

A Typical (supervised) ML Workflow

Itamar Caspi

10, 2019 (updated: 2019-12-01)

Packages and setup

Use the **pacman** package that automatically loads and installs packages:

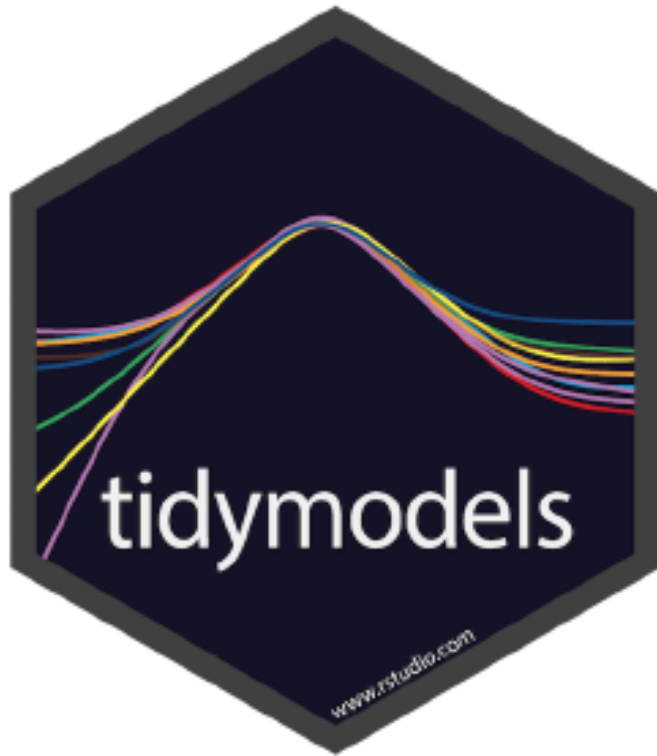
```
if (!require("pacman")) install.packages("pacman")

pacman::p_load(
  tidyverse,    # for data wrangling and visualization
  tidymodels,   # for data modeling
  tune,         # for tuning hyperparameters
  GGally,       # for pairs plot
  naniar,       # for summary statistics
  here          # for referencing folders and files
)
```

Set a theme for **ggplot** (Relevant only for the presentation)

```
theme_set(theme_grey(20))
```

The tidymodels package



"**tidymodels** is a "meta-package" for modeling and statistical analysis that share the underlying design philosophy, grammar, and data structures of the tidyverse."

Supervised ML Workflow

Step 1: Define the Prediction Task

Step 2: Explore the Data

Step 3: Set Model and Tuning Parameters

Step 4: Cross-validation

Step 5: Evaluate the Model

Step 1: Define the Prediction Task

Predicting Boston Housing Prices

We will use the `BostonHousing`: housing data for 506 census tracts of Boston from the 1970 census (Harrison and Rubinfeld, 1978).

- `medv` (target): median value of owner-occupied homes in USD 1000's.
- `lstat`(predictor): percentage of lower status of the population.
- `chas` (predictor): Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).

OBJECTIVE: Predict `medv`.



Source: <https://www.bostonusa.com/>

Load the Data

Load the data

```
boston_raw <- read_csv(here("04-ml-workflow/data", "BostonHousing.csv"))
```

```
## Parsed with column specification:
```

```
## cols(  
##   crim = col_double(),  
##   zn = col_double(),  
##   indus = col_double(),  
##   chas = col_double(),  
##   nox = col_double(),  
##   rm = col_double(),  
##   age = col_double(),  
##   dis = col_double(),  
##   rad = col_double(),  
##   tax = col_double(),  
##   ptratio = col_double(),  
##   b = col_double(),  
##   lstat = col_double(),  
##   medv = col_double()  
## )
```

What Type of Data?

