

Getting Started With stacks

In this article, we'll be working through an example of the workflow of model stacking with the `stacks` package. At a high level, the workflow looks something like this:

1. Define candidate ensemble members using functionality from `rsample`, `parsnip`, `workflows`, `recipes`, and `tune`
2. Initialize a `data_stack` object with `stacks()`
3. Iteratively add candidate ensemble members to the `data_stack` with `add_candidates()`
4. Evaluate how to combine their predictions with `blend_predictions()`
5. Fit candidate ensemble members with non-zero stacking coefficients with `fit_members()`
6. Predict on new data with `predict()` !

The package is closely integrated with the rest of the functionality in `tidymodels`—we'll load those packages as well, in addition to some tidyverse packages to evaluate our results later on.

```
library (https://rdrr.io/r/base/library.html)(tidymodels (https://tidymodels.ti
library (https://rdrr.io/r/base/library.html)(stacks)
library (https://rdrr.io/r/base/library.html)(dplyr (https://dplyr.tidyverse.or
library (https://rdrr.io/r/base/library.html)(purrr (http://purrr.tidyverse.org
```

In this example, we'll make use of the `tree_frogs` data exported with `stacks`, giving experimental results on hatching behavior of red-eyed tree frog embryos!

Red-eyed tree frog (RETF) embryos can hatch earlier than their normal 7ish days if they detect potential predator threat. Researchers wanted to determine how, and when, these tree frog embryos were able to detect stimulus from their environment. To do so, they subjected the embryos at varying developmental stages to “predator stimulus” by jiggling the embryos with a

blunt probe. Beforehand, though some of the embryos were treated with gentamicin, a compound that knocks out their lateral line (a sensory organ.) Researcher Julie Jung and her crew found that these factors inform whether an embryo hatches prematurely or not!

We'll start out with predicting latency (i.e. time to hatch) based on other attributes. We'll need to filter out NAs (i.e. cases where the embryo did not hatch) first.

```
data (https://rdr.io/r/utis/data.html)("tree_frogs")

# subset the data
tree_frogs <- tree_frogs %>%
  filter (https://dplyr.tidyverse.org/reference/filter.html)(!is.na (https://rd
  select (https://dplyr.tidyverse.org/reference/select.html)(-c (https://rdr.i
```

Taking a quick look at the data, it seems like the hatch time is pretty closely related to some of our predictors!

```
library (https://rdr.io/r/base/library.html)(ggplot2 (http://ggplot2.tidyverse

ggplot (https://ggplot2.tidyverse.org/reference/ggplot.html)(tree_frogs) +
  aes (https://ggplot2.tidyverse.org/reference/aes.html)(x = age, y = latency,
  geom_point (https://ggplot2.tidyverse.org/reference/geom_point.html)() +
  labs (https://ggplot2.tidyverse.org/reference/labs.html)(x = "Embryo Age (s)"
```



Let's give this a go!

DEFINE CANDIDATE ENSEMBLE MEMBERS

At the highest level, ensembles are formed from *model definitions*. In this package, model definitions are an instance of a minimal `workflow` (<https://workflows.tidymodels.org/>), containing a *model specification* (as defined in the `parsnip` (<https://parsnip.tidymodels.org/>) package) and, optionally, a *preprocessor* (as defined in the `recipes` (<https://recipes.tidymodels.org/>) package). Model definitions specify the form of candidate ensemble members.

*Model
Definitions*

KNN Model

Linear Regression
Model

Support Vector
Machine Model

Defining the constituent model definitions is undoubtedly the longest part of building an ensemble with `stacks`. If you're familiar with tidymodels "proper," you're probably fine to skip this section, keeping a few things in mind:

- You'll need to save the assessment set predictions and workflow utilized in your `tune_grid()`, `tune_bayes()`, or `fit_resamples()` objects by setting the `control`

arguments `save_pred = TRUE` and `save_workflow = TRUE`. Note the use of the `control_stack_*`() convenience functions below!

- Each model definition must share the same `rsample` `rset` object.

We'll first start out with splitting up the training data, generating resamples, and setting some options that will be used by each model definition.

```
# some setup: resampling and a basic recipe
set.seed (https://rdrr.io/r/base/Random.html)(1)
tree_frogs_split <- initial_split (https://rsample.tidymodels.org/reference/initial_split.html)
tree_frogs_train <- training (https://rsample.tidymodels.org/reference/initial_split.html)
tree_frogs_test  <- testing (https://rsample.tidymodels.org/reference/initial_split.html)

set.seed (https://rdrr.io/r/base/Random.html)(1)
folds <- rsample::vfold_cv (https://rsample.tidymodels.org/reference/vfold_cv.html)

tree_frogs_rec <-
  recipe (https://recipes.tidymodels.org/reference/recipe.html)(latency ~ ., data = tree_frogs_train)

metric <- metric_set (https://rdrr.io/pkg/yardstick/man/metric_set.html)(rmse)
```

Tuning and fitting results for use in ensembles need to be fitted with the control arguments `save_pred = TRUE` and `save_workflow = TRUE`—these settings ensure that the assessment set predictions, as well as the workflow used to fit the resamples, are stored in the resulting object. For convenience, `stacks` supplies some `control_stack_*`() functions to generate the appropriate objects for you.

