Creating model *en mass* with workflow sets

Max Kuhn

2021-05-18

tidymodels.org

Tidy Modeling with R (tmwr.org)

A quick(ish) tour of tidymodels

In tidymodels, there is the idea that a model-oriented data analysis consists of

- a preprocessor, and
- a model

The preprocessor might be a simple formula or a sophisticated recipe.

It's important to consider both of these activities as part of the data analysis process.

- Post-model activities should also be included there (e.g. calibration, cut-off optimization, etc.)
- We don't have those implemented yet.

Basic tidymodels components

building blocks

preprocessors

`y ~ x1 + x2`

recipes

models

parsnip

A relavant example

Let's say that we have some highly correlated predictors and we want to reduce the correlation by first applying principal component analysis to the data.

- AKA principal component regression

A relavant example

Let's say that we have some highly correlated predictors and we want to reduce the correlation by first applying principal component analysis to the data.

- AKA ~~principal component regression~~ feature extraction

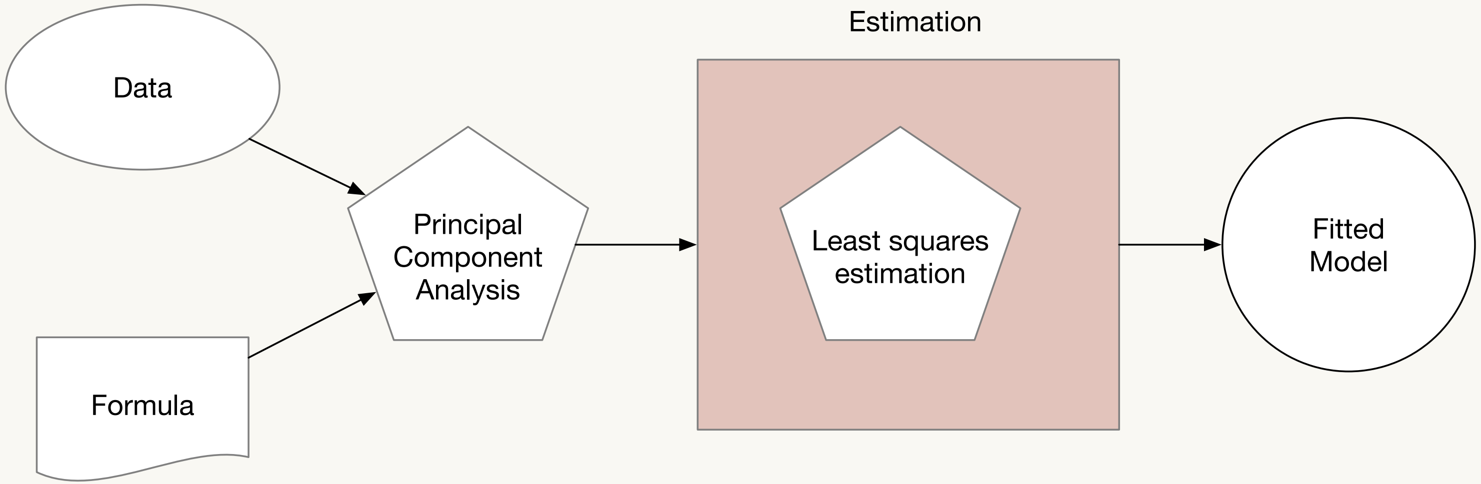
A relavant example

Let's say that we have some highly correlated predictors and we want to reduce the correlation by first applying principal component analysis to the data.

