Modeling with the `tidymodels` Framework

Justin Post

Modeling Process

Given a model, we **fit** the model using data

- Must determine how well the model predicts on **new** data
- Create a test set or use CV (or perhaps both…)
- Judge effectiveness using a **metric** on predictions made from the model

Preparing the Data

General flow for modeling

- Read data in
- EDA (or perhaps after train/test split…)
- Split data into train and test (do response transform first!)

- Modify training data set predictors as needed

    - Center/scale
    - Create factors & dummy variables
    - Create interactions/quadratics/etc.
    - Log transform
    - …

- Fit model(s) on training data

- Use same transformations on the test data or in CV process (*exactly* as done in training set)
- Predict on the test set

Convert Data

- We saw the use of `rsample::initial_split()`
  - **If doing a *non-learned* transformation, do those first outside of** `tidymodels`

```
library(tidyverse)
library(tidymodels)
bike_data <- read_csv("https://www4.stat.ncsu.edu/~online/datasets/bikeDetails.csv") |>
  mutate(log_selling_price = log(selling_price)) |>
  select(-selling_price)
#save creation of new variables for now!
bike_split <- initial_split(bike_data, prop = 0.7)
bike_train <- training(bike_split)
bike_test <- testing(bike_split)
```

- `initial_split()` allows for stratified sampling too!

Data Prepration with `tidymodels`

- `recipes` package within `tidymodels` allows for transformations
    - Process keeps track of proper values to use for you!
    - Start with `` ` ``recipe()` call
        - Denote formula for response/predictors and datato use
        - `summary()` describes current setup (we don't want all of these as predictors)

```
recipe(log_selling_price ~ ., data = bike_train) |>
  summary()
```

```
## # A tibble: 7 x 4
##   variable          type      role      source
##   <chr>             <list>    <chr>     <chr>
## 1 name              <chr [3]> predictor original
## 2 year              <chr [2]> predictor original
## 3 seller_type       <chr [3]> predictor original
## 4 owner             <chr [3]> predictor original
## 5 km_driven         <chr [2]> predictor original
## 6 ex_showroom_price <chr [2]> predictor original
## 7 log_selling_price <chr [2]> outcome   original
```

Data Prepration with `tidymodels`

- `recipes` package within `tidymodels` allows for transformations
    - `update_role()` allows you to declare types of variables (such as `ID`)
    - This keeps the variable around even when not used in a model

```
recipe(log_selling_price ~ ., data = bike_train) |>
  update_role(name, new_role = "ID") |>
  summary()
```

```
## # A tibble: 7 x 4
##   variable          type      role      source
##   <chr>             <list>    <chr>     <chr>
## 1 name              <chr [3]> ID        original
## 2 year              <chr [2]> predictor original
## 3 seller_type       <chr [3]> predictor original
## 4 owner             <chr [3]> predictor original
## 5 km_driven         <chr [2]> predictor original
## 6 ex_showroom_price <chr [2]> predictor original
## 7 log_selling_price <chr [2]> outcome   original
```

Now Add Transformation Steps

- Many `step_*` functions to consider
  - `step_log()` to create our `log_km_driven` variable
  - `step_rm()` to remove a variable
  - `step_dummy()` to create dummy values for categorical variables
  - `step_normalize()` to center and scale numeric predictors

```
recipe(log_selling_price ~ ., data = bike_train) |>
  update_role(name, new_role = "ID") |>
  step_log(km_driven) |>
  step_rm(ex_showroom_price) |>#too many nas
  step_dummy(owner, seller_type) |>
  step_normalize(all_numeric(), -all_outcomes())
```