In this example, we'll be working with `tune_grid()` and `fit_resamples()` from the `tune` package, so we will use the following control settings:

```
ctrl_grid <- control_stack_grid (../reference/control_stack_grid.html)()
ctrl_res  <- control_stack_resamples (../reference/control_stack_grid.html)()
```

We'll define three different model definitions to try to predict time to hatch—a K-nearest neighbors model (with hyperparameters to tune), a linear model, and a support vector machine model (again, with hyperparameters to tune).

Starting out with K-nearest neighbors, we begin by creating a `parsnip` model specification:

```
# create a model definition
knn_spec <-
  nearest_neighbor (https://parsnip.tidymodels.org/reference/nearest_neighbor.h
    mode = "regression",
    neighbors = tune (https://tune.tidymodels.org/reference/tune.html)("k")
  ) %>%
  set_engine (https://parsnip.tidymodels.org/reference/set_engine.html)("kknn")

knn_spec
#> K-Nearest Neighbor Model Specification (regression)
#>
#> Main Arguments:
#>   neighbors = tune("k")
#>
#> Computational engine: kknn
```

Note that, since we are tuning over several possible numbers of neighbors, this model specification defines multiple model configurations. The specific form of those configurations will be determined when specifying the grid search in `tune_grid()`.

From here, we extend the basic recipe defined earlier to fully specify the form of the design matrix for use in a K-nearest neighbors model:

```
# extend the recipe
knn_rec <-
  tree_frogs_rec %>%
  step_dummy (https://recipes.tidymodels.org/reference/step_dummy.html)(all_nom
  step_zv (https://recipes.tidymodels.org/reference/step_zv.html)(all_predictor
  step_meanimpute (https://recipes.tidymodels.org/reference/step_meanimpute.htm
  step_normalize (https://recipes.tidymodels.org/reference/step_normalize.html)

knn_rec
#> Data Recipe
#>
#> Inputs:
#>
#>      role #variables
#> outcome      1
#> predictor      4
#>
#> Operations:
#>
#> Dummy variables from all_nominal()
#> Zero variance filter on all_predictors()
#> Mean Imputation for all_numeric()
#> Centering and scaling for all_numeric()
```

Starting with the basic recipe, we convert categorical variables to dummy variables, remove column with only one observation, impute missing values in numeric variables using the mean, and normalize numeric predictors. Pre-processing instructions for the remaining models are defined similarly.

Now, we combine the model specification and pre-processing instructions defined above to form a workflow object:

```
# add both to a workflow
knn_wflow <-
  workflow (https://workflows.tidymodels.org//reference/workflow.html)() %>%
  add_model (https://workflows.tidymodels.org//reference/add_model.html)(knn_sp)
  add_recipe (https://workflows.tidymodels.org//reference/add_recipe.html)(knn_

knn_wflow
#> == Workflow ==
#> Preprocessor: Recipe
#> Model: nearest_neighbor()
#>
#> — Preprocessor —
#> 4 Recipe Steps
#>
#> • step_dummy()
#> • step_zv()
#> • step_meanimpute()
#> • step_normalize()
#>
#> — Model —
#> K-Nearest Neighbor Model Specification (regression)
#>
#> Main Arguments:
#>   neighbors = tune("k")
#>
#> Computational engine: kkn
```

Finally, we can make use of the workflow, training set resamples, metric set, and control object to tune our hyperparameters. Using the `grid` argument, we specify that we would like to optimize over four possible values of `k` using a grid search.

```

# tune k and fit to the 5-fold cv
set.seed (https://rdrr.io/r/base/Random.html)(2020)
knn_res <-
  tune_grid (https://tune.tidymodels.org/reference/tune_grid.html)(
    knn_wflow,
    resamples = folds,
    metrics = metric,
    grid = 4,
    control = ctrl_grid
  )
#> Loading required package: scales
#>
#> Attaching package: 'scales'
#> The following object is masked from 'package:purrr':
#>
#>     discard
#>
#> Attaching package: 'rlang'
#> The following objects are masked from 'package:purrr':
#>
#>     %@%, as_function, flatten, flatten_chr, flatten_dbl, flatten_int,
#>     flatten_lgl, flatten_raw, invoke, list_along, modify, prepend,
#>     splice
#>
#> Attaching package: 'vctrs'
#> The following object is masked from 'package:tibble':
#>
#>     data_frame
#> The following object is masked from 'package:dplyr':
#>
#>     data_frame
#> The workflow being saved contains a recipe, which is 0.23 Mb in memory. If t

knn_res
#> # Tuning results
#> # 5-fold cross-validation
#> # A tibble: 5 x 5
#>   splits          id    .metrics          .notes          .predictions
#>   <list>          <chr> <list>          <list>          <list>
#> 1 <split [343/86]> Fold1 <tibble [4 x 5]> <tibble [0 x 1]> <tibble [344 x 5]
#> 2 <split [343/86]> Fold2 <tibble [4 x 5]> <tibble [0 x 1]> <tibble [344 x 5]

```



```
#> 3 <split [343/86]> Fold3 <tibble [4 × 5]> <tibble [0 × 1]> <tibble [344 × 5]>
#> 4 <split [343/86]> Fold4 <tibble [4 × 5]> <tibble [0 × 1]> <tibble [344 × 5]>
#> 5 <split [344/85]> Fold5 <tibble [4 × 5]> <tibble [0 × 1]> <tibble [340 × 5]>
```

This `knn_res` object fully specifies the candidate members, and is ready to be included in a `stacks` workflow.