- AKA ~~principal component regression~~ feature extraction

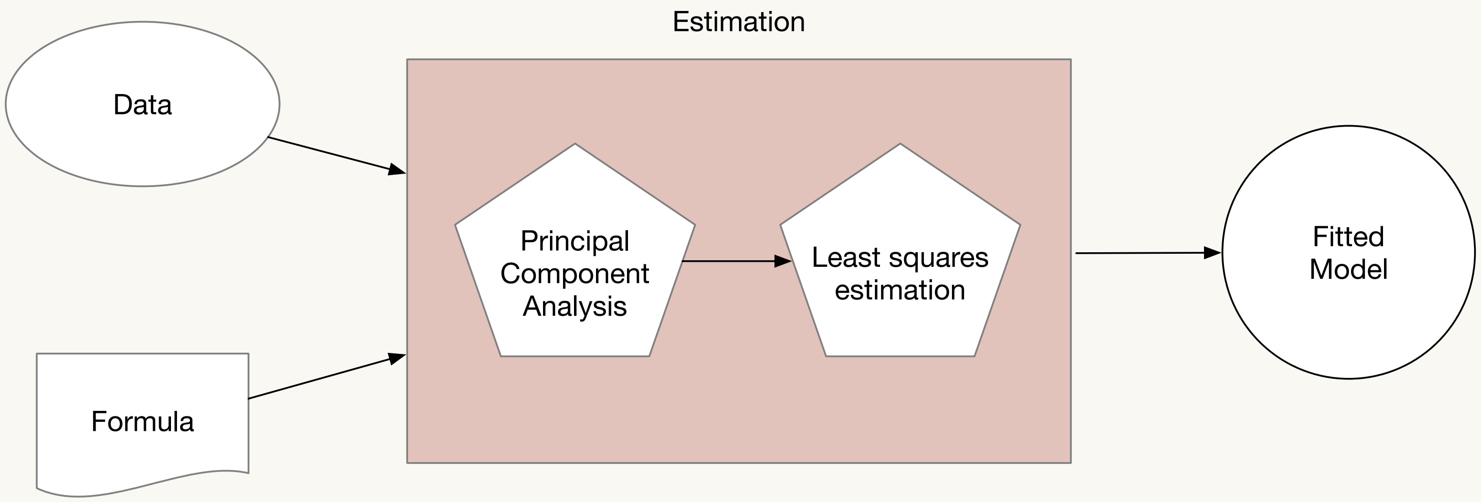
What do we consider the estimation part of this process?

Is it this?



Data
Formula
Principal Component Analysis
Estimation
Least squares estimation
Fitted Model

Or is it this?



Estimation
Data
Formula
Principal Component Analysis
Least squares estimation
Fitted Model

What's the difference?

It is easy to think that the model fit is the only estimation steps.

There are cases where this could go really wrong:

- Poor estimation of performance (buy treating the PCA parts as known)
- Selection bias in feature selection
- Information leakage

These problems are exacerbated as the preprocessors increase in complexity and/or effectiveness.

The model workflow

A *model workflow* is an object that combines a preprocessor and a model object.

Why?

Two reasons:

- It can help organize your work (in case you try a lot of models and preprocessors)
- It encapsulates all of the estimation parts in one object

You can't just estimate one part.

The model workflow in R

```r
library(tidymodels)
data(Chicago)

split <- initial_split(Chicago)
chicago_train <- training(split)
chicago_test <- testing(split)

reg_model <- linear_reg() %>% set_engine("lm")

pca_rec <- recipe(ridership ~ ., data = chicago_train) %>%
  step_date(date, features = c("dow", "month", "year")) %>%
  step_holiday(date) %>%
  update_role(date, new_role = "id") %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_pca(one_of(stations), num_comp = 10)

pca_lm_wflow <-
  workflow() %>%
  add_model(reg_model) %>%
  add_recipe(pca_rec)
```

The good and bad estimation

Bad approach leading to information leakage:

```
modeling_data <-
  prep(pca_rec, training = Chicago) %>%
  bake(new_data = NULL)

bad <- fit(reg_model, modeling_data)

predict(bad, chicago_test)
```

Much better:

```
good <- fit(pca_lm_wflow, chicago_train)

predict(good, chicago_test)
```

Many workflows

You can imagine a case where, for a new data set, we don't know what predictors, features, or model are most effective.

