Tidymodels for Penalized Regression

Jin Seo Jo

We will work through how to fit ridge regression models using tidy models.

Ridge regression is an attempt to imporve the generlaizabiliy of linear regression when there are a lot of potentially correlated features. It tries to do this by imposing a penality on the size of the regression coefficients.

So if the model is

$$y_i = \beta_0 + x_i^T\beta + \epsilon_i$$

then instead of just computing the least squares estimate of β, ridge regression instead minimizes the modified or penalized loss function

$$\sum_{i=1}^{n}(y_i - \beta_0 - x_i^T\beta) + \lambda\|\beta\|_2^2$$

where λ is an unknown parameter that trades of fit (we get standard regression as $\lambda \to 0$) and generalizability.

The example is a TidyTuesday data set about The Office. We will predict the IMDb rating from characteristics about the episodes.

Step1: Download and clean the ratings data

Removing common but unimportant things in the episode names.

```r
library(tidyverse)

ratings_raw <- readr::read_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/da

remove_regex <- "[:punct:]|[:digit:]|parts |part |the |and"

office_ratings <- ratings_raw %>%
  # transmute() adds new variables and drops existing ones.
  transmute(
    episode_name =str_to_lower(title),
    # str_remove_all() removes matched patterns in a string.
    episode_name =str_remove_all(episode_name, remove_regex),
    # str_trim() removes whitespace from start and end of string.
    episode_name =str_trim(episode_name),
    imdb_rating
    )
```

The `office_ratings` is *wild*. It is the entire script of The Office! It lives in the `schrute` package. (Notice the overlap, but it's good practice to pretend that `imdb_rating` column isn't there.)

```r
# install.packages("schrute")
library(schrute)

schrute::theoffice
```

```
## # A tibble: 55,130 x 12
##    index season episode episode_name director  writer     character text
##    <int>  <int>   <int> <chr>        <chr>     <chr>      <chr>     <chr>
##  1     1      1       1 Pilot        Ken Kwap~ Ricky Ger~ Michael   All right J~
##  2     2      1       1 Pilot        Ken Kwap~ Ricky Ger~ Jim       Oh, I told ~
##  3     3      1       1 Pilot        Ken Kwap~ Ricky Ger~ Michael   So you've c~
##  4     4      1       1 Pilot        Ken Kwap~ Ricky Ger~ Jim       Actually, y~
##  5     5      1       1 Pilot        Ken Kwap~ Ricky Ger~ Michael   All right. ~
##  6     6      1       1 Pilot        Ken Kwap~ Ricky Ger~ Michael   Yes, I'd li~
##  7     7      1       1 Pilot        Ken Kwap~ Ricky Ger~ Michael   I've, uh, I~
##  8     8      1       1 Pilot        Ken Kwap~ Ricky Ger~ Pam       Well. I don~
##  9     9      1       1 Pilot        Ken Kwap~ Ricky Ger~ Michael   If you thin~
## 10    10      1       1 Pilot        Ken Kwap~ Ricky Ger~ Pam       What?
## # ... with 55,120 more rows, and 4 more variables: text_w_direction <chr>,
## #   imdb_rating <dbl>, total_votes <int>, air_date <fct>
```

Let's take that and get rid of all of the boring talking, but keep information that might be useful for predicting IMDB ratings.

```r
office_info <- schrute::theoffice %>%
  mutate(season = as.numeric(season),
         episode = as.numeric(episode),
         episode_name = str_to_lower(episode_name),
         episode_name = str_remove_all(episode_name, remove_regex),
         episode_name = str_trim(episode_name)) %>%
  select(season, episode, episode_name, director, writer, character)

office_info
```

```
## # A tibble: 55,130 x 6
##    season episode episode_name director   writer                          character
##     <dbl>   <dbl> <chr>        <chr>      <chr>                           <chr>
##  1      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Michael
##  2      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Jim
##  3      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Michael
##  4      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Jim
##  5      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Michael
##  6      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Michael
##  7      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Michael
##  8      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Pam
##  9      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Michael
## 10      1       1 pilot        Ken Kwapis Ricky Gervais;Stephen Merch~ Pam
## # ... with 55,120 more rows
```