Now, specifying the linear model, note that we are not optimizing over any hyperparameters. Thus, we use the `fit_resamples()` function rather than `tune_grid()` or `tune_bayes()` when fitting to our resamples.

```

# create a model definition
lin_reg_spec <-
  linear_reg (https://parsnip.tidymodels.org/reference/linear_reg.html)() %>%
  set_engine (https://parsnip.tidymodels.org/reference/set_engine.html)("lm")

# extend the recipe
lin_reg_rec <-
  tree_frogs_rec %>%
  step_dummy (https://recipes.tidymodels.org/reference/step_dummy.html)(all_nom
  step_zv (https://recipes.tidymodels.org/reference/step_zv.html)(all_predictor

# add both to a workflow
lin_reg_wflow <-
  workflow (https://workflows.tidymodels.org//reference/workflow.html)() %>%
  add_model (https://workflows.tidymodels.org//reference/add_model.html)(lin_re
  add_recipe (https://workflows.tidymodels.org//reference/add_recipe.html)(lin_

# fit to the 5-fold cv
set.seed (https://rdrr.io/r/base/Random.html)(2020)
lin_reg_res <-
  fit_resamples (https://tune.tidymodels.org/reference/fit_resamples.html)(
    lin_reg_wflow,
    resamples = folds,
    metrics = metric,
    control = ctrl_res
  )
#> The workflow being saved contains a recipe, which is 0.22 Mb in memory. If t

lin_reg_res
#> # Resampling results
#> # 5-fold cross-validation
#> # A tibble: 5 x 5
#>   splits          id   .metrics      .notes      .predictions
#>   <list>         <chr> <list>      <list>      <list>
#> 1 <split [343/86]> Fold1 <tibble [1 x 4]> <tibble [0 x 1]> <tibble [86 x 4]>
#> 2 <split [343/86]> Fold2 <tibble [1 x 4]> <tibble [0 x 1]> <tibble [86 x 4]>
#> 3 <split [343/86]> Fold3 <tibble [1 x 4]> <tibble [0 x 1]> <tibble [86 x 4]>
#> 4 <split [343/86]> Fold4 <tibble [1 x 4]> <tibble [0 x 1]> <tibble [86 x 4]>
#> 5 <split [344/85]> Fold5 <tibble [1 x 4]> <tibble [0 x 1]> <tibble [85 x 4]>

```

Finally, putting together the model definition for the support vector machine:

```
# create a model definition
svm_spec <-
  svm_rbf (https://parsnip.tidymodels.org/reference/svm_rbf.html)(
    cost = tune (https://tune.tidymodels.org/reference/tune.html)("cost"),
    rbf_sigma = tune (https://tune.tidymodels.org/reference/tune.html)("sigma")
  ) %>%
  set_engine (https://parsnip.tidymodels.org/reference/set_engine.html)("kernla
  set_mode (https://parsnip.tidymodels.org/reference/set_args.html)("regression

# extend the recipe
svm_rec <-
  tree_frogs_rec %>%
  step_dummy (https://recipes.tidymodels.org/reference/step_dummy.html)(all_nom
  step_zv (https://recipes.tidymodels.org/reference/step_zv.html)(all_predictor
  step_meanimpute (https://recipes.tidymodels.org/reference/step_meanimpute.htm
  step_corr (https://recipes.tidymodels.org/reference/step_corr.html)(all_predi
  step_normalize (https://recipes.tidymodels.org/reference/step_normalize.html)

# add both to a workflow
svm_wflow <-
  workflow (https://workflows.tidymodels.org//reference/workflow.html)() %>%
  add_model (https://workflows.tidymodels.org//reference/add_model.html)(svm_sp
  add_recipe (https://workflows.tidymodels.org//reference/add_recipe.html)(svm_

# tune cost and sigma and fit to the 5-fold cv
set.seed (https://rdrr.io/r/base/Random.html)(2020)
svm_res <-
  tune_grid (https://tune.tidymodels.org/reference/tune_grid.html)(
    svm_wflow,
    resamples = folds,
    grid = 6,
    metrics = metric,
    control = ctrl_grid
  )
#>
#> Attaching package: 'kernlab'
#> The following object is masked from 'package:scales':
#>
#>     alpha
#> The following object is masked from 'package:ggplot2':
#>
```

