Boston Housing

Lavanya N

3/2/2021

Linear regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It is used to predict values within a continuous range.

The set.seed function is called once to ensure that the knitting is conditionally deterministic.

```r
set.seed(12345)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(tidymodels)
```

```
## -- Attaching packages --------------------------------- tidymodels 0.1.2.9000 --

## v broom     0.7.2      v rsample   0.0.8
## v dials     0.0.9      v tibble    3.0.4
## v infer     0.5.3      v tidyr     1.1.2
## v modeldata 0.1.0      v tune      0.1.2
## v parsnip   0.1.4      v workflows 0.2.1
## v purrr     0.3.4      v yardstick 0.0.7
## v recipes   0.1.15

## Warning: package 'broom' was built under R version 4.0.3

## Warning: package 'dials' was built under R version 4.0.3

## Warning: package 'infer' was built under R version 4.0.3

## Warning: package 'modeldata' was built under R version 4.0.3
## Warning: package 'parsnip' was built under R version 4.0.3

## Warning: package 'recipes' was built under R version 4.0.3

## Warning: package 'rsample' was built under R version 4.0.3

## Warning: package 'tibble' was built under R version 4.0.3

## Warning: package 'tune' was built under R version 4.0.3

## Warning: package 'workflows' was built under R version 4.0.3

## Warning: package 'yardstick' was built under R version 4.0.3

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
```

A linear regression model will be trained and tested using the `BostonHousing` dataset. It contains information collected by the U.S Census Service concerning housing in the area of Boston, Massachusetts. The dataset is small in size with only 506 cases. The details of the variables can be viewed here: https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html.

```
data(BostonHousing, package = "mlbench")
str(BostonHousing, max.level = 1)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : num  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ b      : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

In supervised learning, training data is used to solve some optimization problem. A corresponding matrix of predictors and outcome vector form the testing data, which are additional observations that are not used to obtain regression parameters and are instead used to evaluate the model that produced them. It is important to ensure that the trained model is tested on a different set of data to avoid biasing.

In this case, 80% of the observations is assigned to the training data, while the remainder forms the testing data.

```
BH_split <- initial_split(BostonHousing, prob = 0.80)
BH_train <- training(BH_split)
BH_test  <- testing(BH_split)
```

1. Linear Regression

The data is first prepared for modeling. In this step, variables to be included in the model should be specified. All variables are being included in this model. Subsequently, the type of model or algorithm should be specified. In this case, it is a linear regression model.

```
BH_recipe <- recipe(medv ~ ., data = BH_train) %>%
  step_dummy(all_nominal())
BH_recipe <- prep(BH_recipe, training = BH_train)
lm_model <- linear_reg() %>% set_engine("lm")
lm_wflow <- workflow() %>% add_model(lm_model) %>% add_formula(medv ~ .)
lm_fit <- fit(lm_wflow, data = bake(BH_recipe, new_data = NULL))
```

Finally, predictions of the model can be obtained in the testing data and the root-mean squared error is calculated. In this case, it is 4.482. This value has more meaning when there are multiple linear models to compare for the same purpose.

```
y_hat <- predict(lm_fit, new_data = bake(BH_recipe, new_data = BH_test))
```

```
rmse(bind_cols(select(BH_test, medv), y_hat), truth = medv, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        4.05
```

2. Linear Regression with Penalisation

The standard linear model may perform poorly in situations with a large multivariate dataset that has a smaller number of observations. In such scenarios, a penalised regression model may be a better alternative as it allows the addition of a constraint in the equation and creation of a model that is penalised for having too many variables/fewer observations. The consequence of imposing this penalty, is to reduce (i.e. shrink) the coefficient values towards zero. This allows the less contributive variables to have a coefficient close to zero or equal zero. The shrinkage requires the selection of a tuning parameter (lambda) that determines the amount of shrinkage. Tuning parameters can be pre-specified or selected optimally.

Since variables are being compared to one another but have different units, they should be standardised using `step_center` and `step_scale`. The penalty is also pre-specified to have a value of 0.001.

```
norm_recipe <-
  recipe(medv ~ ., data = BH_train) %>%
  step_zv(all_predictors()) %>%
  step_lincomb(all_numeric()) %>%
  step_dummy(all_nominal()) %>%
  step_center(all_predictors()) %>%
```

```r
  step_scale(all_predictors()) %>%
  prep(training = BH_train)