- A good example of this, for the Chicago data, is shown in our *Feature Engineering and Selection* book.

We might want to define a group of preprocessors to try in concert with numerous models.

- PCA versus PLS and other extraction methods.
- Simple filtering of highly correlated predictors.
- No extraction and use a regularized model.

and so on.

Where do I use a workflow set?

When would you use this?

It's good for cases where you are starting from scratch and need to fit (and/or tune) a lot of models.

It might be good for variable selection (see example at end).

Would you always want to do this?

Absolutely not. For well-defined problems, it is overkill.

I was hesitant to even create this package since it might be used inappropriately.

A workflow set

These are objects that contain multiple workflows.

They can be estimated, evaluates, and ranked with simple APIs.

Let's create one by crossing several recipes with two models for the Chicago data.

Example objects

```r
reg_model <- linear_reg() %>% set_engine("lm")

nnet_model <-
  mlp(penalty = tune(), hidden_units = tune(), epochs = tune()) %>%
  set_engine("nnet") %>%
  set_mode("regression")
```

```r
simple_rec <- recipe(ridership ~ ., data = chicago_train) %>%
  step_date(date, features = c("dow", "month", "year")) %>%
  step_holiday(date) %>%
  update_role(date, new_role = "id") %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors())

pca_rec <- simple_rec %>%
  step_pca(one_of(stations), num_comp = tune())

pls_rec <- simple_rec %>%
  step_pls(one_of(stations), outcome = "ridership", num_comp = tune())

filter_rec <- simple_rec %>%
  step_corr(one_of(stations), threshold = tune())
```

Creating the workflow set

```
chicago_wflows <-
  workflow_set(
    preproc = list(basic = simple_rec, pca = pca_rec, pls = pls_rec,
                   filtered = filter_rec),
    models = list(lm = reg_model, nnet = nnet_model)
  )
chicago_wflows
```

```
## # A workflow set/tibble: 8 x 4
##   wflow_id       info             option       result
##   <chr>          <list>           <list>       <list>
## 1 basic_lm       <tibble [1 × 4]> <wrkflw__ > <list [0]>
## 2 basic_nnet     <tibble [1 × 4]> <wrkflw__ > <list [0]>
## 3 pca_lm         <tibble [1 × 4]> <wrkflw__ > <list [0]>
## 4 pca_nnet       <tibble [1 × 4]> <wrkflw__ > <list [0]>
## 5 pls_lm         <tibble [1 × 4]> <wrkflw__ > <list [0]>
## 6 pls_nnet       <tibble [1 × 4]> <wrkflw__ > <list [0]>
## 7 filtered_lm    <tibble [1 × 4]> <wrkflw__ > <list [0]>
## 8 filtered_nnet <tibble [1 × 4]> <wrkflw__ > <list [0]>
```

Evaluating many models

```
# 43 resamples via rolling forecast origin resampling
chicago_rs <-
  sliding_period(
    chicago_train,
    date,
    period = "month",
    lookback = 12 * 12,  # a rolling 12-year data set for fitting
    assess_stop = 1      # evaluate the model on the first new month
  )

chicago_res <-
  chicago_wflows %>%
  workflow_map(resamples = chicago_rs,
               seed = 1,
               # Optional arguments to tune_grid()
               grid = 20,
               metrics = metric_set(rmse))
```

Results

```
chicago_res
```

```
## # A workflow set/tibble: 8 x 4
##   wflow_id      info              option       result
##   <chr>         <list>            <list>       <list>
## 1 basic_lm      <tibble [1 × 4]> <wrkflw__  > <rsmp[+]>
## 2 basic_nnet    <tibble [1 × 4]> <wrkflw__  > <tune[+]>
## 3 pca_lm        <tibble [1 × 4]> <wrkflw__  > <tune[+]>
## 4 pca_nnet      <tibble [1 × 4]> <wrkflw__  > <tune[+]>
## 5 pls_lm        <tibble [1 × 4]> <wrkflw__  > <tune[+]>
## 6 pls_nnet      <tibble [1 × 4]> <wrkflw__  > <tune[+]>
## 7 filtered_lm   <tibble [1 × 4]> <wrkflw__  > <tune[+]>
## 8 filtered_nnet <tibble [1 × 4]> <wrkflw__  > <tune[+]>
```