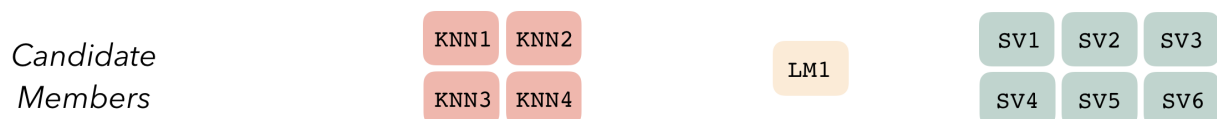
```

#>      alpha
#> The following object is masked from 'package:purrr':
#>
#>      cross
#> The workflow being saved contains a recipe, which is 0.23 Mb in memory. If t

svm_res
#> # Tuning results
#> # 5-fold cross-validation
#> # A tibble: 5 x 5
#>   splits          id    .metrics      .notes      .predictions
#>   <list>         <chr> <list>      <list>      <list>
#> 1 <split [343/86]> Fold1 <tibble [6 × 6]> <tibble [0 × 1]> <tibble [516 × 6]
#> 2 <split [343/86]> Fold2 <tibble [6 × 6]> <tibble [0 × 1]> <tibble [516 × 6]
#> 3 <split [343/86]> Fold3 <tibble [6 × 6]> <tibble [0 × 1]> <tibble [516 × 6]
#> 4 <split [343/86]> Fold4 <tibble [6 × 6]> <tibble [0 × 1]> <tibble [516 × 6]
#> 5 <split [344/85]> Fold5 <tibble [6 × 6]> <tibble [0 × 1]> <tibble [510 × 6]

```

Altogether, we've created three model definitions, where the K-nearest neighbors model definition specifies 4 model configurations, the linear regression specifies 1, and the support vector machine specifies 6.



With these three model definitions fully specified, we are ready to begin stacking these model configurations. (Note that, in most applied settings, one would likely specify many more than 11 candidate members.)

PUTTING TOGETHER A STACK

The first step to building an ensemble with stacks is to create a `data_stack` object—in this package, data stacks are tibbles (with some extra attributes) that contain the assessment set predictions for each candidate ensemble member.

Data Stack
(data_stack < tibble)

outcome	KNN1	KNN2	KNN3	KNN4	LM1	SV1	SV2	SV3	SV4	SV5	SV6
...
...
...

We can initialize a data stack using the `stacks()` function.

```
stacks (../reference/stacks.html)()
#> # A data stack with 0 model definitions and 0 candidate members.
```

The `stacks()` function works sort of like the `ggplot()` constructor from `ggplot2`—the function creates a basic structure that the object will be built on top of—except you'll pipe the outputs rather than adding them with `+`.

The `add_candidates()` function adds ensemble members to the stack.

```
tree_frogs_data_st <-
  stacks (../reference/stacks.html)() %>%
  add_candidates (../reference/add_candidates.html)(knn_res) %>%
  add_candidates (../reference/add_candidates.html)(lin_reg_res) %>%
  add_candidates (../reference/add_candidates.html)(svm_res)

tree_frogs_data_st
#> # A data stack with 3 model definitions and 11 candidate members:
#> #   knn_res: 4 model configurations
#> #   lin_reg_res: 1 model configuration
#> #   svm_res: 6 model configurations
#> # Outcome: latency (numeric)
```

As mentioned before, under the hood, a `data_stack` object is really just a tibble with some extra attributes. Checking out the actual data:

```
as_tibble (https://tibble.tidyverse.org/reference/as_tibble.html)(tree_frogs_da
#> # A tibble: 429 x 12
#>   latency knn_res_1_1 knn_res_1_2 knn_res_1_3 knn_res_1_4 lin_reg_res_1_1
#>   <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
#> 1     360     -0.343     -0.427     -0.469     -0.478     194.
#> 2     106     -0.336     -0.423     -0.438     -0.444     123.
#> 3     180     -0.343     -0.427     -0.469     -0.478     138.
#> 4      60     -0.350     -0.385     -0.401     -0.407     122.
#> 5      39     -0.251     -0.357     -0.427     -0.441      82.9
#> 6     214     -0.420     -0.455     -0.505     -0.515     134.
#> 7      50     -0.336     -0.423     -0.438     -0.444      37.2
#> 8     224     -0.336     -0.423     -0.438     -0.444     125.
#> 9      63     -0.420     -0.455     -0.505     -0.515      40.3
#> 10     33     -0.336     -0.423     -0.438     -0.444      38.3
#> # ... with 419 more rows, and 6 more variables: svm_res_1_6 <dbl>,
#> #   svm_res_1_5 <dbl>, svm_res_1_3 <dbl>, svm_res_1_1 <dbl>, svm_res_1_2 <db
#> #   svm_res_1_4 <dbl>
```

The first column gives the first response value, and the remaining columns give the assessment set predictions for each ensemble member. Since we're in the regression case, there's only one column per ensemble member. In classification settings, there are as many columns as there are levels of the outcome variable per candidate ensemble member.