glmn_fit <-
  linear_reg(penalty = 0.001, mixture = 0.5) %>%
  set_engine("glmnet") %>%
  fit(medv ~ ., data = bake(norm_recipe, new_data = NULL))
```

The penalised regression model has a rmse of 4.467, which is lower than that of the original linear regression model.

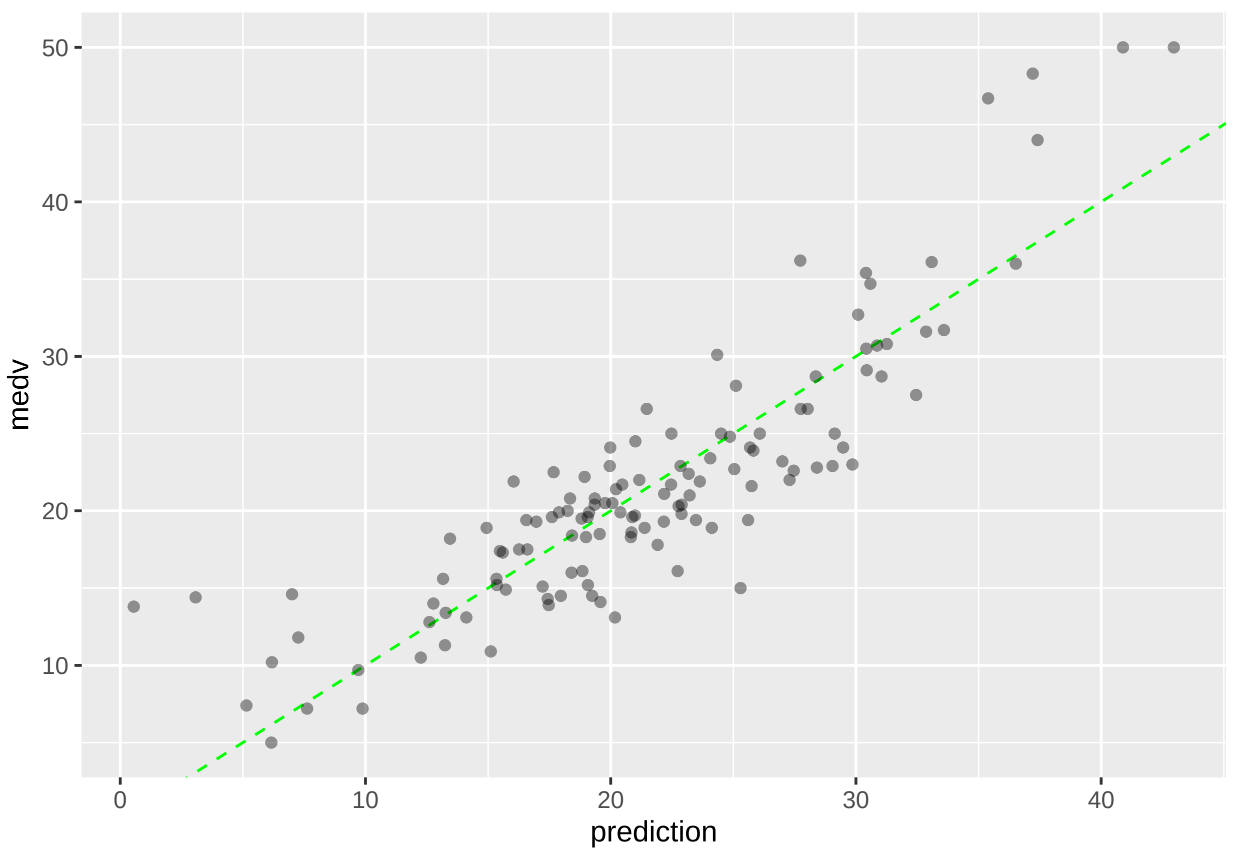
```r
test_normalized <- bake(norm_recipe, new_data = BH_test, all_predictors())

test_results <- bind_cols(select(BH_test, medv),
                          predict(glmn_fit, new_data = test_normalized))
rmse(test_results, truth = medv, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        4.04
```

The plot presents an alternate way of evaluating the performance of the model. The model performs well for a certain range of `medv` values.

```r
test_results %>%
  gather(model, prediction, -medv) %>%
  ggplot(aes(x = prediction, y = medv)) +
  geom_abline(col = "green", lty = 2) +
  geom_point(alpha = .4)
```



3. Linear Regression with Penalisation (Grid Search)

Instead of specifying the tuning parameters, grid search can be employed to find the optimal parameters. This performs cross-validation to see which value of the tuning parameter yields the best predictions outside the data that are used to obtain the coefficients. In this, the optimal parameter is chosen from 50 possible options.

```
BH_rs <- bootstraps(BH_train, times = 50)
tune_spec <- linear_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")
wf <- workflow() %>% add_recipe(norm_recipe) %>%
  remove_formula() %>% add_formula(medv ~ .)
```

```
## Warning: The workflow has no formula preprocessor to remove.
```

The value of the tuning parameter is selected such that the resulting regression model has the lowest rmse.

```
my_grid <- grid_regular(penalty(), mixture(), levels = 10)
doParallel::registerDoParallel()
results <- tune_grid(wf %>% add_model(tune_spec),
                     resamples = BH_rs, grid = my_grid)
lowest_rmse <- results %>% select_best("rmse")
final <- finalize_workflow(wf %>% add_model(tune_spec), lowest_rmse)
```

The penalised regression model using grid search has a rmse of 4.434, which is lower than that of the other 2 models.

```r
final_fit <- fit(final, data = bake(norm_recipe, new_data = NULL))
final_results <- bind_cols(select(BH_test, medv),
                          predict(final_fit, new_data = test_normalized))
rmse(final_results, truth = medv, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        4.03
```

4. Conclusion

The penalised regression model using grid search yielded the best predictions in the testing data in terms of rmse.