We can use the `glimpse()` function in order to better understand the data structure:

```
glimpse(boston_raw)
```

```
## Observations: 506
## Variables: 14
## $ crim      [3m [90m<dbl> [39m [23m 0.00632, 0.02731, 0.02729, 0.03237, 0.06905, 0.02985, ...
## $ zn        [3m [90m<dbl> [39m [23m 18.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.5, 12.5, 12.5, 12.5, ...
## $ indus      [3m [90m<dbl> [39m [23m 2.31, 7.07, 7.07, 2.18, 2.18, 2.18, 7.87, 7.87, 7.87, ...
## $ chas       [3m [90m<dbl> [39m [23m 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ nox        [3m [90m<dbl> [39m [23m 0.538, 0.469, 0.469, 0.458, 0.458, 0.458, 0.524, 0.524, ...
## $ rm         [3m [90m<dbl> [39m [23m 6.575, 6.421, 7.185, 6.998, 7.147, 6.430, 6.012, 6.172, ...
## $ age        [3m [90m<dbl> [39m [23m 65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96.1, 100.0, ...
## $ dis        [3m [90m<dbl> [39m [23m 4.0900, 4.9671, 4.9671, 6.0622, 6.0622, 6.0622, 5.5605, ...
## $ rad        [3m [90m<dbl> [39m [23m 1, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, ...
## $ tax        [3m [90m<dbl> [39m [23m 296, 242, 242, 222, 222, 222, 311, 311, 311, 311, 311, ...
## $ ptratio    [3m [90m<dbl> [39m [23m 15.3, 17.8, 17.8, 18.7, 18.7, 18.7, 15.2, 15.2, 15.2, ...
## $ b          [3m [90m<dbl> [39m [23m 396.90, 396.90, 392.83, 394.63, 396.90, 394.12, 395.60, ...
## $ lstat      [3m [90m<dbl> [39m [23m 4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19.15, 29.9, ...
## $ medv       [3m [90m<dbl> [39m [23m 24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, ...
```

The `chas` variable is mostly zero \Rightarrow should be a factor.

Initial Data Filtering

Select medv and lstat

```
boston <- boston_raw %>%  
  as_tibble() %>%  
  select(medv, lstat, chas) %>%  
  mutate(chas = as_factor(chas))  
  
head(boston)
```

```
## # A tibble: 6 x 3  
##   medv lstat chas  
##   <dbl> <dbl> <fct>  
## 1  24    4.98 0  
## 2  21.6   9.14 0  
## 3  34.7   4.03 0  
## 4  33.4   2.94 0  
## 5  36.2   5.33 0  
## 6  28.7   5.21 0
```

Step 2: Split the Data

Initial Split

We will use the `initial_split()`, `training()` and `testing()` functions from the **rsample** package to perform an initial train-test split

Set seed for reproducibility

```
set.seed(1203)
```

Initial split:

```
boston_split <- boston %>%  
  initial_split(prop = 2/3, strata = medv)
```

```
boston_split
```

```
## <338/168/506>
```

Prepare Training and Test Sets

```
boston_train_raw <- training(boston_split)
boston_test_raw  <- testing(boston_split)

head(boston_train_raw, 5)
```

```
## # A tibble: 5 x 3
##   medv lstat chas
##   <dbl> <dbl> <fct>
## 1  24    4.98  0
## 2  34.7  4.03  0
## 3  33.4  2.94  0
## 4  36.2  5.33  0
## 5  28.7  5.21  0
```

```
head(boston_test_raw, 5)
```

```
## # A tibble: 5 x 3
##   medv lstat chas
##   <dbl> <dbl> <fct>
## 1  21.6  9.14  0
## 2  16.5 29.9  0
## 3  18.9 17.1  0
## 4  18.9 13.3  0
## 5  20.4  8.26  0
```