That's it! We're now ready to evaluate how it is that we need to combine predictions from each candidate ensemble member.

FIT THE STACK

The outputs from each of these candidate ensemble members are highly correlated, so the `blend_predictions()` function performs regularization to figure out how we can combine the outputs from the stack members to come up with a final prediction.

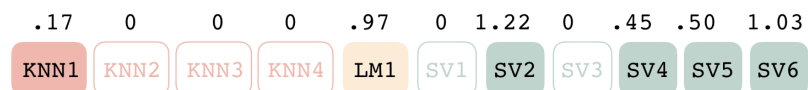
```

tree_frogs_model_st <-
  tree_frogs_data_st %>%
  blend_predictions (../reference/blend_predictions.html)()
#> Loading required package: Matrix
#>
#> Attaching package: 'Matrix'
#> The following objects are masked from 'package:tidyr':
#>
#>     expand, pack, unpack
#> Loaded glmnet 4.0-2

```

The `blend_predictions` function determines how member model output will ultimately be combined in the final prediction by fitting a LASSO model on the data stack, predicting the true assessment set outcome using the predictions from each of the candidate members. Candidates with nonzero stacking coefficients become members.

*Stacking
Coefficients*

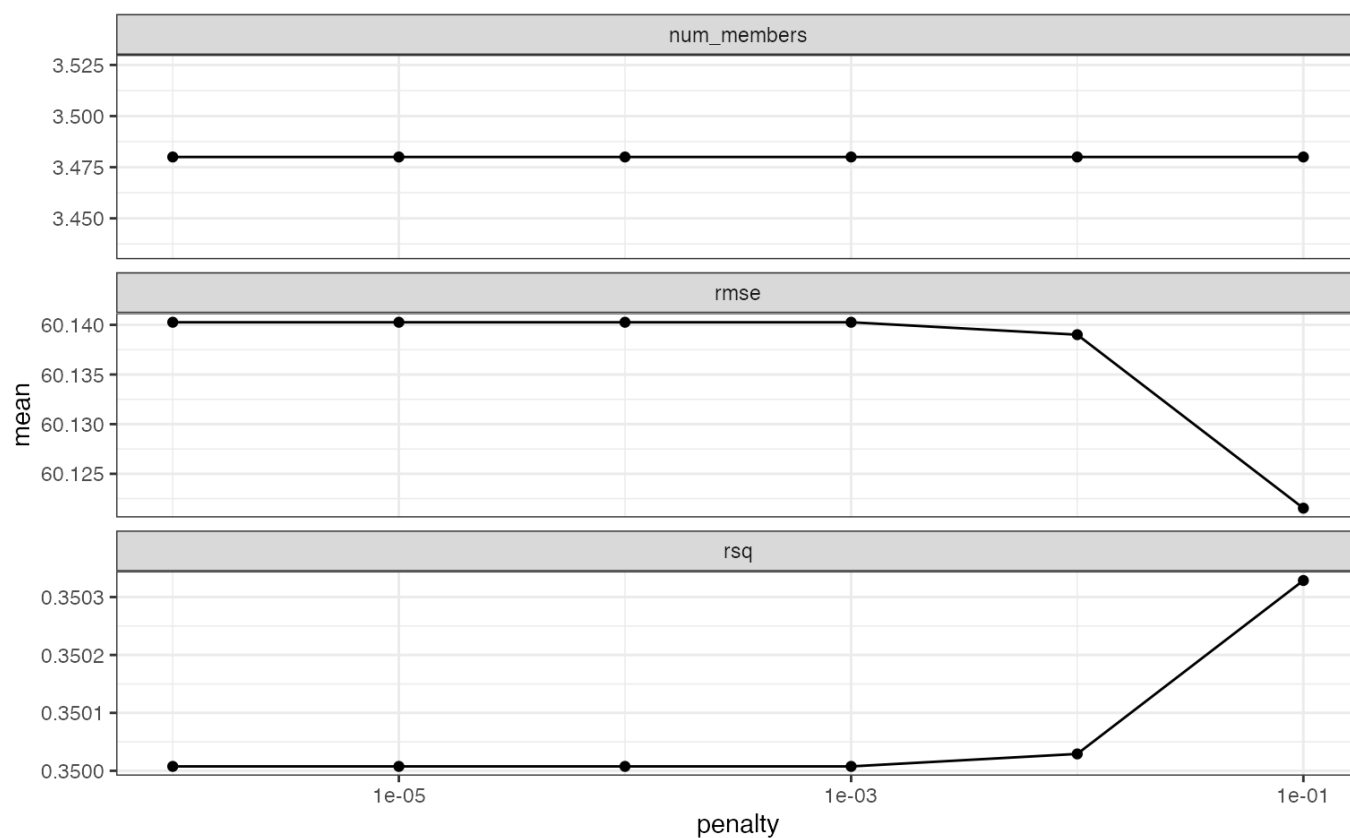


To make sure that we have the right trade-off between minimizing the number of members and optimizing performance, we can use the `autoplot()` method:

```

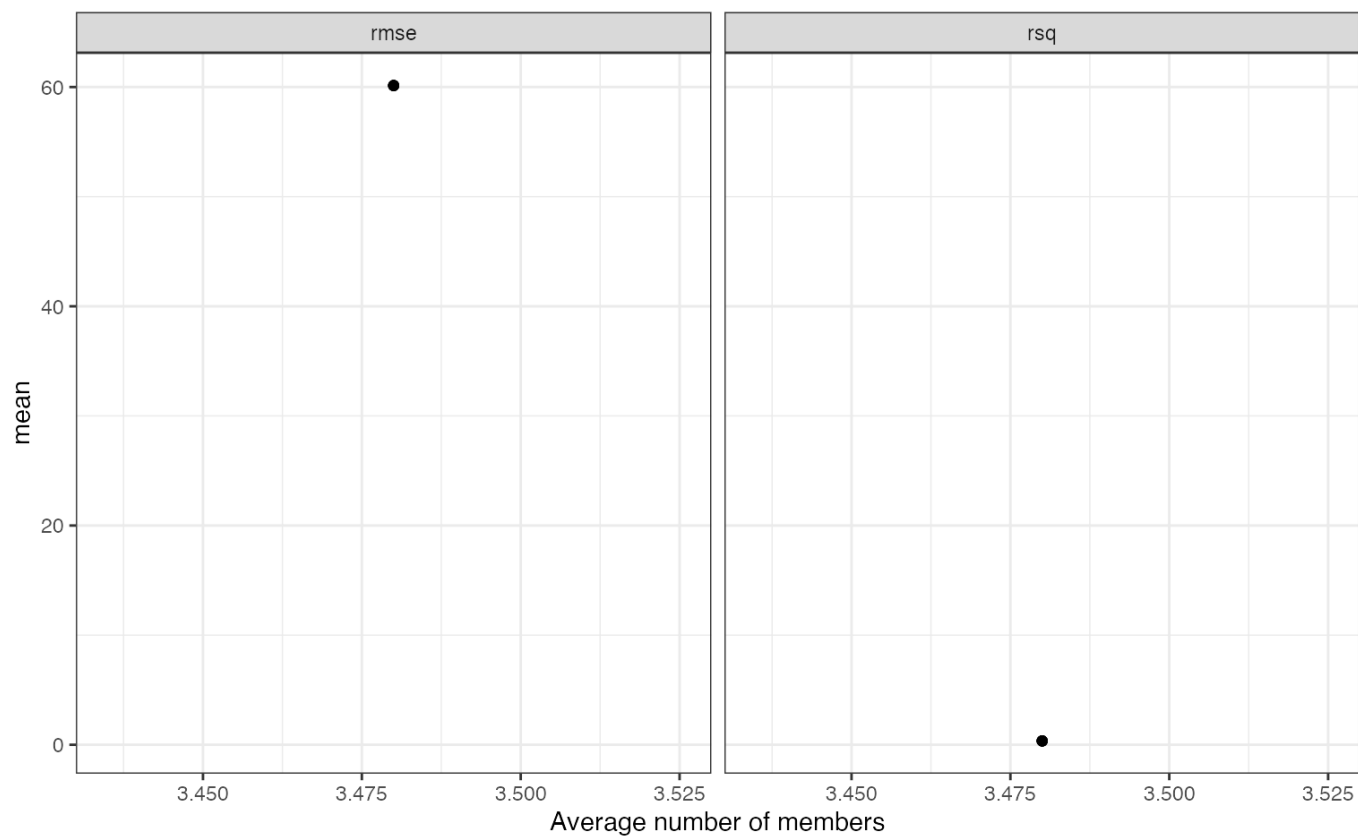
theme_set (https://ggplot2.tidyverse.org/reference/theme_get.html)(theme_bw (ht
autoplot (https://ggplot2.tidyverse.org/reference/autoplot.html)(tree_frogs_mod