prep() & bake() the Recipe

- If you have at least one preprocessing operation, `prep()` 'estimates the required parameters from a training set that can be later applied to other data sets'
- `bake()` applies the computations to data

```
recipe(log_selling_price ~ ., data = bike_train) |>
  update_role(name, new_role = "ID") |>
  step_log(km_driven) |>
  step_rm(ex_showroom_price) |>
  step_dummy(owner, seller_type) |>
  step_normalize(all_numeric(), -all_outcomes()) |>
  prep(training = bike_train) |>
  bake(bike_train)
```

```
## # A tibble: 742 x 8
##   name         year km_driven log_selling_price owner_X2nd.owner owner_X3rd.owner
##   <fct>       <dbl>     <dbl>             <dbl>            <dbl>            <dbl>
## 1 Bajaj Di~ -0.405    0.808              10.3             -0.367           -0.111
## 2 Honda Ac~  0.274   -1.05               10.6             -0.367           -0.111
## 3 Bajaj Pu~ -1.99    -0.0115              9.80            -0.367           -0.111
## 4 Hero HF ~  0.726    0.197              10.5             -0.367           -0.111
## 5 Royal En~ -0.179    0.788              11.4             -0.367           -0.111
## # i 737 more rows
## # i 2 more variables: owner_X4th.owner <dbl>, seller_type_Individual <dbl>
```

`parsnip` for Creating a Model

- `prep()` and `bake()` steps are not required but help us debug/see what things look like
- Once we have our `recipe()` ready, we also need do our modeling setup
  - Use `parsnip` package to specify a model
  - `parsnip` abstracts away the individual package syntax
  - Specify the model type and model engine
  - This page allows us to search for a model type so we can see which `model` and `engine` we want to specify!

Creating a Model with `tidymodels`

- Fit MLR model with `linear_reg()`
- Engine set to `lm` for basic models
- Info page

```
linear_reg() %>%
  set_engine("lm") %>%
  translate()
```

```
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
##
## Model fit template:
## stats::lm(formula = missing_arg(), data = missing_arg(), weights = missing_arg())
```

Creating a Model with `tidymodels`

- Set up our model and recipes

```
bike_rec <- recipe(log_selling_price ~ ., data = bike_train) |>
  update_role(name, new_role = "ID") |>
  step_log(km_driven) |>
  step_rm(ex_showroom_price) |>
  step_dummy(owner, seller_type) |>
  step_normalize(all_numeric(), -all_outcomes())

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

`workflow()`s with `tidymodels`

- Now we can create a `workflow()`
  - Add our recipe and model with their corresponding functions

```
bike_wfl <- workflow() |>
  add_recipe(bike_rec) |>
  add_model(bike_mod)
bike_wfl
```

```
## == Workflow ==================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor --------------------------------------------------------------
## 4 Recipe Steps
##
## * step_log()
## * step_rm()
## * step_dummy()
## * step_normalize()
##
## -- Model ---------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

`fit()` That Model!

- Finally, `fit()` allows us to fit our model to a data set!
- `tidy()` puts the results into a `tibble`

```
bike_fit <- bike_wfl |>
  fit(bike_train)
bike_fit |>
  tidy()
```

```
## # A tibble: 7 x 5
##   term                   estimate std.error statistic  p.value
##   <chr>                     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)            10.7        0.0179   598.    0
## 2 year                    0.336      0.0210    16.0   1.00e-49
## 3 km_driven              -0.241      0.0204   -11.8   1.80e-29
## 4 owner_X2nd.owner        0.00538    0.0183     0.293 7.69e- 1
## 5 owner_X3rd.owner        0.0740     0.0183     4.03  6.08e- 5
## 6 owner_X4th.owner        0.0333     0.0180     1.85  6.52e- 2
## 7 seller_type_Individual  0.00835    0.0180     0.464 6.43e- 1
```

Find Test Set Metric(s)

- Here we don't have a bunch of models we are comparing, only one is fit
- Can use `last_fit()` on the original `initial_split()` object (`bike_split`) to see how it performs on the test set
- `collect_metrics()` returns the metrics on the test set!

```
bike_wfl |>
  last_fit(bike_split) |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       0.556 Preprocessor1_Model1
## 2 rsq     standard       0.466 Preprocessor1_Model1
```

Find Test Set Metric(s)

- Here we don't have a bunch of models we are comparing, only one is fit
- Can use `last_fit()` on the original `initial_split()` object (`bike_split`) to see how it performs on the test set
- `collect_metrics()` returns the metrics on the test set!

```
bike_wfl |>
  last_fit(bike_split) |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       0.556 Preprocessor1_Model1
## 2 rsq     standard       0.466 Preprocessor1_Model1
```

The same transformations from the training set are used on the test set!