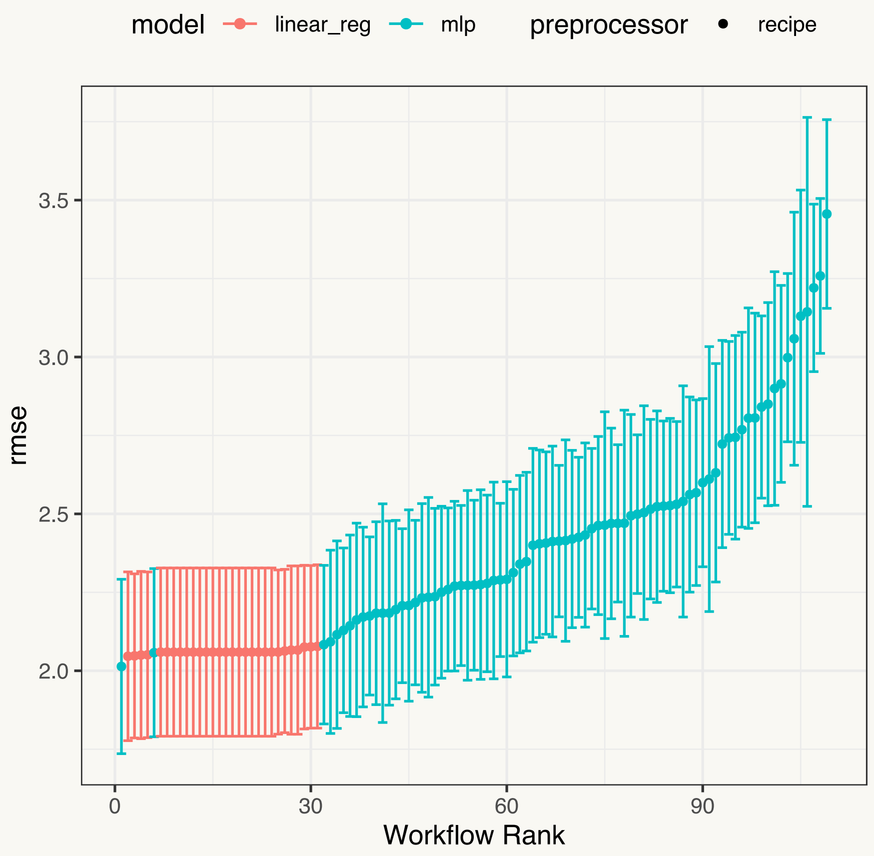
Rankings

```
chicago_res %>%
  rank_results(select_best = TRUE) %>%
  dplyr::select(wflow_id, .metric, mean, std_err, n, rank)
```

```
## # A tibble: 8 x 6
##   wflow_id      .metric  mean std_err     n  rank
##   <chr>         <chr>   <dbl>   <dbl> <int> <int>
## 1 pls_nnet      rmse     2.01   0.169    43     1
## 2 pls_lm        rmse     2.05   0.164    43     2
## 3 pca_lm        rmse     2.05   0.159    43     3
## 4 basic_lm      rmse     2.05   0.161    43     4
## 5 pca_nnet      rmse     2.06   0.163    43     5
## 6 filtered_lm   rmse     2.06   0.163    43     6
## 7 filtered_nnet rmse     2.08   0.154    43     7
## 8 basic_nnet    rmse     2.18   0.212    43     8
```