Step 3: Explore the Data

Summary Statistics Using `skimr`

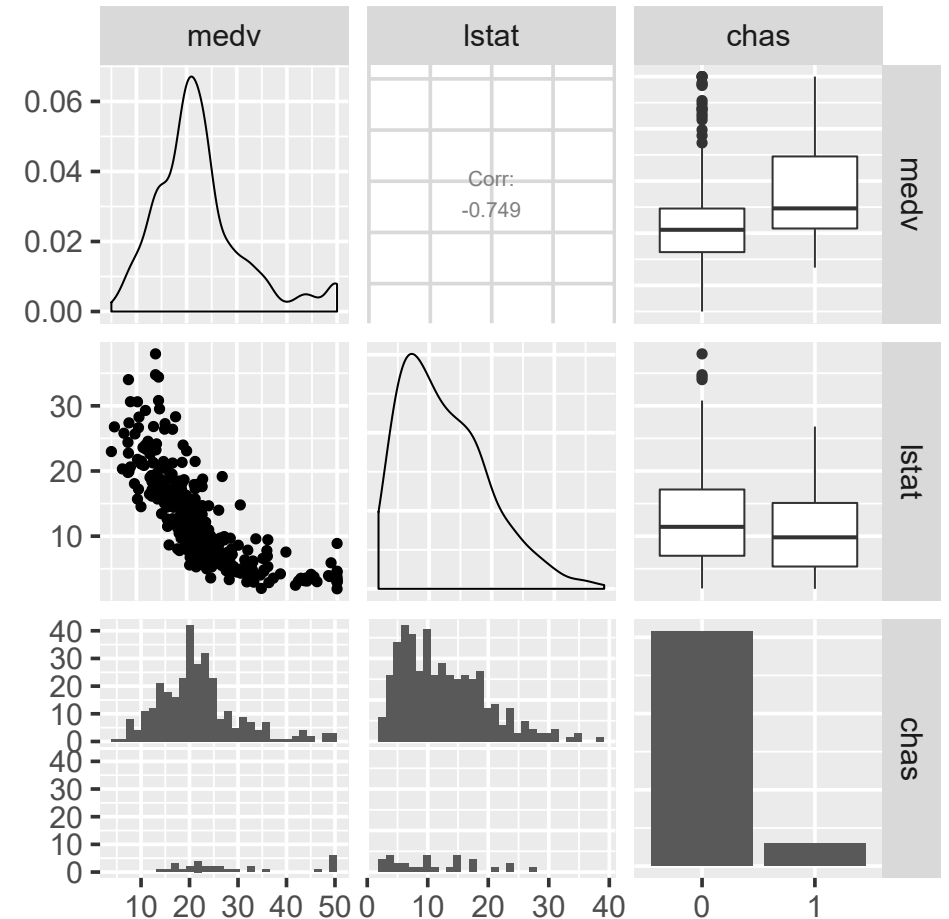
```
boston_train_raw %>%  
  skim() %>% partition()
```

(Does not come out well on these slides)

Pairs Plot Using GGally

We now use a **pairs plot** which compactly plots every variable in a dataset against every other one.

```
boston_train_raw %>% ggpairs()
```



Select a Model

We choose the class of polynomial models:

$$medv_i = \beta_0 + \sum_{j=1}^{\lambda} \beta_j lstat_i^j + \varepsilon_i$$

```
boston_train_raw %>% ggplot(aes(lstat, medv)) +  
  geom_point() +  
  geom_smooth(  
    method = lm,  
    formula = y ~ poly(x,1),  
    se = FALSE,  
    color = "blue"  
  ) +  
  geom_smooth(  
    method = lm,  
    formula = y ~ poly(x,10),  
    se = FALSE,  
    color = "red"  
  )
```


Step 4: Set Model and Tuning Parameters

Data Preprocessing using recipes

The `recipes` package is a great tool for data preprocessing that fits in naturally with the tidy approach to ML.

```
boston_rec <-  
  recipe(medv ~ lstat + chas, data = boston_train_raw) %>%  
    step_poly(lstat, degree = tune("lambda")) %>%  
    step_dummy(chas)
```

```
boston_rec
```

```
## Data Recipe  
##  
## Inputs:  
##  
##      role #variables  
## outcome      1  
## predictor      2  
##  
## Operations:  
##  
## Orthogonal polynomials on lstat  
## Dummy variables from chas
```

Set a Grid for λ

What are our tuning parameters?

```
boston_rec %>% parameters()
```

```
## Collection of 1 parameters for tuning  
##  
##      id parameter type object class  
## lambda          degree      nparam[+]
```

We need to tune the polynomial degree parameter (λ) when building our models on the train data. In this example, we will set the range between 1 and 8:

```
lambda_grid <- expand_grid("lambda" = 1:8)
```

Define the Model

We will use the linear regression model

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

```
lm_mod
```

```
## Linear Regression Model Specification (regression)  
##  
## Computational engine: lm
```

Note that there are no tuning parameters here.

