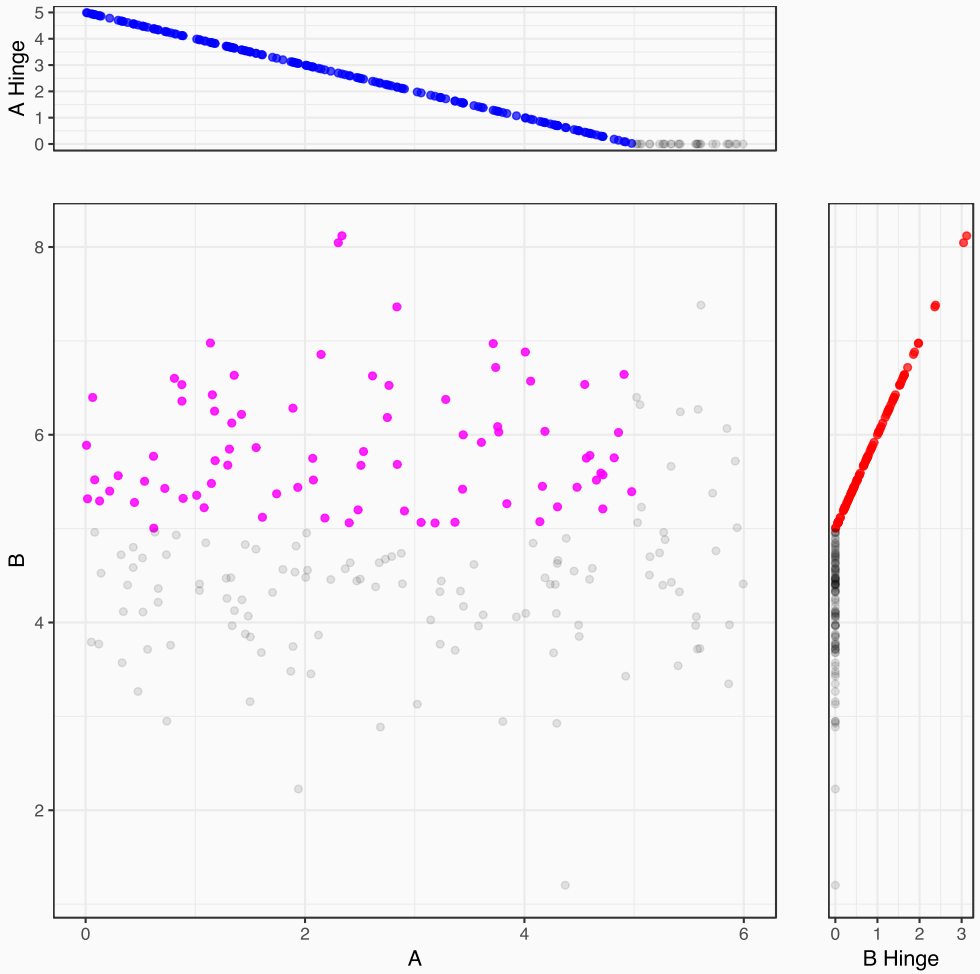
The parameters are estimated by added the MARS features into ordinary linear regression models using least squares.



Aspects of MARS Models

- The model also tests to see if a simple linear term is best (i.e. not split). This is also how dummy variables are evaluated.
- The model automatically conducts *feature selection*; if a predictor is never used in a split, it is functionally independent of the model. This is really good!
- If an additive model is used (as in the previous example), the functional form of each predictor can be determined (and visualized) independently for each predictor.
- A *second degree* MARS model also evaluates interactions of two hinge features (e.g. `h(x0 - x) * h(z - z0)`). This can be useful in isolating regions of bivariate predictor space since it divides two-dimensional space into four quadrants. (see next slide)

Second Degree MARS Term Example



MARS in R

The `mda` package has a `mars` function but the `earth` package is far superior.

The `earth()` function has both formula and non-formula interfaces. It can also be used with generalized linear models and flexible discriminant analysis.

To use the nominal growing and GCV pruning process, the syntax is

```
earth(y ~ ., data)

# or

earth(x = x, y = y)
```

The feature creation process can be controlled using the `nk`, `nprune`, and `pmethod` parameters although this can be somewhat complex.

There is a variable importance method that tracks the changes in the GCV results as features are added to the model.

MARS via `caret`

There is a several ways to fit the MARS model using `train()`.

- `method = "earth"` avoids pruning using GCV and uses external resampling to choose the number of retained model terms (using the sub-model trick). The two tuning parameters are `nprune` (number of retained features) and `degree` (the amount of interaction allowed).
- `method = "gcvEarth"` is also available and uses GCV. The `degree` parameter requires tuning.

I usually use the manual method to better understand the pruning process.

For preprocessing, there is no need to remove zero-variance predictors here (beyond computational efficiency) but dummy variables are required for qualitative predictors.

Centering and scaling are not required.

Tuning the Model

We can reuse much of the `glmnet` syntax to tune the model.

```
ctrl$verboseIter <- FALSE

mars_grid <- expand.grid(degree = 1:2, nprune = seq(2, 26, by = 2))

# Using the same seed to obtain the same
# resamples as the glmnet model.
set.seed(92598)
mars_mod <- train(
  basic_rec,
  data = car_train,
  method = "earth",
  tuneGrid = mars_grid,
  trControl = ctrl
)
```

Running the resampling models (plus the last one), this takes 4.5m on my laptop.