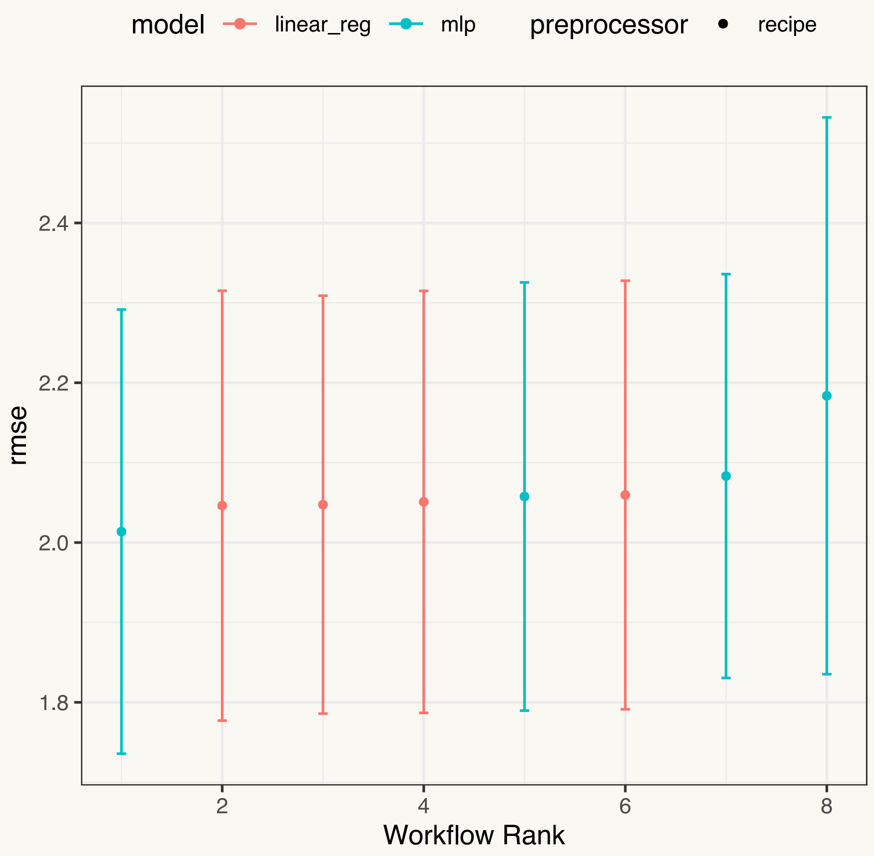
Rankings (everything)

```
chicago_res %>%
  autoplot()
```



Rankings (best of each workflow)

```r
chicago_res %>%
  autoplot(select_best = TRUE)
```



Passing individual options to models

The *minimum* correlation between station predictors is 0.88. Let's adjust the parameter range for the threshold parameter to be between 0.87 and .99.

We can create `dials` parameter objects to do this.

```r
lm_param <-
  pull_workflow(chicago_wflows, "filtered_lm") %>%
  parameters() %>%
  update(threshold = threshold(c(0.87, .99)))

nnet_param <-
  pull_workflow(chicago_wflows, "filtered_nnet") %>%
  parameters() %>%
  update(threshold = threshold(c(0.87, .99)))
```

Passing individual options to models

Now let's update the `option` file for those two workflows and refit:

```
filtered_res <-
  chicago_res %>%
  option_add(param_info = lm_param,   id = "filtered_lm") %>%
  option_add(param_info = nnet_param, id = "filtered_nnet") %>%
  filter(grepl("filtered", wflow_id)) %>%
  workflow_map(resamples = chicago_rs,
               seed = 1,
               # Optional arguments to tune_grid()
               grid = 20,
               metrics = metric_set(rmse))
```

```
## Warning: There are existing options that are being modified
##     filtered_lm: 'resamples', 'grid', 'metrics'
##     filtered_nnet: 'resamples', 'grid', 'metrics'
```