We are going to collect a few other things:
- How often characters speak per episode
- Who was involved in writing and directing

Characters speak per episode:

```
characters <- office_info %>%
  count(episode_name, character) %>%
  add_count(character, wt = n, name = "character_count") %>%
  filter(character_count > 800) %>%
  select(-character_count) %>%
  pivot_wider(
    names_from = character,
    values_from = n,
    values_fill = list(n = 0)
  )

characters
```

```
## # A tibble: 185 x 16
##    episode_name  Andy Angela Darryl Dwight   Jim Kelly Kevin Michael Oscar   Pam
##    <chr>        <int>  <int>  <int>  <int> <int> <int> <int>   <int> <int> <int>
##  1 a benihana ~    28     37      3     61    44     5    14     108     1    57
##  2 aarm            44     39     30     87    89     0    30       0    28    34
##  3 after hours     20     11     14     60    55     8     4       0    10    15
##  4 alliance         0      7      0     47    49     0     3      68    14    22
##  5 angry y         53      7      5     16    19    13     9       0     7    29
##  6 baby shower     13     13      9     35    27     2     4      79     3    25
##  7 back from v~     3      4      6     22    25     0     5      70     0    33
##  8 banker           1      2      0     17     0     0     2      44     0     5
##  9 basketball       0      3     15     25    21     0     1     104     2    14
## 10 beach games     18      8      0     38    22     9     5     105     5    23
## # ... with 175 more rows, and 5 more variables: Phyllis <int>, Ryan <int>,
## #   Toby <int>, Erin <int>, Jan <int>
```

The writers and directors:

```
creators <- office_info %>%
  distinct(episode_name, director, writer) %>% # Beware of utiples and drop cols
  pivot_longer(cols = director:writer,
               names_to = "role",
               values_to = "person") %>%
  separate_rows(person, sep = ";") %>% # One row per person
  add_count(person) %>% # Do the counting
  filter(n > 10) %>% # Throw out the rare people
  distinct(episode_name, person) %>% # Make sure it's distinct
  mutate(person_value = 1) %>%
  pivot_wider(
    names_from = person,
    values_from = person_value,
    values_fill = list(person_value = 0)
  )

creators
```

```
## # A tibble: 135 x 14
##    episode_name      `Ken Kwapis` `Greg Daniels` `B.J. Novak` `Paul Lieberstein`
##    <chr>                    <dbl>          <dbl>        <dbl>              <dbl>
##  1 pilot                        1              1            0                  0
##   2 diversity day                  1              0             1                  0
##   3 health care                    0              0             0                  1
##   4 basketball                     0              1             0                  0
##   5 hot girl                       0              0             0                  0
##   6 dundies                        0              1             0                  0
##   7 sexual harassment              1              0             1                  0
##   8 office olympics                0              0             0                  0
##   9 fire                           1              0             1                  0
##  10 halloween                      0              1             0                  0
## # ... with 125 more rows, and 9 more variables: Mindy Kaling <dbl>,
## #   Paul Feig <dbl>, Gene Stupnitsky <dbl>, Lee Eisenberg <dbl>,
## #   Jennifer Celotta <dbl>, Randall Einhorn <dbl>, Brent Forrester <dbl>,
## #   Jeffrey Blitz <dbl>, Justin Spitzer <dbl>
```

Now that we've got all of those things sorted out, we need to join them together.

All of the tibbles `office_info`, `characters`, and `creators` have one common column: `episode_name`. We can use this to join them together to get a final data set.

The magic function here is `ineer_join`. This is a great example of a Two Table Verb, so called becuase it acts on two tables. The purpose of `inner_join` is to join these two tables in such a way that the output has all of the rows that are in **both** tables.