While We Wait, Can I Interest You in Parallelism?

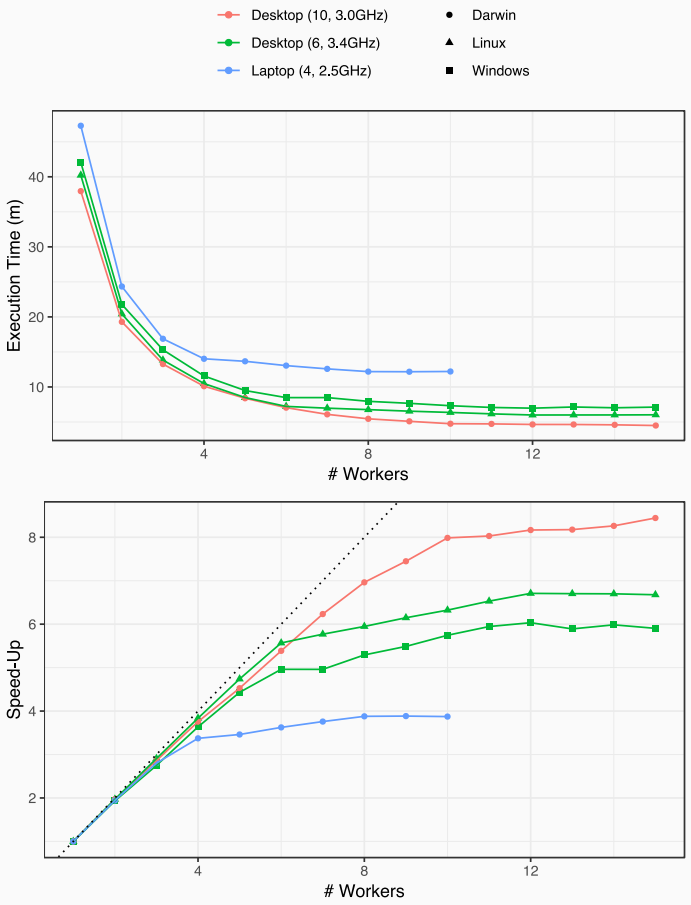
There is no real barrier to running these in parallel.

Can we benefit from splitting the fits up to run on multiple cores?

These speed-ups can be very model- and data-dependent but this pattern generally holds.

Note that there is little incremental benefit to using more workers than physical cores on the computer. Use `parallel::detectCores(logical = FALSE)`.

(A lot more details can be found in this blog post)



Running in Parallel for `caret`

To loop through the models and data sets, `caret` uses the `foreach` package, which can parallelize `for` loops.

`foreach` has a number of *parallel backends* which allow various technologies to be used in conjunction with the package.

On CRAN, these are the "`do{X}`" packages, such as `doAzureParallel`, `doFuture`, `doMC`, `doMPI`, `doParallel`, `doRedis`, and `doSNOW`.

For example, `doMC` uses the `multicore` package, which forks processes to split computations (for unix and OS X). `doParallel` can be used for all operating systems.

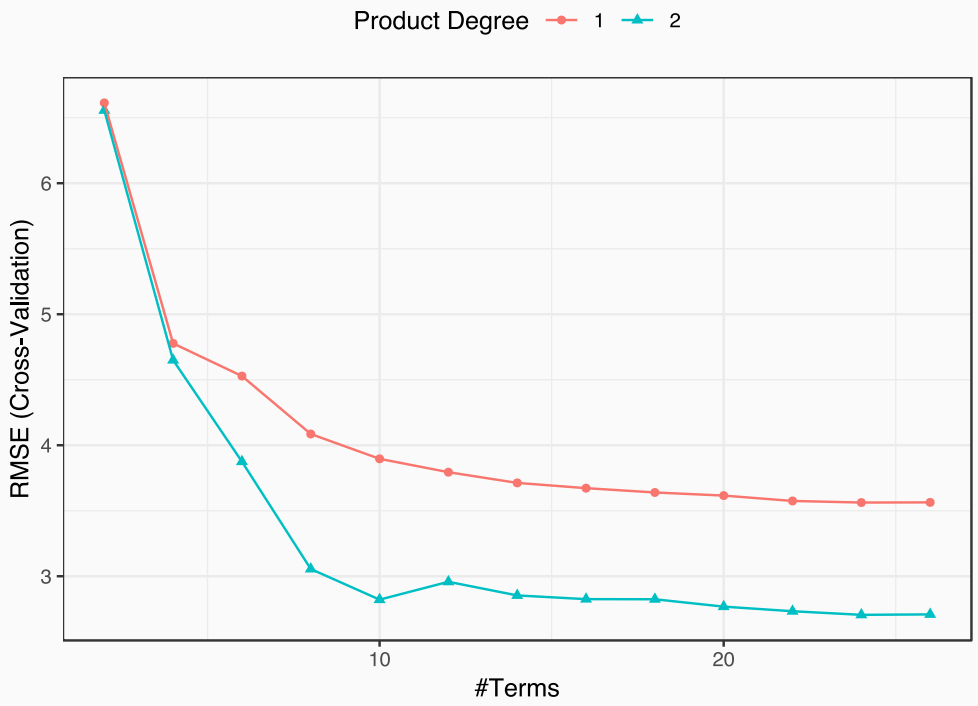
To use parallel processing in `caret`, no changes are needed when calling `train()`.