Fitting the Model with Cross-Validation

- Let's use 10 fold CV in the training set instead
  - Compare to another model's CV fit on the training set
  - Send best model to test set

Fitting the Model with Cross-Validation

- Let's use 10 fold CV in the training set instead
    - Compare to another model's CV fit on the training set
    - Send best model to test set
- Use `vfold_cv()` to split the data up and use `fit_resamples()` to fit the model appropriately

```
bike_10_fold <- vfold_cv(bike_train, 10)
bike_CV_fits <- bike_wfl |>
  fit_resamples(bike_10_fold)
bike_CV_fits
```

```
## # Resampling results
## # 10-fold cross-validation
## # A tibble: 10 x 4
##    splits            id     .metrics         .notes
##    <list>            <chr>  <list>           <list>
##  1 <split [667/75]> Fold01 <tibble [2 x 4]> <tibble [0 x 3]>
##  2 <split [667/75]> Fold02 <tibble [2 x 4]> <tibble [0 x 3]>
##  3 <split [668/74]> Fold03 <tibble [2 x 4]> <tibble [0 x 3]>
##  4 <split [668/74]> Fold04 <tibble [2 x 4]> <tibble [0 x 3]>
##  5 <split [668/74]> Fold05 <tibble [2 x 4]> <tibble [0 x 3]>
##  6 <split [668/74]> Fold06 <tibble [2 x 4]> <tibble [0 x 3]>
##  7 <split [668/74]> Fold07 <tibble [2 x 4]> <tibble [0 x 3]>
```

Fitting the Model with Cross-Validation

- Combine the metrics using `collect_metrics()`

```
bike_CV_fits |>
  collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.495    10  0.0257 Preprocessor1_Model1
## 2 rsq     standard   0.498    10  0.0462 Preprocessor1_Model1
```

- This is our CV error on the training set!

Fit another Model with Cross-Validation for Comparison

- Let's build another recipe that includes interaction terms

```
bike_int_rec <- recipe(log_selling_price ~ ., data = bike_train) |>
  update_role(name, new_role = "ID") |>
  step_log(km_driven) |>
  step_rm(ex_showroom_price) |>
  step_dummy(owner, seller_type) |>
  step_normalize(all_numeric(), -all_outcomes()) |>
  step_interact(terms = ~km_driven*year*starts_with("seller_type"))
```

Fit another Model with Cross-Validation for Comparison

- Fit the model to the resamples and see our metric

```
bike_int_CV_fits <- workflow() |>
  add_recipe(bike_int_rec) |>
  add_model(bike_mod) |>
  fit_resamples(bike_10_fold)
rbind(bike_CV_fits |> collect_metrics(),
      bike_int_CV_fits |> collect_metrics())
```

```
## # A tibble: 4 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.495    10  0.0257 Preprocessor1_Model1
## 2 rsq     standard   0.498    10  0.0462 Preprocessor1_Model1
## 3 rmse    standard   0.540    10  0.0390 Preprocessor1_Model1
## 4 rsq     standard   0.460    10  0.0505 Preprocessor1_Model1
```

- Simpler model is better here
- Could now compare its perofrmance on the test set to some other 'best' models

Recap

- `tidymodels` provides a framework for predictive modeling
- Define a recipe
- Define a model and engine
- Fit the models (perhaps using cross-validation) and investigate metrics