This is an example:

```
df1 <- tibble(x = c(1, 2), y = 2:1)
df2 <- tibble(x = c(3, 1), a = 10, b = "a")
inner_join(df1, df2, by = "x")
```

```
## # A tibble: 1 x 4
##       x     y     a b
##   <dbl> <int> <dbl> <chr>
## 1     1     2    10 a
```

See that this has kept *only* the row that had a common value of x (the **by** argument).
(When we skip the **by** argument, it will just try to join on everything that is common.)

Other types of joins:
- `left_join()` includes all rows in the first table
- `right_join()` includes all rows in the second table
- `full_join()` includes all rows in either table

Let's do it with the data. We are also hitting it with a `janitor::clean_names()`, which is a wonderful way to clean up the column headings and remove any odd puntuation or capitalization or any other type of non-standard thing.

```
office <- office_info %>%
  distinct(season, episode, episode_name) %>%
  inner_join(characters) %>%
  inner_join(creators) %>%
  inner_join(office_ratings %>%
               select(episode_name, imdb_rating)) %>%
  janitor::clean_names()
```

```
## Joining, by = "episode_name"
## Joining, by = "episode_name"
## Joining, by = "episode_name"
```

```
office
```

```
## # A tibble: 136 x 32
##    season episode episode_name       andy angela darryl dwight   jim kelly kevin
##     <dbl>   <dbl> <chr>             <int>  <int>  <int>  <int> <int> <int> <int>
##  1      1       1 pilot                 0      1      0     29    36     0     1
##  2      1       2 diversity day         0      4      0     17    25     2     8
##  3      1       3 health care           0      5      0     62    42     0     6
##  4      1       5 basketball            0      3     15     25    21     0     1
##  5      1       6 hot girl              0      3      0     28    55     0     5
##  6      2       1 dundies               0      1      1     32    32     7     1
##  7      2       2 sexual harassment     0      2      9     11    16     0     6
##  8      2       3 office olympics       0      6      0     55    55     0     9
##  9      2       4 fire                  0     17      0     65    51     4     5
## 10      2       5 halloween             0     13      0     33    30     3     2
## # ... with 126 more rows, and 22 more variables: michael <int>, oscar <int>,
## #   pam <int>, phyllis <int>, ryan <int>, toby <int>, erin <int>, jan <int>,
## #   ken_kwapis <dbl>, greg_daniels <dbl>, b_j_novak <dbl>,
## #   paul_lieberstein <dbl>, mindy_kaling <dbl>, paul_feig <dbl>,
## #   gene_stupnitsky <dbl>, lee_eisenberg <dbl>, jennifer_celotta <dbl>,
## #   randall_einhorn <dbl>, brent_forrester <dbl>, jeffrey_blitz <dbl>,
## #   justin_spitzer <dbl>, imdb_rating <dbl>
```

Now our data is clean, we can do some modelling.

Organize the data

The first step in any pipeline is always building a recipe to make our data analysis standard and to allow us to document the transformations we made.
First, we need to split our data. Because we are going to be doing complicated things later, we need to split our data into a test and training set.

The `rsample` package, which is part of `tidymodels` makes this a breeze.

```
library(tidymodels)
```

```
office_split <- initial_split(office, strata = season)
office_train <- training(office_split)
office_test <- testing(office_split)
```

- The `initial_split()` function has a **prop** argument that controls how much data is in the training set. The default value **prop = 0.75** is fine for our purposes.
- The `strata` argument makes sure this 75/25 split is carried out for each season. Typically, the variable you want to stratify by is any variable that will have uneven sampling.
- The `training()` and `testing()` functions extract the test and training data sets.

Build a recipe

For predictive modelling, we need to specify which column is being predicted. The `recipe()` function takes a formula argument, which controls this. For the moment, we are just going to regress `imdb_rating` against everything else, so our formula is `imdb_rating ~ .`. We also need to do this to the training data.

```r
office_rec <- recipe(imdb_rating ~ ., data = office_train) %>%
  update_role(episode_name, new_role = "ID") %>%
  step_zv(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes())
```

So there are three more steps.
- The first one labels the column `episode_name` as an "ID" column, which is basically just a column that we keep around for fun and plotting, but we don't want to be in our model.
- `step_zv` removes any column that has zero variance (i.e. is all the same value). We are applying this to all numeric columns, but not to the outcomes. (In this case, the outcome is `imdb_rating`.)
- We are normalizing all of the numeric columns that are the outcome.