The parallel technology must be *registered* with `foreach` prior to calling `train()`:

```
library(doParallel)
cl <- makeCluster(6)
registerDoParallel(cl)

# run `train()`...

stopCluster(cl)
```

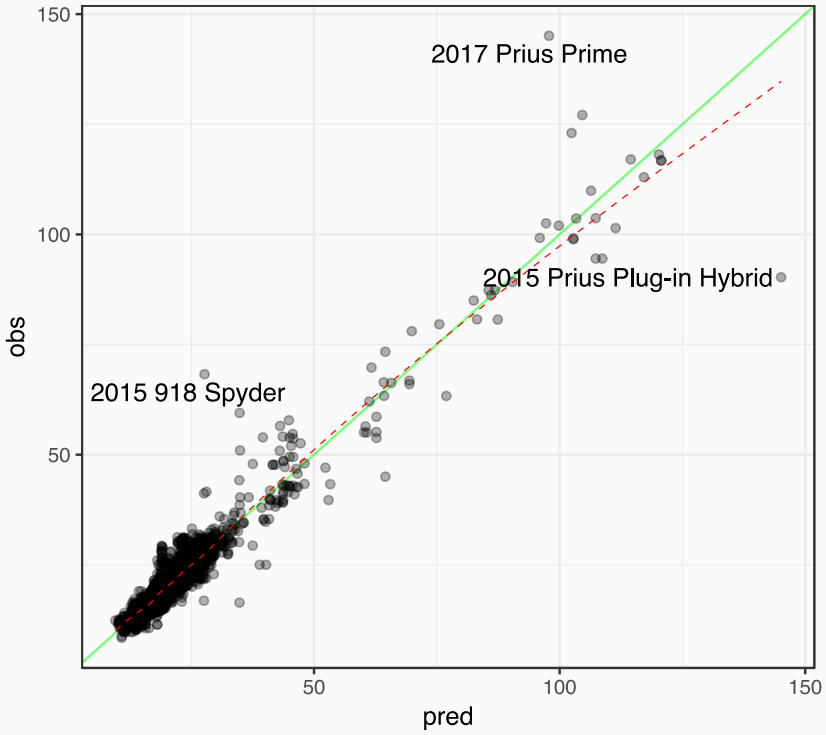
Resampling Profile for MARS

```
ggplot(mars_mod) + theme(legend.position = "top")
```

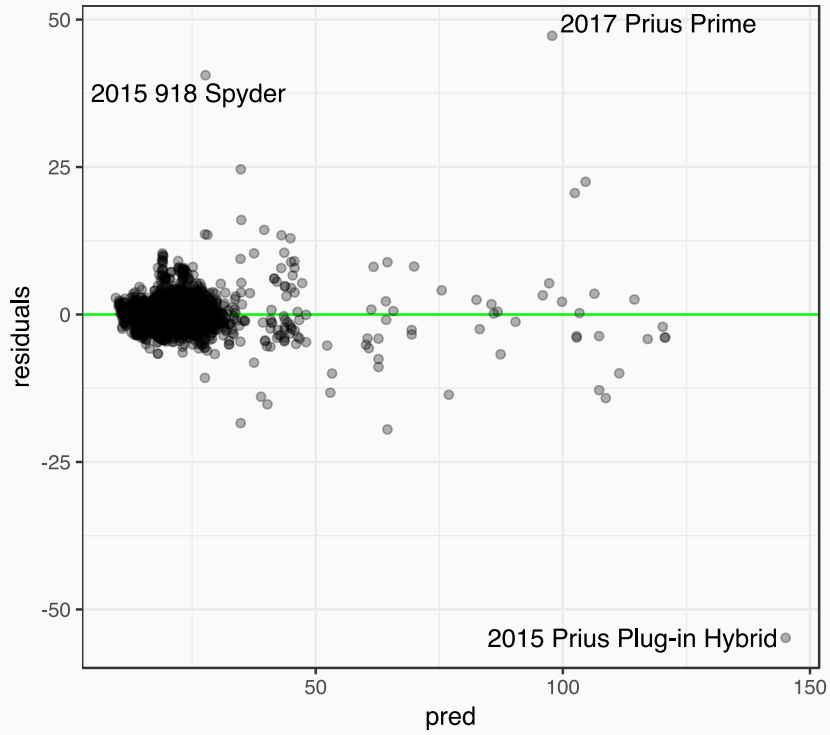


MARS Performance Plots

```
obs_pred_plot(mars_mod, car_train)
```




```
resid_plot(mars_mod, car_train)
```



The Underlying Model

```
library(earth)
mars_mod$finalModel
```

```
## Selected 24 of 31 terms, and 19 of 114 predictors
## Termination condition: RSq changed by less than 0.001 at 31 terms
## Importance: plug_in_hybrid, year, eng_displ, start_stop, transmission_Automatic_variable_gear_ratios, ...
## Number of terms at each degree of interaction: 1 9 14
## GCV 5.19    RSS 17691    GRSq 0.946    RSq 0.948
```