```

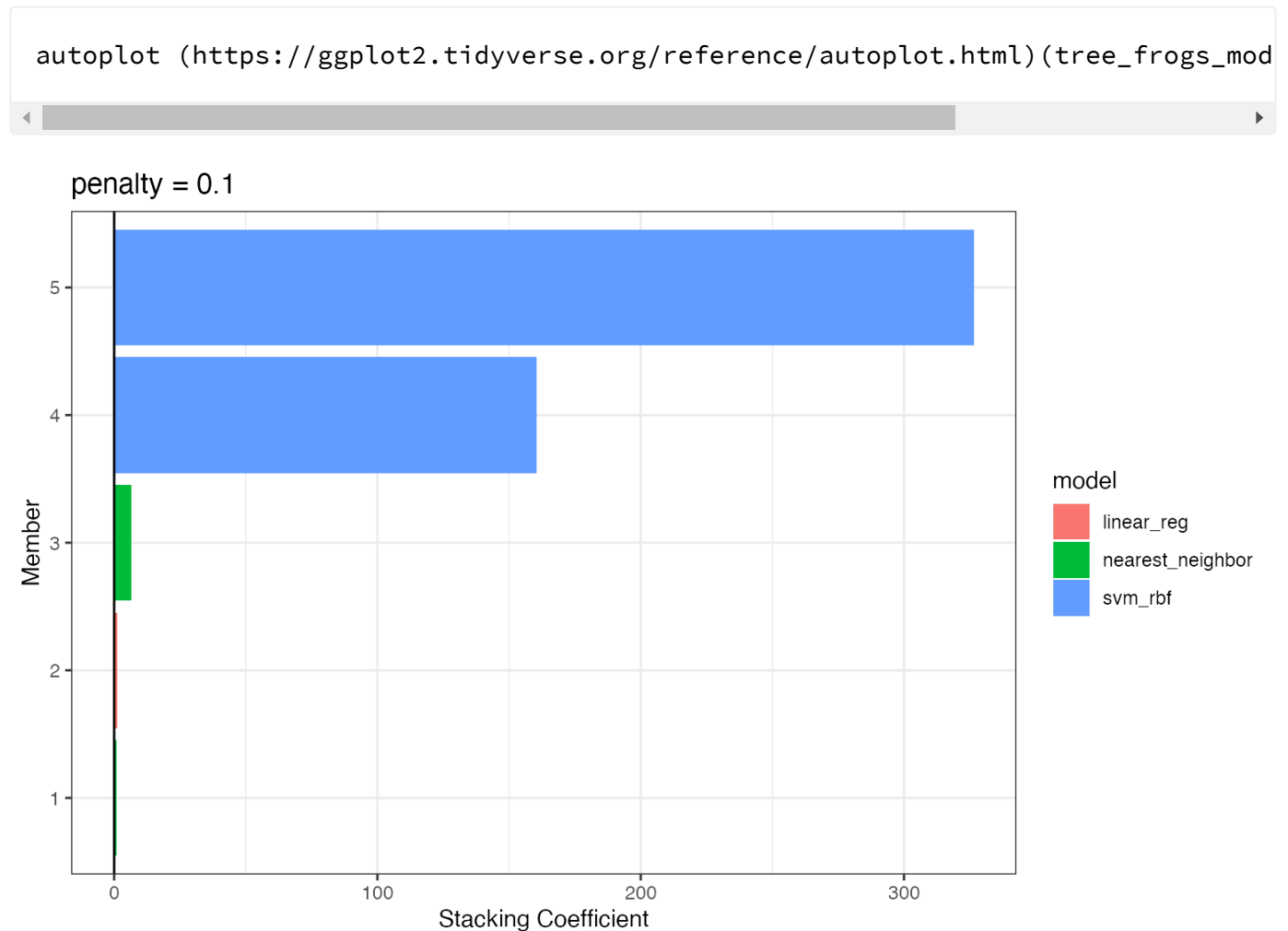



To show the relationship more directly:

```
autoplot (https://ggplot2.tidyverse.org/reference/autoplot.html)(tree_frogs_mod
```



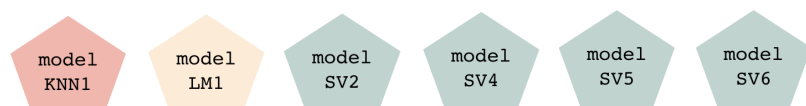
If these results were not good enough, `blend_predictions()` could be called again with different values of `penalty`. As it is, `blend_predictions()` picks the penalty parameter with the numerically optimal results. To see the top results:



Now that we know how to combine our model output, we can fit the candidates with non-zero stacking coefficients on the full training set.

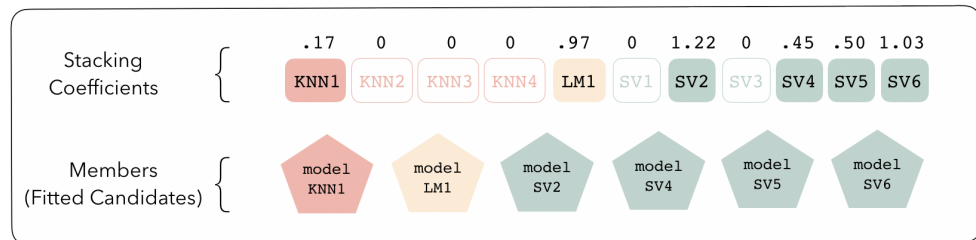
```
tree_frogs_model_st <-
  tree_frogs_model_st %>%
  fit_members (../reference/fit_members.html)()
```

Members



Model stacks can be thought of as a group of fitted member models and a set of instructions on how to combine their predictions.

