Stacks

Nate Wells

Math 243: Stat Learning

December 7th, 2020

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 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)

folds <- vfold_cv(train_data, v = 10)

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

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())
```

Candidate Models:

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

Candidate Models:

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

- Note that KNN and Random Forests require us to tune hyperparameters.

Candidate Models:

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

- Note that KNN and Random Forests 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_fit<- knn_wf %>% tune_grid(
  resamples = folds,
  grid = 4,
  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,
    control = ctrl_res
  )
```

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,
    control = ctrl_res
  )
```

Model Comparisons

```
collect_metrics(knn_fit)
```

```
## # A tibble: 8 x 7
##       k .metric .estimator      mean     n   std_err .config
##   <int> <chr>   <chr>          <dbl> <int>     <dbl> <fct>
## 1     2 rmse    standard   44103.       10 4132.     Preprocessor1_Model1
## 2     2 rsq     standard       0.683    10    0.0458 Preprocessor1_Model1
## 3     5 rmse    standard   36729.       10 3073.     Preprocessor1_Model2
## 4     5 rsq     standard       0.746    10    0.0441 Preprocessor1_Model2
## 5     8 rmse    standard   34887.       10 2693.     Preprocessor1_Model3
## 6     8 rsq     standard       0.772    10    0.0405 Preprocessor1_Model3
## 7    12 rmse    standard   34744.       10 2685.     Preprocessor1_Model4
## 8    12 rsq     standard       0.778    10    0.0378 Preprocessor1_Model4
```

```
collect_metrics(lm_fit)
```

```
## # A tibble: 2 x 6
##   .metric .estimator      mean     n   std_err .config
##   <chr>   <chr>          <dbl> <int>     <dbl> <fct>
## 1 rmse    standard   28946.       10 2139.     Preprocessor1_Model1
## 2 rsq     standard       0.846    10    0.0180 Preprocessor1_Model1
```

```
collect_metrics(rf_fit)
```

```
## # A tibble: 2 x 6
##   .metric .estimator      mean     n   std_err .config
##   <chr>   <chr>          <dbl> <int>     <dbl> <fct>
## 1 rmse    standard   26086.       10 1346.     Preprocessor1_Model1
## 2 rsq     standard       0.864    10    0.0268 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(rf_fit)

house_st
```

```
## # A data stack with 3 model definitions and 6 candidate members:
## #   knn_fit: 4 model configurations
## #   lm_fit: 1 model configuration
## #   rf_fit: 1 model configuration
## # Outcome: SalePrice (integer)
```

View the results

```
as_tibble(house_st)
```

```
## # A tibble: 150 x 7
##    SalePrice knn_fit_1_1 knn_fit_1_2 knn_fit_1_3 knn_fit_1_4 lm_fit_1_1
##        <int>       <dbl>       <dbl>       <dbl>       <dbl>      <dbl>
##  1    181500     160653.     160634.     161194.     162951.    184738.
##  2    223500     211218.     213342.     211898.     216205.    224672.
##  3    200000     203409.     211608.     209698.     208085.    261875.
##  4    149000     190914.     170669.     162758.     156798.    183815.
##  5    154000     132612.     137469.     138698.     142545.    161986.
##  6    134800     150763.     135536.     132409.     132958.    118624.
##  7    306000     202576.     227097.     229373.     227532.    256144.
##  8    144000     121566.     128493.     132799.     134461.    142724.
##  9    177000     386937.     312231.     282935.     262879.    258805.
## 10    385000     396401.     309550.     282696.     256982.    338928.
## # ... with 140 more rows, and 1 more variable: rf_fit_1_1 <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)

```
house_st_blend <- house_st %>% blend_predictions()
```

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)

```
house_st_blend <- house_st %>% blend_predictions()
```

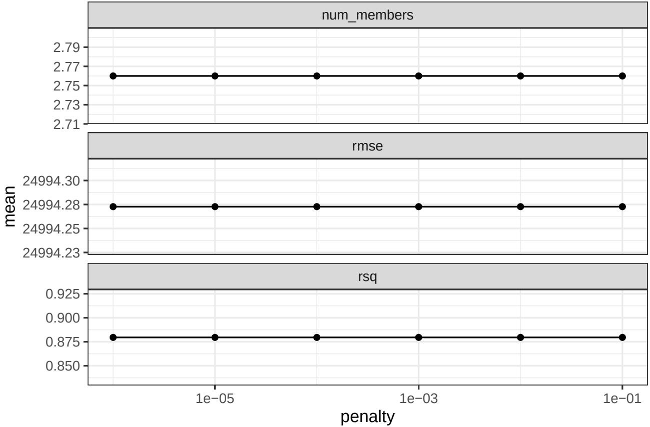
Which models did we keep?

```
house_st_blend
```

```
## # A tibble: 3 x 3
##   member      type               weight
##   <chr>       <chr>               <dbl>
## 1 rf_fit_1_1  rand_forest      0.813
## 2 lm_fit_1_1  linear_reg       0.303
## 3 knn_fit_1_3 nearest_neighbor 0.00415
```

Plots

How do results vary depending on LASSO penalty?



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:

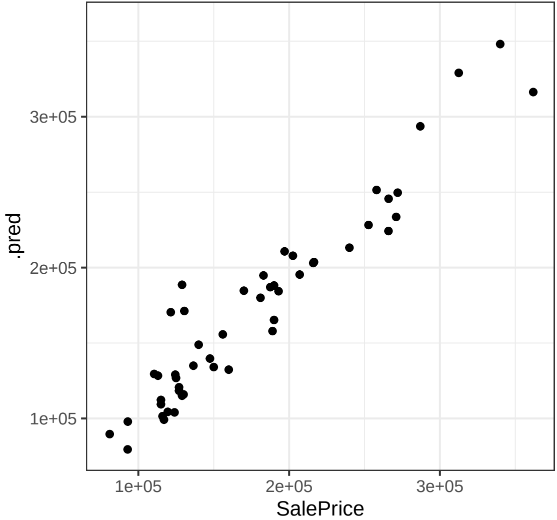
```
house_en_fit<- house_st_blend %>% fit_members()
```

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

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       20947. .pred
## 3 rmse    standard       37456. knn_fit_1_3
## 4 rmse    standard       24410. lm_fit_1_1
## 5 rmse    standard       25437. rf_fit_1_1
```