Stacks

Nate Wells

Math 243: Stat Learning

November 29th, 2021

Outline

In today's class, we will...

- Discuss `stacks` package for implementing ensemble learning with `tidymodels`

Section 1

Intro to stacks

What is stacks?

stacks is an R package for ensemble learning compatible with the tidymodels framework, developed by Simon Couch ’21 and Max Kuhn.

General Procedure

1. Define candidate models using the `tidymodels` framework (`rsample`, `parsnip`, `workflow`, `recipe`, `tune`)
2. Initialize a `data_stack` object with `stacks()`
3. Iteratively add candidate ensemble members to the `data_stack` using `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 using `predict()`

Our House

The `house` data contains information on 30 predictors for 200 houses in Ames, Iowa

- We perform data preprocessing using a `recipe`

```
set.seed(1221)
data_split <- initial_split(house , prop = 3/4)
train_data <- training(data_split)
test_data <- testing(data_split)

house_rec <-
  recipe(SalePrice ~ ., data = train_data) %>%
  update_role(Id, new_role = "ID") %>%
  step_log(LotArea, base = 10) %>%
  step_mutate(TotalBath = FullBath+0.5*HalfBath) %>%
  step_rm(FullBath, HalfBath) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric(), -all_outcomes())
```

Additional Definitions

There are a few more arguments we need to set before we begin:

Additional Definitions

There are a few more arguments we need to set before we begin:

- First, we create 10-fold cross-validation folds and specify our metric of interest

```
folds <- vfold_cv(train_data, v = 10)
metric <- metric_set(rmse)
```

Additional Definitions

There are a few more arguments we need to set before we begin:

- First, we create 10-fold cross-validation folds and specify our metric of interest

```
folds <- vfold_cv(train_data, v = 10)
metric <- metric_set(rmse)
```

- The `stacks` package requires that we save validation set predictions and workflows from `tune_grid` and `fit_resamples` after each model has been made.
- The package provides the helper functions to set appropriate values in function arguments.

Additional Definitions

There are a few more arguments we need to set before we begin:

- First, we create 10-fold cross-validation folds and specify our metric of interest

```
folds <- vfold_cv(train_data, v = 10)
metric <- metric_set(rmse)
```

- The stacks package requires that we save validation set predictions and workflows from tune_grid and fit_resamples after each model has been made.
- The package provides the helper functions to set appropriate values in function arguments.

```
ctrl_grid <- control_stack_grid()
ctrl_res <- control_stack_resamples()
```

Candidate Models:

- Let's build an ensemble from KNN, Linear Regression, LASSO, and a Random Forest.

Candidate Models:

- Let's build an ensemble from KNN, Linear Regression, LASSO, and a Random Forest.
- Note that KNN and LASSO require us to tune hyperparameters.

Candidate Models:

- Let's build an ensemble from KNN, Linear Regression, LASSO, and a Random Forest.
- Note that KNN and LASSO require us to tune hyperparameters.
- We'll also need to determine how to weight each individual model in our final ensemble:

KNN Model

- We begin with KNN

```
knn_mod <- nearest_neighbor(
    mode = "regression",
    neighbors = tune("k")) %>%
  set_engine("kknn")
```

KNN Model

- We begin with KNN

```
knn_mod <- nearest_neighbor(
    mode = "regression",
    neighbors = tune("k")) %>%
  set_engine("kknn")
```

- And then create a workflow:

```
knn_wf<- workflow() %>%
  add_model(knn_mod) %>%
  add_recipe(house_rec)
```

KNN Model

- We begin with KNN

```
knn_mod <- nearest_neighbor(
    mode = "regression",
    neighbors = tune("k")) %>%
  set_engine("kknn")
```

- And then create a workflow:

```
knn_wf<- workflow() %>%
  add_model(knn_mod) %>%
  add_recipe(house_rec)
```