Model Stack
(model_stack < list)



To identify which model configurations were assigned what stacking coefficients, we can make use of the `collect_parameters()` function:

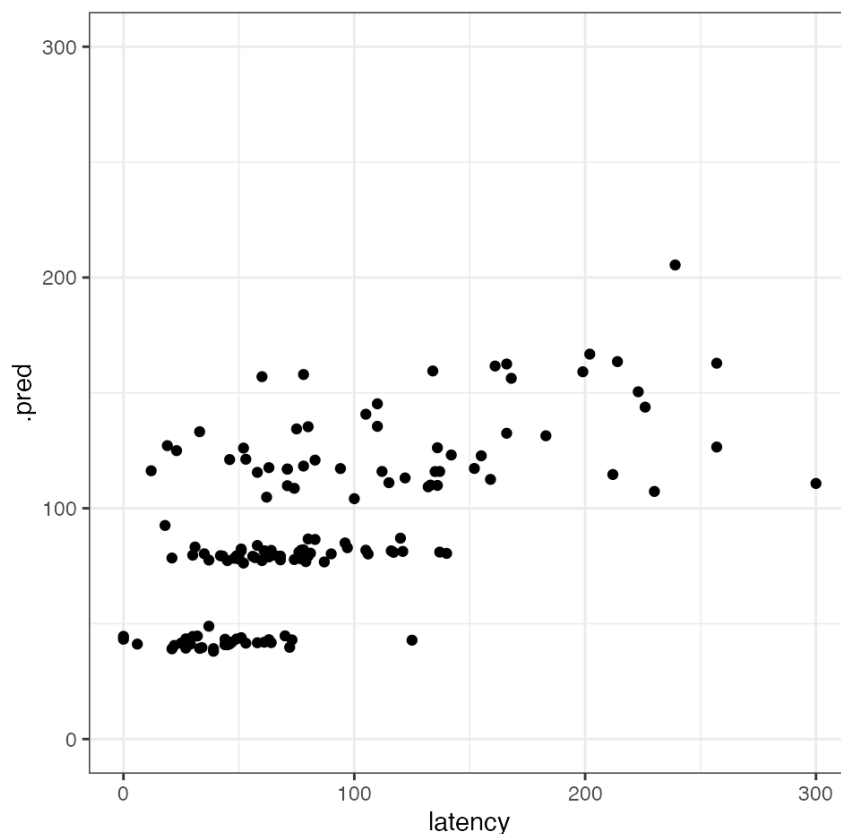
```
collect_parameters (../reference/collect_parameters.html)(tree_frogs_model_st,
#> # A tibble: 6 x 4
#>   member      cost    sigma  coef
#>   <chr>      <dbl>    <dbl> <dbl>
#> 1 svm_res_1_1  0.510    4.24e- 4  326.
#> 2 svm_res_1_2  4.57     1.29e- 3  160.
#> 3 svm_res_1_3  0.0378   8.18e- 1    0
#> 4 svm_res_1_4 14.1      8.16e- 7    0
#> 5 svm_res_1_5  0.0126   1.25e- 8    0
#> 6 svm_res_1_6  0.00154  1.63e-10    0
```

This object is now ready to predict with new data!

```
tree_frogs_test <-
  tree_frogs_test %>%
  bind_cols (https://dplyr.tidyverse.org/reference/bind.html)(predict (https://
```

Juxtaposing the predictions with the true data:

```
ggplot (https://ggplot2.tidyverse.org/reference/ggplot.html)(tree_frogs_test) +
  aes (https://ggplot2.tidyverse.org/reference/aes.html)(x = latency,
    y = .pred) +
  geom_point (https://ggplot2.tidyverse.org/reference/geom_point.html)() +
  coord_obs_pred (https://tune.tidymodels.org/reference/coord_obs_pred.html)()
```



Looks like our predictions were pretty strong! How do the stacks predictions perform, though, as compared to the members' predictions? We can use the `type = "members"` argument to generate predictions from each of the ensemble members.

```
member_preds <-  
  tree_frogs_test %>%  
    select (https://dplyr.tidyverse.org/reference/select.html)(latency) %>%  
    bind_cols (https://dplyr.tidyverse.org/reference/bind.html)(predict (https://
```

Now, evaluating the root mean squared error from each model:

```
map_dfr (https://purrr.tidyverse.org/reference/map.html)(member_preds, rmse, tr
  mutate (https://dplyr.tidyverse.org/reference/mutate.html)(member = colnames
#> # A tibble: 7 x 4
#>   .metric .estimator .estimate member
#>   <chr>    <chr>          <dbl> <chr>
#> 1 rmse    standard           0    latency
#> 2 rmse    standard        44.5    .pred
#> 3 rmse    standard       104.    knn_res_1_1
#> 4 rmse    standard       104.    knn_res_1_2
#> 5 rmse    standard       44.7    lin_reg_res_1_1
#> 6 rmse    standard       104.    svm_res_1_1
#> 7 rmse    standard       104.    svm_res_1_2
```

As we can see, the stacked ensemble outperforms each of the member models, though is closely followed by one of its members.

Voila! You've now made use of the stacks package to predict red-eyed tree frog embryo hatching using a stacked ensemble! The full visual outline for these steps can be found here (<https://github.com/tidymodels/stacks/blob/main/inst/figs/outline.png>).

stacks is a part of the **tidymodels** ecosystem, a collection of modeling packages designed with common APIs and a shared philosophy.

Developed by Simon Couch, Max Kuhn. Site built by pkgdown (<https://pkgdown.r-lib.org>).