Model Terms

```
##   20.1
##   -     7664 * plug_in_hybrid
##   +     3.77 * make_Mazda
##   +     2.61 * transmission_Automatic_AVS7
##   -     1.02 * transmission_Automatic_S6
##   +     2.46 * transmission_Automatic_variable_gear_ratios
##   -    0.513 * h(cylinders-6)
##   +     5.09 * h(2.8-eng_displ)
##   -     1.35 * h(eng_displ-2.8)
##   -  0.00982 * h(102-four_door_pass_vol)
##   +     3.82 * year*plug_in_hybrid
##   +     17.5 * start_stop*transmission_Automatic_AM6
##   +     13.4 * start_stop*transmission_Automatic_AVS6
##   +     17.4 * start_stop*transmission_Automatic_variable_gear_ratios
##   -     15.4 * plug_in_hybrid*turbo_charged
##   -     26.6 * plug_in_hybrid*make_Cadillac
##   +     16.6 * plug_in_hybrid*make_Toyota
##   +     25.2 * plug_in_hybrid*transmission_Automatic_AMS7
##   -     15.7 * super_charged*transmission_Automatic_variable_gear_ratios
##   +     23.9 * h(2.8-eng_displ)*plug_in_hybrid
##   -     1.65 * h(2.8-eng_displ)*turbo_charged
##   +     1.88 * h(2.8-eng_displ)*drive_FrontWheel_Drive
##   +   0.0751 * h(1.6-eng_displ)*h(102-four_door_pass_vol)
##   +     1.78 * h(10-four_door_lug_vol)*h(four_door_pass_vol-102)
```

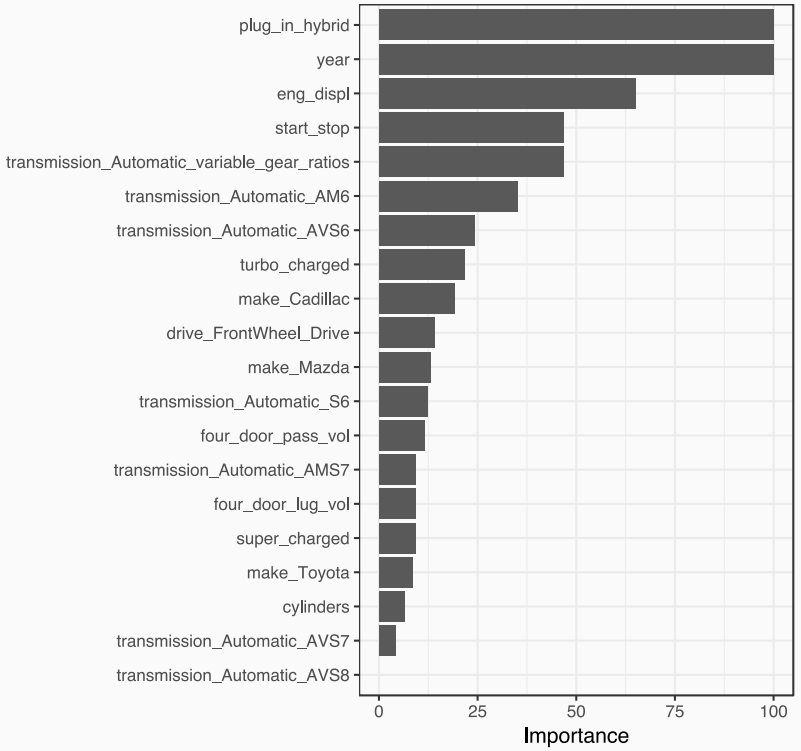
Variable Importance Scores

Recall that as MARS adds or drops terms from the model, the change in the GCV statistic is used to determine the worth of the terms.

`earth` tracks the changes for each predictor and measures the variable importance based on how much the GCV error *decreases* when the model term is added.

This is *cumulative* when multiple terms involve the same predictor multiple times.

```
mars_imp <- varImp(mars_mod)
ggplot(mars_imp, top = 20) + xlab("")
```



Use GCV to Tune the Model

```
set.seed(92598)
mars_gcv_mod <- train(
  basic_rec,
  data = car_train,
  method = "gcvEarth",
  tuneGrid = data.frame(degree = 1:2),
  trControl = ctrl
)
mars_gcv_mod$finalModel
```

```
## Selected 27 of 31 terms, and 19 of 114 predictors
## Termination condition: RSq changed by less than 0.001 at 31 terms
## Importance: plug_in_hybrid, year, eng_displ, start_stop, transmission_Automatic_variable_gear_ratios, ...
## Number of terms at each degree of interaction: 1 11 15
## GCV 5.13    RSS 17406    GRSq 0.946    RSq 0.948
```

The results are very similar to the previous model. This is not typical but it is faster (4.9-fold). We will exploit this in a moment.

Ensembles via Bagging

Bagging Models

Bagging is a method of creating *ensembles of the model type*.

Instead of using the training set, many variations of the data are created that spawn multiple *versions* of the same model.