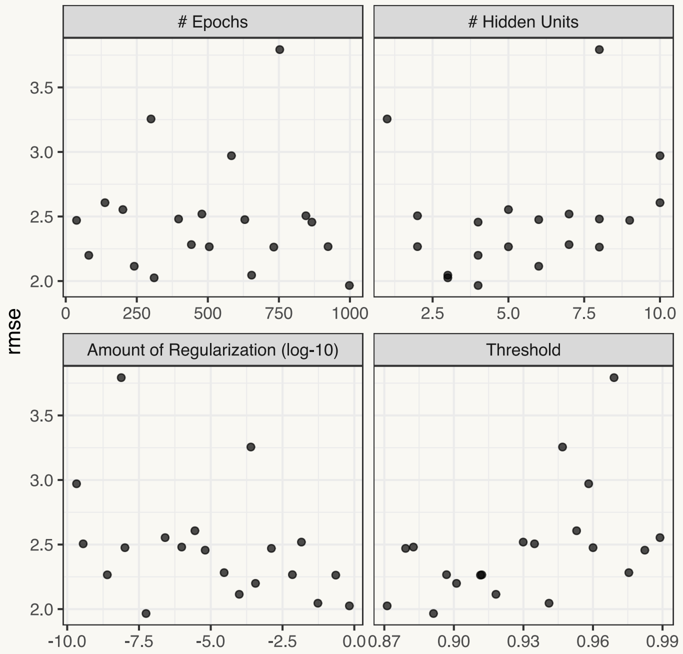
Updating the set

```r
updated_res <-
  chicago_res %>%
  filter(!grepl("filtered", wflow_id)) %>%
  bind_rows(filtered_res)

updated_res %>%
  rank_results(select_best = TRUE) %>%
  dplyr::select(wflow_id, .metric, mean, std_err, n, rank)
```

```
## # A tibble: 8 x 6
##   wflow_id      .metric  mean std_err     n  rank
##   <chr>         <chr>   <dbl>   <dbl> <int> <int>
## 1 filtered_nnet rmse     1.97   0.154    43     1
## 2 pls_nnet      rmse     2.01   0.169    43     2
## 3 filtered_lm   rmse     2.05   0.160    43     3
## 4 pls_lm        rmse     2.05   0.164    43     4
## 5 pca_lm        rmse     2.05   0.159    43     5
## 6 basic_lm      rmse     2.05   0.161    43     6
## 7 pca_nnet      rmse     2.06   0.163    43     7
## 8 basic_nnet    rmse     2.18   0.212    43     8
```

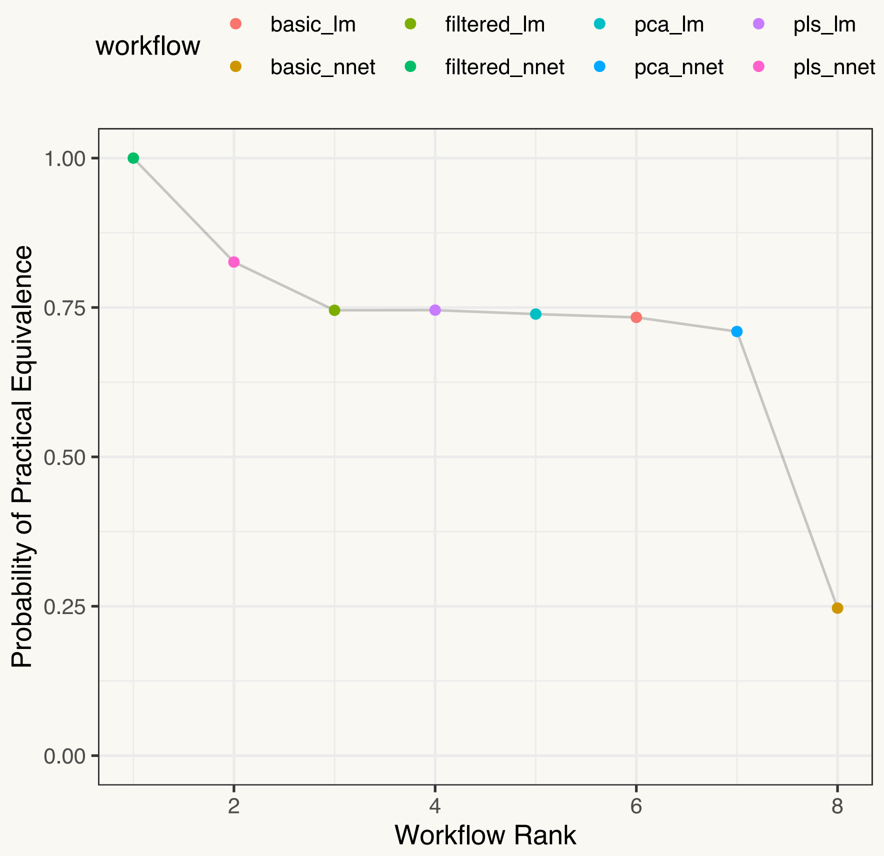
Plot the tuning results for a model:

```r
updated_res %>%
  autoplot(id = "filtered_nnet")
```