Step 5: Cross-validation

Split the Training Set to 5-folds

We will use the `vfold_cv()` function from the **rsample** package to split the training set to 5-folds:

```
cv_splits <- boston_train_raw %>%  
  vfold_cv(v = 5)
```

```
cv_splits
```

```
## # 5-fold cross-validation  
## # A tibble: 5 x 2  
##   splits      id  
##   <named list> <chr>  
## 1 <split [270/68]> Fold1  
## 2 <split [270/68]> Fold2  
## 3 <split [270/68]> Fold3  
## 4 <split [271/67]> Fold4  
## 5 <split [271/67]> Fold5
```

Estimate CV-RMSE Over the λ Grid

We now estimate the CV-RMSE for each value of λ .

```
boston_cv <- tune_grid(  
  boston_rec,  
  model = lm_mod,  
  rs     = cv_splits,  
  grid   = lambda_grid)
```

```
boston_cv
```

```
## # 5-fold cross-validation  
## # A tibble: 5 x 4  
##   splits      id    .metrics      .notes  
## * <list>      <chr> <list>      <list>  
## 1 <split [270/68]> Fold1 <tibble [16 x 4]> <tibble [0 x 1]>  
## 2 <split [270/68]> Fold2 <tibble [16 x 4]> <tibble [0 x 1]>  
## 3 <split [270/68]> Fold3 <tibble [16 x 4]> <tibble [0 x 1]>  
## 4 <split [271/67]> Fold4 <tibble [16 x 4]> <tibble [0 x 1]>  
## 5 <split [271/67]> Fold5 <tibble [16 x 4]> <tibble [0 x 1]>
```

Find the Optimal λ

Let's find the top-3 performing models

```
rmse_vals <- boston_cv %>%  
  estimate() %>%  
  filter(.metric == "rmse") %>%  
  arrange(mean)  
  
head(rmse_vals, 3)
```

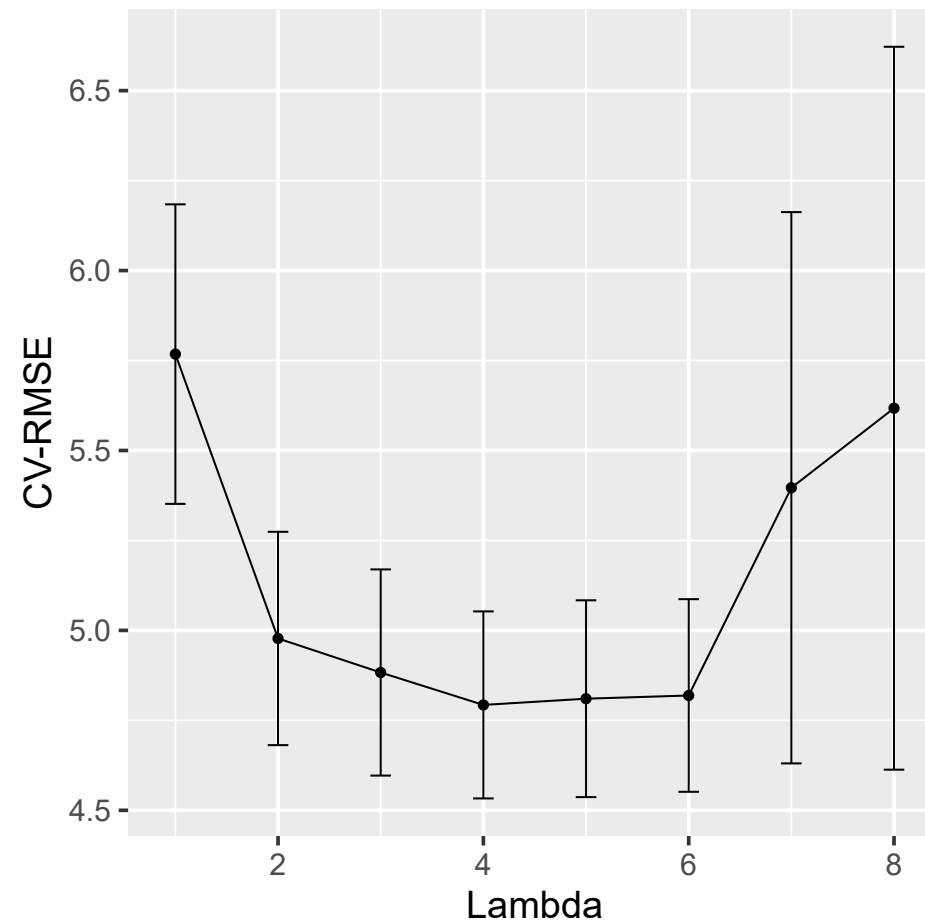
```
## # A tibble: 3 x 6  
##   lambda .metric .estimator  mean     n std_err  
##   <int> <chr>    <chr>    <dbl> <int>  <dbl>  
## 1     4 rmse     standard  4.79     5  0.260  
## 2     5 rmse     standard  4.81     5  0.273  
## 3     6 rmse     standard  4.82     5  0.268
```