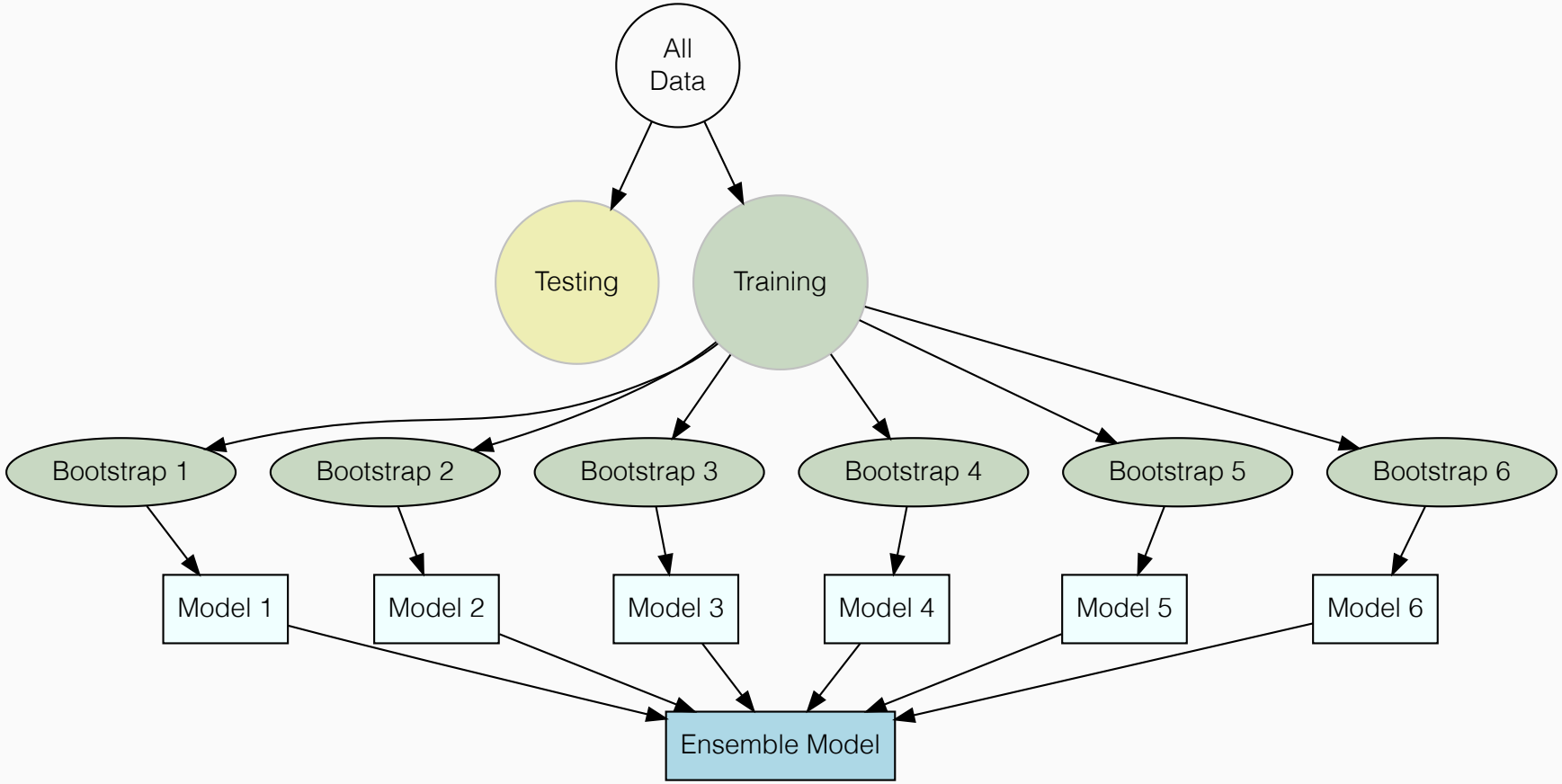
When predicting a new sample, the individual predictions are generated for each model in the ensemble, and these are blended into a single value. This reduces the variation in the predictions since are averaging pseudo-replicates of the model.

Bagging creates the data sets using a *bootstrap sample* of the training set.

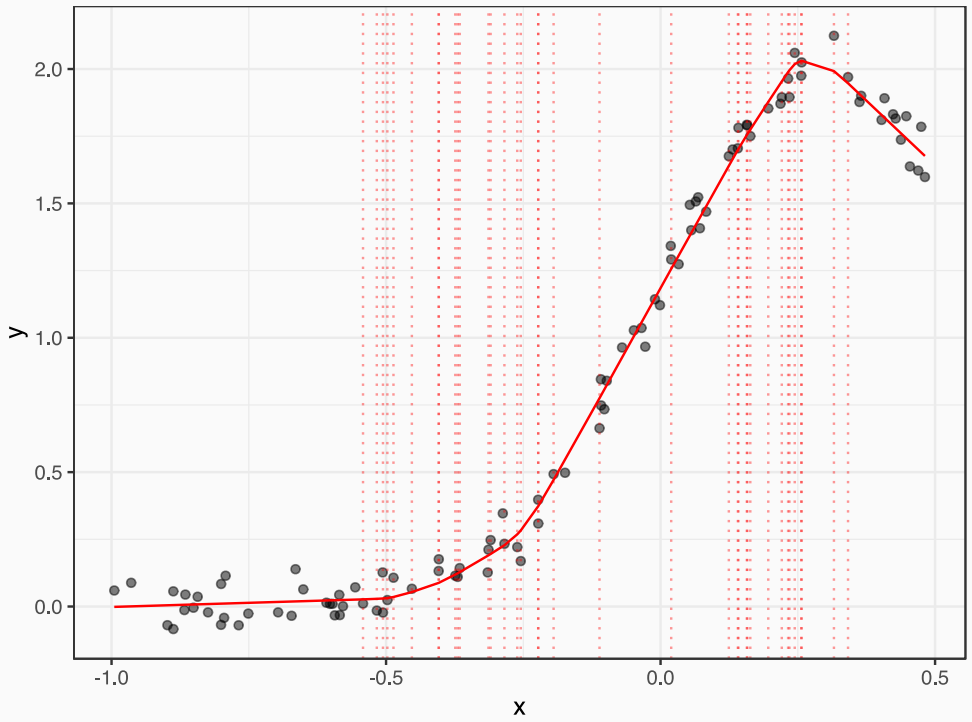
Bagging is most useful when the underlying model has some *instability*. This means that slight variations in the data cause significant changes in the model fit.