Bayesian analysis of the results

```r
library(tidyposterior)
rmse_res <-
  updated_res %>%
  perf_mod(
    iter = 5000,
    chains = 10,
    cores = 10,
    seed = 2,
    refresh = 0
  )

# Assess a difference of 150 riders in MAE
rmse_res %>%
  autoplot(type = "ROPE", size = 0.15)
```



Bonus: Fast screening using racing

Racing is an adaptive grid search method that only focuses on tuning parameter combinations that have a good probability of being the best results.

After an initial period, some tuning parameter combinations are removed if they have no chance of being best.

This reduces the overall number of model fits.

See Sections 13.4.4 and 15.4 of *Tidy Models with R*

Bonus: Fast screening using racing

To use it, map over `"tune_race_anova"` or `"tune_race_win_loss"` after loading the `finetune` package:

```
library(finetune)
racing_res <-
  chicago_res %>%
  workflow_map("tune_race_anova",
               resamples = chicago_rs,
               seed = 1,
               grid = 20,
               metrics = metric_set(rmse))
```

Bonus Bonus: Create an ensemble via stacking

```
library(stacks)
ens <-
  stacks() %>%
  add_candidates(updated_res) %>%
  blend_predictions() %>%
  fit_members()
```

This requires the devel version of `stacks` and you will have had to save the out-of-sample predictions for each model (via a `control` argument).

This is similar to the approach taken in [h2o AutoML](#).

See [`stacks.tidymodels.org`](#) and the upcoming [Chapter 20 of *Tidy Models with R*](#) for more information.

Model selection/variable assessment

Let's say that we want to assess the impact of each predictor in a model.

We can fit the model with and without the predictor and assess the effect.

```
data(biomass)
biomass <- biomass %>% select(-sample, -dataset)

f <- HHV ~ (.)^2
loo_f <- leave_var_out_formulas(f, data = biomass, full_model = TRUE)
length(loo_f)
```

```
## [1] 16
```

```
loo_f[["oxygen"]]
```

```
## HHV ~ carbon + hydrogen + nitrogen + sulfur + carbon:hydrogen +
##     carbon:nitrogen + carbon:sulfur + hydrogen:nitrogen + hydrogen:sulfur +
##     nitrogen:sulfur
## <environment: base>
```