- Now we tune and fit

```
knn_grid <- data.frame(k = 1+2*0:20)

knn_fit<- knn_wf %>% tune_grid(
  resamples = folds,
  metrics = metric,
  grid = knn_grid,
  control = ctrl_grid)
```

Linear Model

- On to the linear model:

```
lm_mod <- linear_reg() %>% set_engine("lm")
```

Linear Model

- On to the linear model:

```
lm_mod <- linear_reg() %>% set_engine("lm")
```

- Create the workflow

```
lm_wf<-workflow() %>%
  add_model(lm_mod) %>%
  add_recipe(house_rec)
```

Linear Model

- On to the linear model:

```
lm_mod <- linear_reg() %>% set_engine("lm")
```

- Create the workflow

```
lm_wf<-workflow() %>%
  add_model(lm_mod) %>%
  add_recipe(house_rec)
```

- And fit the model (no hyperparamters need to be tuned)

```
lm_fit <- lm_wf %>%
  fit_resamples(resamples = folds,
                metrics = metric,
                control = ctrl_res)
```

LASSO

Now, our LASSO mod:

```
lasso_mod <- linear_reg(
    mode = "regression",
    penalty = tune("lambda")) %>%
  set_engine("glmnet")
```

LASSO

Now, our LASSO mod:

```
lasso_mod <- linear_reg(
    mode = "regression",
    penalty = tune("lambda")) %>%
  set_engine("glmnet")
```

- And then create a workflow:

```
lasso_wf<- workflow() %>%
  add_model(lasso_mod) %>%
  add_recipe(house_rec)
```

LASSO

Now, our LASSO mod:

```
lasso_mod <- linear_reg(
    mode = "regression",
    penalty = tune("lambda")) %>%
  set_engine("glmnet")
```

- And then create a workflow:

```
lasso_wf<- workflow() %>%
  add_model(lasso_mod) %>%
  add_recipe(house_rec)
```

- Now we tune and fit

```
lasso_grid <- data.frame(lambda = 10^seq(-2, 8, length = 50))

lasso_fit<- lasso_wf %>% tune_grid(
  resamples = folds,
  metrics = metric,
  grid = lasso_grid,
  control = ctrl_grid)
```

Random Forest

- And finally our random forest

```
rf_mod <- rand_forest(mode = "regression") %>%
  set_engine("randomForest")
```

Random Forest

- And finally our random forest

```
rf_mod <- rand_forest(mode = "regression") %>%
  set_engine("randomForest")
```

- Create a workflow:

```
rf_wf <- workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(house_rec)
```

Random Forest

- And finally our random forest

```
rf_mod <- rand_forest(mode = "regression") %>%
  set_engine("randomForest")
```

- Create a workflow:

```
rf_wf <- workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(house_rec)
```

- And fit:

```
rf_fit <- rf_wf %>%
  fit_resamples(resamples = folds,
                metrics = metric,
                control = ctrl_res)
```

Model Comparisons

```r
show_best(knn_fit)
```

```
## # A tibble: 5 x 7
##       k .metric .estimator   mean     n std_err .config
##   <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <fct>
## 1    13 rmse    standard   35321.    10   2720. Preprocessor1_Model07
## 2    15 rmse    standard   35347.    10   2770. Preprocessor1_Model08
## 3    11 rmse    standard   35376.    10   2696. Preprocessor1_Model06
## 4    17 rmse    standard   35484.    10   2840. Preprocessor1_Model09
## 5     9 rmse    standard   35557.    10   2706. Preprocessor1_Model05
```

```r
show_best(lm_fit)
```

```
## # A tibble: 1 x 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <fct>
## 1 rmse    standard   29432.    10   2206. Preprocessor1_Model1
```

```r
show_best(lasso_fit)
```

```
## # A tibble: 5 x 7
##   lambda .metric .estimator   mean     n std_err .config
##    <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <fct>
## 1  2024. rmse    standard   27559.    10   2092. Preprocessor1_Model27
## 2  3237. rmse    standard   27633.    10   2100. Preprocessor1_Model28
## 3  1265. rmse    standard   27801.    10   2058. Preprocessor1_Model26
## 4   791. rmse    standard   28119.    10   2026. Preprocessor1_Model25
## 5  5179. rmse    standard   28163.    10   2206. Preprocessor1_Model29
```

```r
show_best(rf_fit)
```

```
## # A tibble: 1 x 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <fct>
## 1 rmse    standard   26223.    10   1702. Preprocessor1_Model1
```