For example, simple linear regression would not be a suitable candidate for ensembles but MARS has *potential* for improvement. It does have the effect of *smoothing* the model predictions.

Bagging Process



Bagged Additive MARS Example



Does Bagging Help the Cars Model?

```
set.seed(92598)
mars_gcv_bag <- train(
  basic_rec,
  data = car_train,
  method = "bagEarthGCV",
  tuneGrid = data.frame(degree = 1:2),
  trControl = ctrl,
  # Number of bootstraps for `bagEarth` function
  B = 50
)
```

On my laptop, this will take about 39m to run without parallel processing

Does Bagging Help the Cars Model?

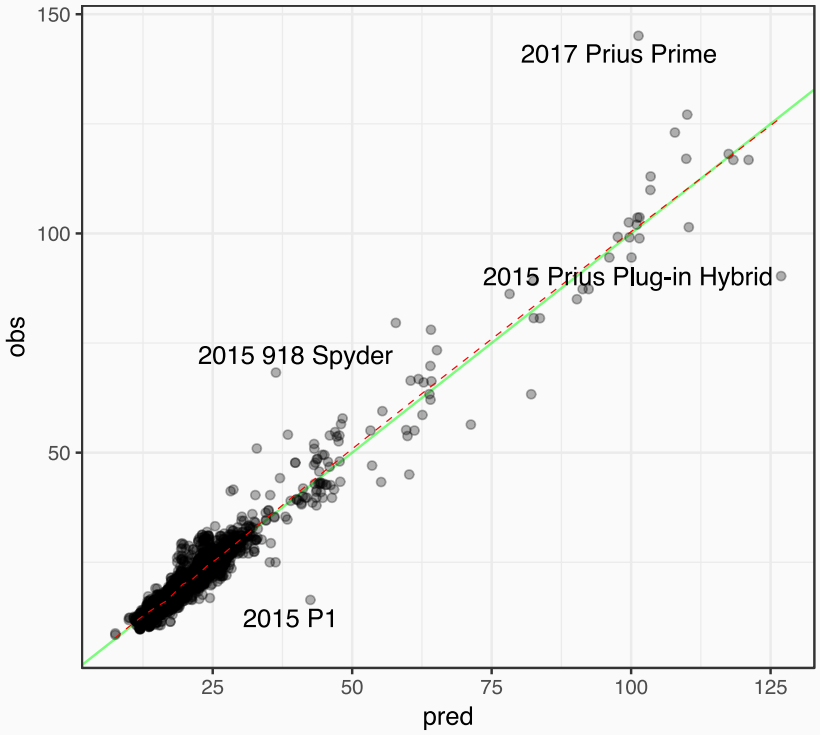
```
mars_gcv_bag
```

```
## Bagged MARS using gCV Pruning
##
## 3526 samples
##   29 predictor
##
## Recipe steps: other, other, dummy, zv
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3174, 3173, 3174, 3174, 3173, 3173, ...
## Resampling results across tuning parameters:
##
##   degree  RMSE  Rsquared  MAE
##   1       3.46  0.872     1.91
##   2       2.40  0.933     1.46
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was degree = 2.
```

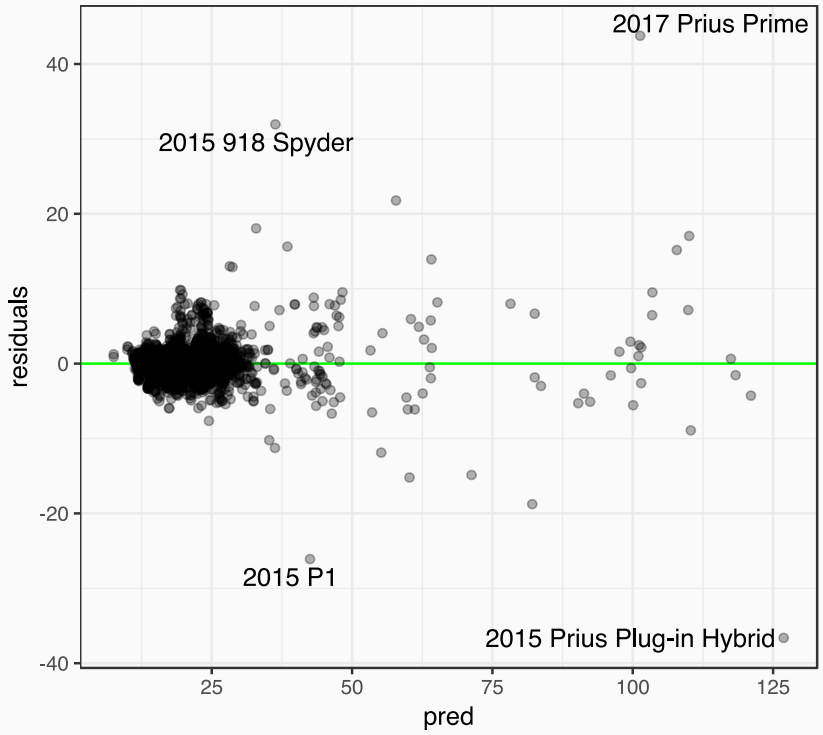
Smaller RMSE (was 2.69 mpg) but is it real?