"[I]n reality there is rarely if ever a true underlying model, and even if there was a true underlying model, selecting that model will not necessarily give the best forecasts..."

— **Rob J. Hyndman**

And Now Using a Graph

```
rmse_vals %>%  
  ggplot(aes(x = lambda, y = mean)) +  
  geom_point() +  
  geom_line() +  
  geom_errorbar(  
    aes(ymin = mean - std_err, ymax = mean + std_err,  
        width = 0.2)  
  ) +  
  labs(y = "CV-RMSE",  
       x = "Lambda")
```



Step 6: Evaluate the Model

Use the Test Set to Evaluate the Best Model

Prepare a recipe with the optimal $\lambda = 4$

```
boston_prep <-  
  recipe(medv ~ lstat + chas, data = boston_train_raw) %>%  
  step_poly(lstat, degree = 4) %>%  
  step_dummy(chas) %>%  
  prep()
```

Apply the Recipe to the Training and Test Sets

`juice()` applies the recipe to the training set and `bake()` to the test set.

```
boston_train <- boston_prep %>%  
  juice()  
  
boston_test <- boston_prep %>%  
  bake(new_data = boston_test_raw)
```

For example, let's take a look at the training set:

```
head(boston_train, 3)
```

```
## # A tibble: 3 x 6  
##   medv lstat_poly_1 lstat_poly_2 lstat_poly_3 lstat_poly_4 chas_X1  
##   <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>  
## 1  24      -0.0586      0.0449     -0.0214     -0.00896      0  
## 2  34.7     -0.0660      0.0627     -0.0515      0.0283      0  
## 3  33.4     -0.0744      0.0851     -0.0940      0.0917      0
```

Fit the Model to the Training Set

Fit the optimal model ($\lambda = 4$) to the training set:

```
boston_fit <- lm_mod %>%  
  fit(medv ~ ., data = boston_train)
```

Here are the estimated coefficients:

```
boston_fit %>% tidy()
```

```
## # A tibble: 6 x 5  
##   term          estimate std.error statistic    p.value  
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>  
## 1 (Intercept)    22.1      0.272      81.3 3.10e-221  
## 2 lstat_poly_1 -123.       4.78     -25.8 3.59e- 81  
## 3 lstat_poly_2   53.3      4.78      11.1 9.85e- 25  
## 4 lstat_poly_3  -20.1      4.80      -4.19 3.59e- 5  
## 5 lstat_poly_4   20.3      4.79       4.24 2.94e- 5  
## 6 chas_X1        4.76      0.924      5.15 4.43e- 7
```

Make Predictions Using the Test Set

Create a tibble with the predictions and ground-truth

```
boston_pred <- boston_fit %>%  
  predict(new_data = boston_test) %>%  
  bind_cols(boston_test) %>%  
  select(medv, .pred)  
  
head(boston_pred)
```

```
## # A tibble: 6 x 2  
##   medv .pred  
##   <dbl> <dbl>  
## 1  21.6  22.9  
## 2  16.5   9.80  
## 3  18.9  16.6  
## 4  18.9  18.7  
## 5  20.4  24.3  
## 6  18.2  21.4
```

Note that this is the first time we make use of the test set!