Now we can actually prepare that recipe with the `prep` function. This does things like calculate the centering and the scaling. It does not apply them yet!

The `string_as_factors = FALSE` argument is just to make sure that `episode_name` isn't converted.

```r
office_prep <- office_rec %>%
  prep(strings_as_factors = FALSE)
```

Fit the ridge regression

We will fit this ridge regression with a package called `glmnet`. But we are going to use tidymodel bindings so we don't have to work too hard to understand how to use it. One of the best things that tidymodels does is provide a clean and common interfact to these functions.

We specify a model with two things: A *specification* and an *engine*.

```r
ridge_reg_spec <- linear_reg(penalty = 0.3, mixture = 0) %>%
  set_engine("glmnet")

ridge_reg_spec
```

```
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = 0.3
##   mixture = 0
##
## Computational engine: glmnet
```

It tells us that
- we are doing a linear regression-type problem (that tells us what loss function we are using).
- we have set the penalty parameter $\lambda = 0.3$. This is arbitrary. We will learn how to find it from data later.
- `mixture = 0` says we should do ridge regression. `mixture = 1` uses the LASSO.

And the `set_engine()` function tells the specification exactly what package to use. (There are multiple options.)

Workflows

When we are doing real data analysis, there are often multiple models and multiple data sets lying around. `tidymodels` has a nice concept to ensure that a particular recipe and a particular model specification can *stay* linked. This is a workflow.

We don't need anything advanced here, so for this one moment we are going to just add the data recipe to the workflow.

```
wf_rr <- workflow() %>%
  add_recipe(office_rec)

wf_rr
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: None
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_zv()
## * step_normalize()
```

Fit the model

To fit the model, we add its specification to the workflow and then call the `fit` fcuntion.

```
simple_rr_fit <- wf_rr %>%
  add_model(ridge_reg_spec) %>%
  fit(data = office_train)
```

We can then view the fit by "pulling" it out of the workflow.

```
simple_rr_fit %>%
  pull_workflow_fit() %>%
  tidy()
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-1
```

```
## # A tibble: 31 x 3
##    term        estimate penalty
##    <chr>          <dbl>   <dbl>
##  1 (Intercept)   8.36       0.3
##  2 season       -0.0300     0.3
##  3 episode       0.0482     0.3
##  4 andy          0.0363     0.3
##  5 angela        0.0224     0.3
##  6 darryl        0.0422     0.3
##  7 dwight        0.0418     0.3
##  8 jim           0.0487     0.3
##  9 kelly        -0.0335     0.3
## 10 kevin         0.0313     0.3
## # ... with 21 more rows
```

We probably shouldn't just choose an arbitrary value for the penalty.

We really need to work out some sort of value for this tuning parameter. Unsurprisingly, there is a way to do that in tidymodels. Instead of giving `penalty` a specific value we can set it to the function `tune()`.

The new spec looks like this.

```
tune_rr_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
  set_engine("glmnet")
```

We need to have some idea of what values to try. Because the bad news is that we are basically going to just fit this model a lot for a range of different penalty values. There's a helper function for this called `grid_regular()` and another function called `penalty()` that knows things about what a penalty has to look like.

But if you want to, you can just make a one column tibble with a column called `penalty` with whatever values you want. I'm going to choose 20 values that are evenly spaced in log space.

```
lambda_grid <- grid_regular(penalty(), levels = 20)

lambda_grid
```

```
## # A tibble: 20 x 1
##     penalty
##       <dbl>
##  1 1   e-10
##  2 3.36e-10
##  3 1.13e- 9
##  4 3.79e- 9
##  5 1.27e- 8
##  6 4.28e- 8
##  7 1.44e- 7
##  8 4.83e- 7
##  9 1.62e- 6
## 10 5.46e- 6
## 11 1.83e- 5
## 12 6.16e- 5
## 13 2.07e- 4
## 14 6.95e- 4
## 15 2.34e- 3
## 16 7.85e- 3
## 17 2.64e- 2
## 18 8.86e- 2
## 19 2.98e- 1
## 20 1   e+ 0
```

Finding λ

We will try to find a value of λ that has the smallest estimated test error. We can get this through Cross Validation. We need to do this because we do not want to touch our test set.