Bagged MARS Performance Plots

```
obs_pred_plot(mars_gcv_bag, car_train)
```




```
resid_plot(mars_gcv_bag, car_train)
```



Comparing Models via Bayesian Analysis

Collecting and Analyzing the Resampling Results

First, we can use the `resamples` function in `caret` to collect and collate the cross-validation results across the different models.

```
rs <- resamples(
  list(glmnet = glmn_mod, MARS = mars_mod,  bagged_MARS = mars_gcv_bag)
)
```

The `tidyposterior` package is designed to estimate the relationship between the *outcome metrics* (i.e. RMSE) as a function of the model type (i.e. MARS) in a way that takes into account the resample-to-resample covariances that can occur.

A Bayesian linear model is used here for that purpose.

I recommend the book *Statistical Rethinking* if you are new to Bayesian analysis.

Bayes' Rule will be discussed in more detail in the Classification notes to come.

Bayesian Hierarchical Linear Model

If we did a basic ANOVA model to compare models, it might look like:

$$RMSE = b_0 + b_1 m_1 + b_2 m_2$$

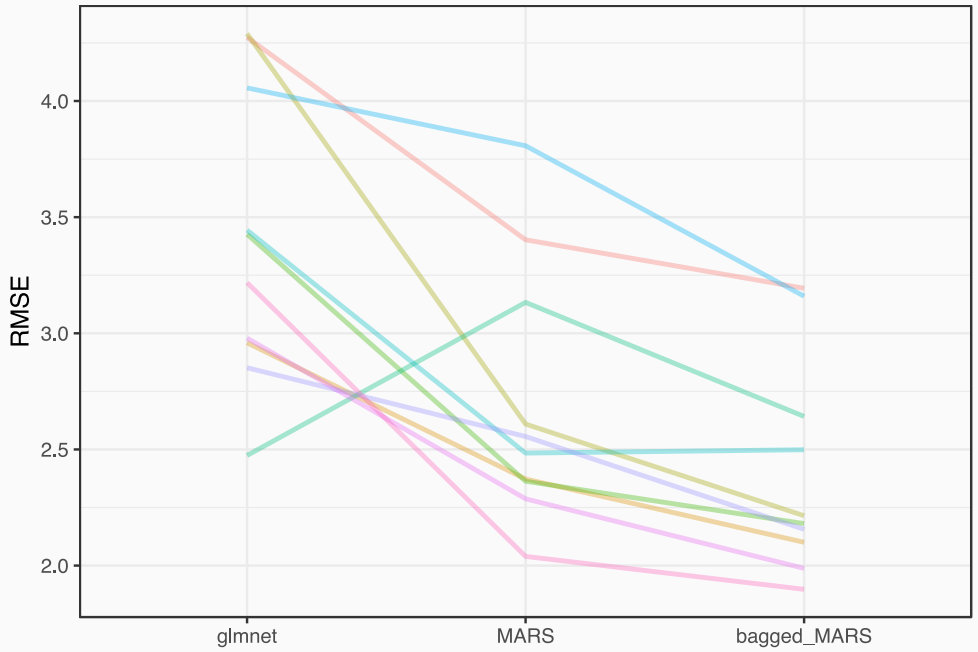
where the m_j are indicator variables for the model (`glmnet`, MARS, etc).

However, there are usually resample-to-resample effects. To account for this, we can make this ANOVA model *specific to a resample*:

$$RMSE_i = (b_0 + b_{i0}) + b_1 m_{i1} + b_2 m_{i2}$$

where *i* is the i^{th} cross-validation fold.

Random Intercept?



Bayesian Hierarchical Linear Model

We might assume that each

$$RMSE_{ij} \sim N(\beta_{i0} + \beta_j m_{ij}, \sigma^2)$$

and that the b parameters have some multivariate normal distribution with mean β and some covariance matrix. The distribution of the β values, along with a distribution for the variance parameter, are the *prior distributions*.

Bayesian analysis can be used to estimate these parameters. `tidyposterior` uses Stan to fit the model.

There are options to change the assumed distribution of the metric (i.e. gamma instead of normality) or to transform the metric to normality. Different variances per model can also be estimated and the priors can be changed.

Comparing Models using Bayesian Analysis

`tidyposterior::perf_mod()` can take the `resamples` object as input, configure the Bayesian model, and estimate the parameters:

```
library(tidyposterior)
rmse_mod <- perf_mod(rs, seed = 4344, iter = 5000, metric = "RMSE")
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000102 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.02 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 1: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 1: Iteration: 3000 / 5000 [ 60%]  (Sampling)
```