Test-RMSE

Calculate the test root mean square error (test-RMSE):

```
boston_pred %>%  
  rmse(medv, .pred)
```

```
## # A tibble: 1 x 3  
##   .metric .estimator .estimate  
##   <chr>   <chr>      <dbl>  
## 1 rmse    standard      5.94
```

The above is a measure of our model's performance on "general" data.

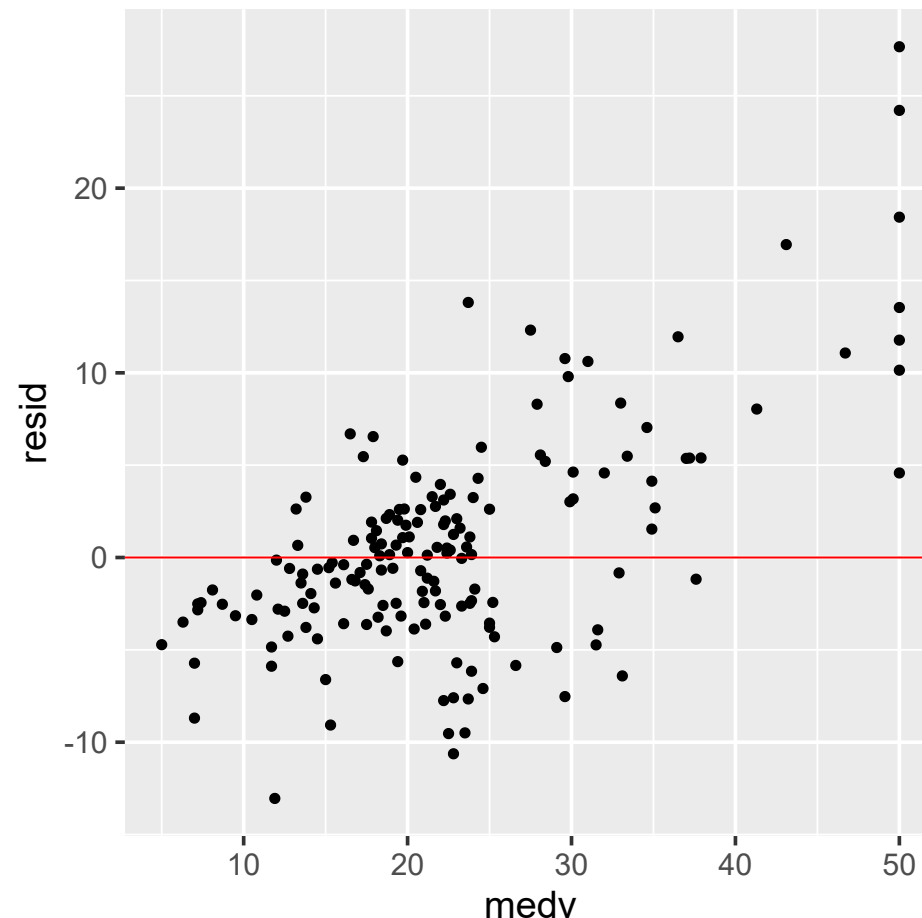
NOTE: the test set RMSE estimates the expected squared prediction error on unseen data *given* the best model.

Always plot your prediction errors

Plotting the prediction errors ($y_i - \hat{y}_i$) vs. the target provides valuable information about prediction quality.

```
boston_pred %>%  
  mutate(resid = medv - .pred) %>%  
  ggplot(aes(medv, resid)) +  
  geom_point() +  
  geom_hline(yintercept = 0, color = "red")
```

For example, our predictions for high-end levels of `medv` are extremely biased \Rightarrow there's room for improvement...




```
slides::end()
```

 [Source code](#)