Model selection/variable assessment

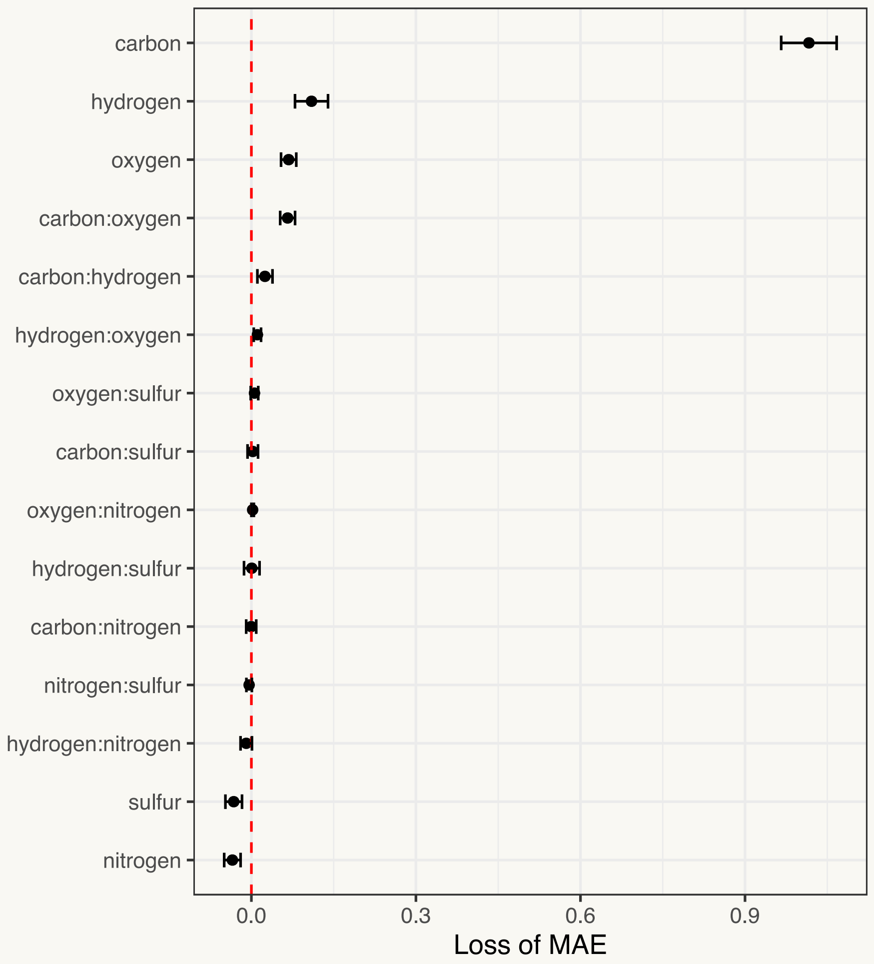
```r
set.seed(3)
boots <- bootstraps(biomass)
biomass_models <-
  workflow_set(
    preproc = loo_f,
    models = list(lm = reg_model)
  ) %>%
  workflow_map(
    "fit_resamples",
    resamples = boots,
    metrics = metric_set(mae),
    seed = 4
  )

mae_stats <-
  collect_metrics(biomass_models,
                  summarize = FALSE) %>%
  dplyr::select(wflow_id, .estimate, id)

full_model <-
  filter(mae_stats,
         wflow_id == "everything_lm") %>%
  dplyr::select(id, full_model = .estimate)
```

Estimate the mean absolute error for various predictor subsets.

If the predictor is important, removing it makes performance worse.

Which model terms are important?

```
left_join(mae_stats, full_model, by = "id") %>%
  mutate(loss = .estimate - full_model) %>%
  filter(wflow_id != "everything_lm") %>%
  mutate(term = gsub("_lm", "", wflow_id)) %>%
  nest_by(term) %>%
  mutate(stats = list(t.test(loss ~ 1, data = data))) %>%
  summarise(tidy(stats), .groups = "drop") %>%
  ggplot(aes(x = estimate, y = reorder(term, estimate)))+
  geom_point() +
  geom_vline(xintercept = 0, lty = 2, col = "red") +
  geom_errorbar(aes(xmin = conf.low, xmax = conf.high),
                width = .25) +
  labs(x = "Loss of MAE", y = NULL)
```



Summary

Workflow sets can be very helpful when you need to fit a lot of models or want to create a stacking ensemble.

They can also be nice if you want to track/collate/rank results over models that have already been fit (via `as_workflow_set()`).

I don't see them as a staple of regular tidymodels analyses.