Showing the Posterior Distributions

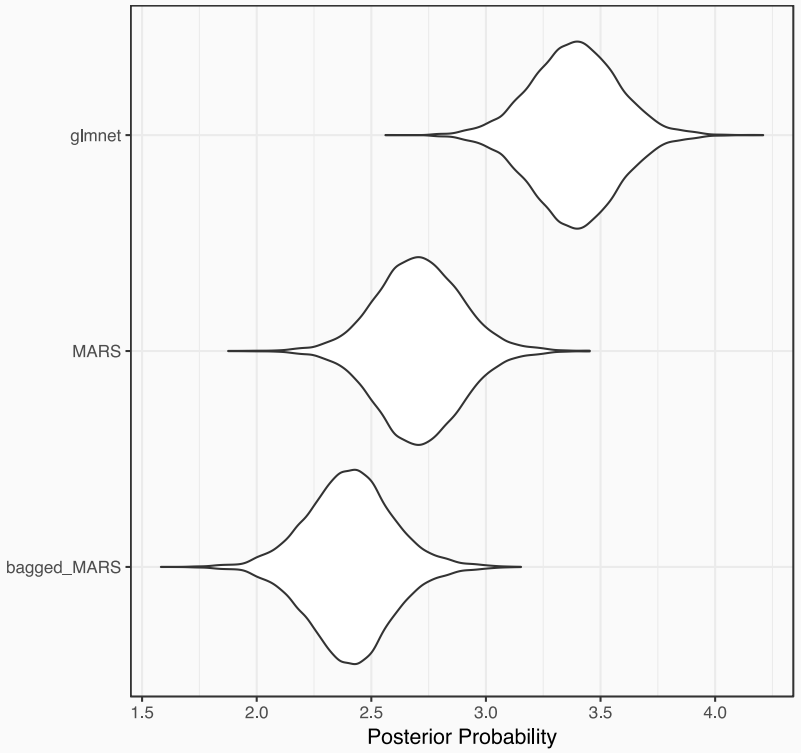
Bayesian analysis can produce probability distributions for the estimated parameters (aka the *posterior distributions*). These can be used to compare models.

```
posteriors <- tidy(rmse_mod, seed = 366784)
summary(posteriors)
```

```
## # A tibble: 3 x 4
##   model        mean lower upper
##   <chr>       <dbl> <dbl> <dbl>
## 1 bagged_MARS  2.41  2.10  2.72
## 2 glmnet       3.39  3.08  3.69
## 3 MARS         2.71  2.41  3.01
```

These are 90% *credible intervals*.

```
ggplot(posteriors) + coord_flip()
```



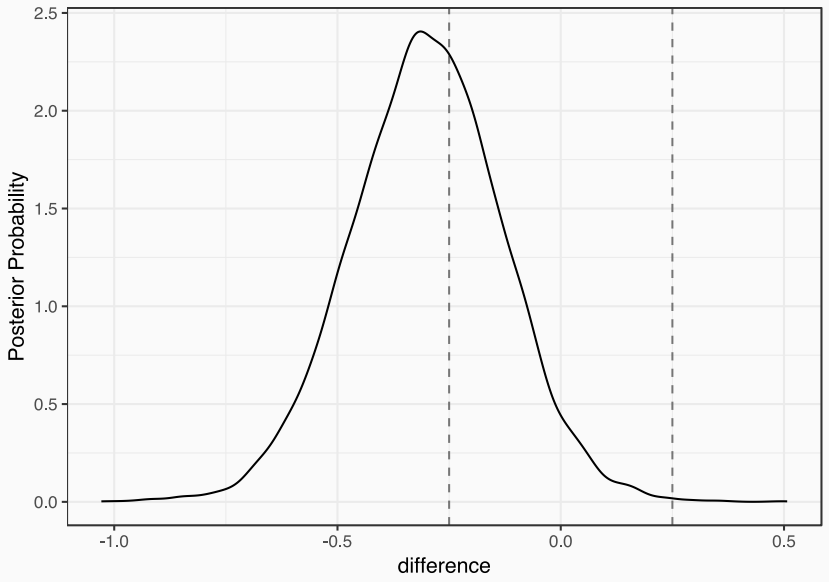
Comparing Models

Once the posteriors for each model are calculated, it is pretty easy to compute the posterior for the differences between models and plot them.

```
differences <-
  contrast_models(
    rmse_mod,
    list_1 = "bagged_MARS",
    list_2 = "MARS",
    seed = 2581
  )
```

If we know the size of a practical difference in RMSE values, this can be included into the analysis to get ROPE estimates (Region of Practical Equivalence).

```
ggplot(differences, size = 0.25)
```



Comparing Models

If we think that 0.25 MPG is a real difference, we can assess which models are practically different from one another.

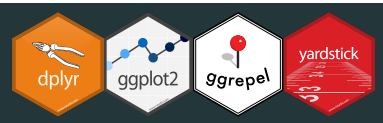
```
summary(differences, size = 0.25) %>%
  dplyr::select(contrast, mean, size, contains("pract"))
```

```
## # A tibble: 1 x 6
##   contrast              mean  size pract_neg pract_equiv pract_pos
##   <chr>                <dbl> <dbl>     <dbl>       <dbl>     <dbl>
## 1 bagged_MARS vs MARS -0.300  0.25     0.615       0.384   0.00120
```

`pract_neg` is the probability that the difference in RMSE is practically negative based on our thoughts about `size`.

Test Set Results

Predicting the Test Set



Making predictions on new data is pretty simple:

```
car_test <- car_test %>%
  mutate(pred = predict(mars_gcv_bag, car_test))

rmse(car_test, truth = mpg, estimate = pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        2.16
```

```
ggplot(car_test, aes(x = mpg, y = pred, label = model))
  geom_abline(col = "green", alpha = .5) +
  geom_point(alpha = .3) +
  geom_smooth(se = FALSE, col = "red",
              lty = 2, lwd = .5, alpha = .5) +
  geom_text_repel(
    data = car_test  %>% dplyr::filter(abs(mpg - pred)
    segment.color = "grey50"
    )
```