We are going to use k-fold, where the data is randomly split into k groups of roughly equal sizes (these are called the folds). We can use the `vfold_cv()` function to do this. We are going to do these splits in a stratified manner again, because if you need to stratify by a variable at one point in the analysis you need to do at every point for the same reason.

This will make 1- lists with various things in them, including the subset of the data.

```
folds <- vfold_cv(office_train, v = 10, strata = season)
folds
```

```
## #  10-fold cross-validation using stratification
## # A tibble: 10 x 2
##    splits          id
##    <list>          <chr>
##  1 <split [91/12]> Fold01
##  2 <split [91/12]> Fold02
##  3 <split [91/12]> Fold03
##  4 <split [91/12]> Fold04
##  5 <split [91/12]> Fold05
##  6 <split [93/10]> Fold06
##  7 <split [94/9]>  Fold07
##  8 <split [95/8]>  Fold08
##  9 <split [95/8]>  Fold09
## 10 <split [95/8]>  Fold10
```

```
folds$splits[[1]]$data
```

```
## # A tibble: 103 x 32
##    season episode episode_name       andy angela darryl dwight   jim kelly kevin
##     <dbl>   <dbl> <chr>             <int>  <int>  <int>  <int> <int> <int> <int>
##  1      1       1 pilot                 0      1      0     29    36     0     1
##  2      1       2 diversity day         0      4      0     17    25     2     8
##  3      1       3 health care           0      5      0     62    42     0     6
##  4      1       5 basketball            0      3     15     25    21     0     1
##  5      1       6 hot girl              0      3      0     28    55     0     5
##  6      2       1 dundies               0      1      1     32    32     7     1
##  7      2       3 office olympics       0      6      0     55    55     0     9
##  8      2       4 fire                  0     17      0     65    51     4     5
##  9      2       5 halloween             0     13      0     33    30     3     2
## 10      2       8 performance revi~     0      5      0     42    26     2     4
## # ... with 93 more rows, and 22 more variables: michael <int>, oscar <int>,
## #   pam <int>, phyllis <int>, ryan <int>, toby <int>, erin <int>, jan <int>,
## #   ken_kwapis <dbl>, greg_daniels <dbl>, b_j_novak <dbl>,
## #   paul_lieberstein <dbl>, mindy_kaling <dbl>, paul_feig <dbl>,
## #   gene_stupnitsky <dbl>, lee_eisenberg <dbl>, jennifer_celotta <dbl>,
## #   randall_einhorn <dbl>, brent_forrester <dbl>, jeffrey_blitz <dbl>,
## #   justin_spitzer <dbl>, imdb_rating <dbl>
```