Assemble the stack

- Initialize a data stack using `stacks()` and add models using `add_candidates()`

```
house_st <- stacks() %>%
  add_candidates(knn_fit) %>%
  add_candidates(lm_fit) %>%
  add_candidates(lasso_fit) %>%
  add_candidates(rf_fit)

house_st
```

```
## # A data stack with 4 model definitions and 42 candidate members:
## #   knn_fit: 21 model configurations
## #   lm_fit: 1 model configuration
## #   lasso_fit: 19 model configurations
## #   rf_fit: 1 model configuration
## # Outcome: SalePrice (integer)
```

View the results

```r
as_tibble(house_st)
```

```
## # A tibble: 150 x 43
##    SalePrice knn_fit_1_01 knn_fit_1_02 knn_fit_1_03 knn_fit_1_04 knn_fit_1_05
##        <int>        <dbl>        <dbl>        <dbl>        <dbl>        <dbl>
##  1    181500       157000      155945.      157129.      159518.      161213.
##  2    223500       214000      211433.      213342.      212579.      212093.
##  3    200000       213000      210744.      214239.      215301.      215143.
##  4    149000       197500      181938.      169051.      163401.      159450.
##  5    154000       156000      146808.      141622.      141656.      143595.
##  6    134800       123000      117001.      113778.      114715.      118051.
##  7    306000       245350      238819.      241925.      238714.      237535.
##  8    144000       119500      123946.      127121.      129489.      130933.
##  9    177000       423000      352397.      312471.      290560.      276843.
## 10    385000       440000      353588.      304609.      280451.      268259.
## # ... with 140 more rows, and 37 more variables: knn_fit_1_06 <dbl>,
## #   knn_fit_1_07 <dbl>, knn_fit_1_08 <dbl>, knn_fit_1_09 <dbl>,
## #   knn_fit_1_10 <dbl>, knn_fit_1_11 <dbl>, knn_fit_1_12 <dbl>,
## #   knn_fit_1_13 <dbl>, knn_fit_1_14 <dbl>, knn_fit_1_15 <dbl>,
## #   knn_fit_1_16 <dbl>, knn_fit_1_17 <dbl>, knn_fit_1_18 <dbl>,
## #   knn_fit_1_19 <dbl>, knn_fit_1_20 <dbl>, knn_fit_1_21 <dbl>,
## #   lm_fit_1_1 <dbl>, lasso_fit_1_01 <dbl>, lasso_fit_1_18 <dbl>, ...
```

Fit the stack

- We want our ensemble prediction to be a linear combination of the predictions from our candidate model.
  - How do find the coefficients for this lin. combo?

Fit the stack

- We want our ensemble prediction to be a linear combination of the predictions from our candidate model.
  - How do find the coefficients for this lin. combo?
  - LASSO! (implemented by the `blend_predictions()` function)

Fit the stack

- We want our ensemble prediction to be a linear combination of the predictions from our candidate model.
    - How do find the coefficients for this lin. combo?
    - LASSO! (implemented by the `blend_predictions()` function)

```
stacks_grid <-  10^seq(-3, 6, length = 20)
house_st_blend <- house_st %>%
  blend_predictions(penalty = stacks_grid, metric = metric)
```

Fit the stack

- We want our ensemble prediction to be a linear combination of the predictions from our candidate model.
    - How do find the coefficients for this lin. combo?
    - LASSO! (implemented by the `blend_predictions()` function)

```
stacks_grid <-  10^seq(-3, 6, length = 20)
house_st_blend <- house_st %>%
  blend_predictions(penalty = stacks_grid, metric = metric)
```

- Which models did we keep?

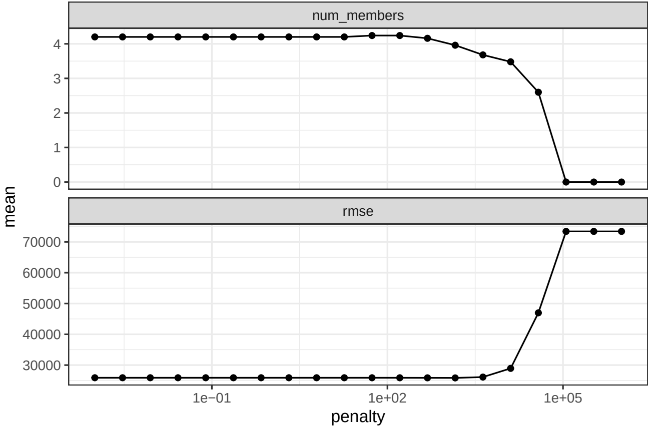
```
house_st_blend
```

```
## # A tibble: 3 x 3
##   member        type        weight
##   <chr>         <chr>        <dbl>
## 1 rf_fit_1_1    rand_forest 0.785
## 2 lm_fit_1_1    linear_reg  0.276
## 3 lasso_fit_1_27 linear_reg  0.0445
```

Plots

How do results vary depending on LASSO penalty?

```
theme_set(theme_bw())
autoplot(house_st_blend)
```



Fit Relevant Models

- Now we fit candidates with non-zero stacking coefficients on the training set:

Fit Relevant Models

- Now we fit candidates with non-zero stacking coefficients on the training set:

```
house_en_fit<- house_st_blend %>% fit_members()
```

Fit Relevant Models

- Now we fit candidates with non-zero stacking coefficients on the training set:

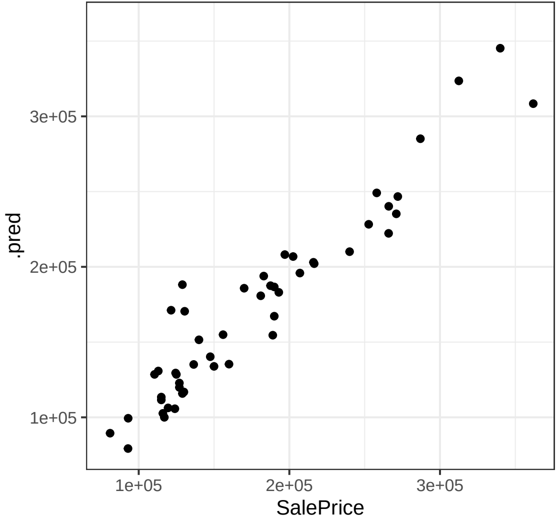
```
house_en_fit<- house_st_blend %>% fit_members()
```

- And predict with new data

```
house_preds<- test_data %>% bind_cols(predict(house_en_fit, .))
```

Results

- How did we do?



Comparison

- How does the ensemble compare to its constituents?

```
member_preds <- house_preds %>% select(SalePrice) %>%
  bind_cols(predict(house_en_fit, test_data, members = T))

map_dfr(member_preds, rmse, truth = SalePrice, data = member_preds) %>%
  mutate(member = colnames(member_preds))
```

```
## # A tibble: 5 x 4
##   .metric .estimator .estimate member
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard           0  SalePrice
## 2 rmse    standard      21403. .pred
## 3 rmse    standard      24410. lm_fit_1_1
## 4 rmse    standard      25451. lasso_fit_1_27
## 5 rmse    standard      26290. rf_fit_1_1
```