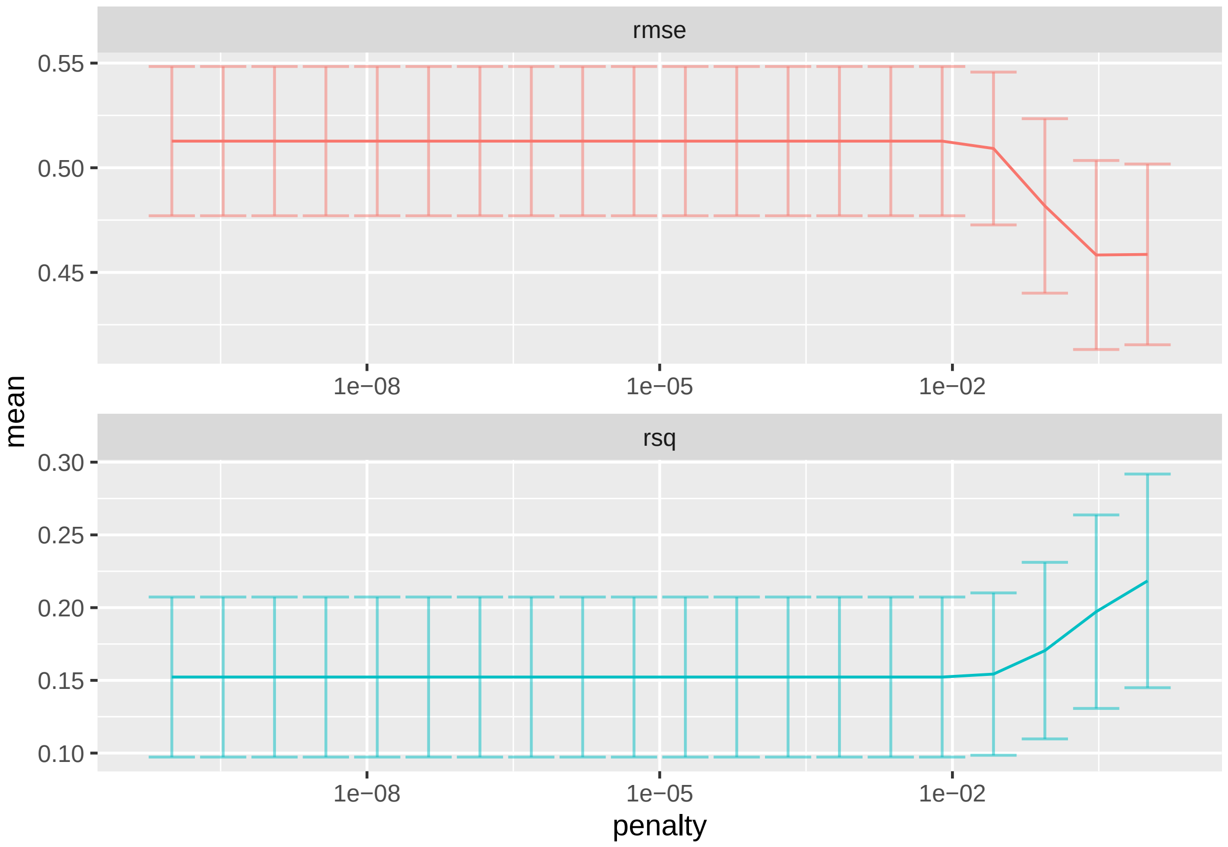
Now we can actually tune our model to find a good value of λ

```
wf_rr <- wf_rr %>% add_model(tune_rr_spec)

rr_grid <- tune_grid(
  wf_rr,
  resamples = folds,
  grid = lambda_grid
)
```

We collected a bunch of metrics that we can now look at.

```
rr_grid %>% collect_metrics() %>%
  ggplot(aes(x = penalty, y = mean, colour = .metric)) +
  geom_errorbar(aes(ymin = mean - std_err, ymax = mean + std_err), alpha = 0.5) +
  geom_line() +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10() +
  theme(legend.position = "none")
```



Note that "rmse" represents the root mean squared error, and "rsq" represents R-squared.

We see that a relatively large penalty is good. We can select the best one!

```
lowest_rmse <- rr_grid %>% select_best("rmse") # We want it small

lowest_rmse
```

```
## # A tibble: 1 x 2
##   penalty .config
##     <dbl> <chr>
## 1   0.298 Preprocessor1_Model19
```

```
final_rr <- finalize_workflow(wf_rr, lowest_rmse)

final_rr
```

```
## == Workflow ===================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ---------------------------------------------------------------
## 2 Recipe Steps
##
## * step_zv()
## * step_normalize()
##
## -- Model ----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = 0.297635144163131
##   mixture = 0
##
## Computational engine: glmnet
```

Now with our workflow finalized, we can finally fit this model on the whole training set, and then evaluate it on the test set.

```
last_fit(final_rr, office_split) %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       0.434 Preprocessor1_Model1
## 2 rsq     standard       0.302 Preprocessor1_Model1
```