Linear Regression for Prediction & the `predict` Function

Machine Learning - Sections 3.1.1 & 3.1.2

Marc Omar Haddad

Published: 9 February, 2020

Updated: 12 February, 2020

Linear Regression can be considered to be a form of Machine Learning. Although it is too rigid to be useful in general, it can be very effective in certain cases. It is also a baseline approach to Machine Learning in that it is often used if more complex methods are impractical.

Linking Linear Regression and Machine Learning

We can use Galton's data set to exhibit the link between Linear Regression and Machine Learning.

```r
library(HistData)

galton_heights = GaltonFamilies %>%
  filter(childNum == 1 & gender == "male") %>%
  select(father, childHeight) %>%
  rename(son = childHeight)
```

Our task is to build a Machine Learning algorithm that predicts the `son`'s height Y using the `father`'s height X.

First, we generate our `test_set` and `train_set`.

```r
y = galton_heights$son
test_index = createDataPartition(y, times = 1, p = 0.5, list = FALSE)

train_set = galton_heights %>% slice(-test_index)
test_set = galton_heights %>% slice(test_index)
```

To see if our eventual algorithm performs better than merely guessing, we create an algorithm that estimates `son` by simply finding the average of all `son` heights in our `train_set`, and calculating for R^2 Loss.

```r
avg = mean(train_set$son)
avg
```

```
## [1] 70
```

```r
mean((avg - test_set$son)^2) # R^2 Loss
```

```
## [1] 7.8
```

Our goal is to construct an algorithm that is better than the one above.

We know from our earlier lesson on Linear Regression that if both our variables (X, Y) follow a **bivariate normal distribution**, the **Conditional Expectation** is *equal* to the **Regression Line**.

$$f(x) = \mathrm{E}(Y \mid X = x) = \beta_0 \; + \; \beta_1 x$$

In `R`, we can calculate the values β_0 and β_1 with the following simple formula:

```r
fit = lm(son ~ father, data = train_set)
fit$coef
```

```
## (Intercept)      father
##        42.8         0.4
```

This gives us an estimate of the Conditional Expectation:

$$\hat{f}(x) = 42.8 + 0.4x$$

With x being equal to `father` height.

Now we assess our function to see if we've improved upon our initial "estimate" function.

```r
y_hat = fit$coef[1] + fit$coef[2] * test_set$father
mean((y_hat - test_set$son)^2)
```

```
## [1] 5.8
```

As we can see, our algorithm does indeed perform better than simply estimating the son's height, as evidenced by our lower R^2 Loss value.

The `predict` Function

The `predict` function takes two arguments: a fitted object from functions like `lm` and `glm`, and a dataframe with our new predictors for which we want to predict; `predict` then **returns a prediction**.

Instead of having to write out the formula for our earlier regression line, we can use `predict`:

```r
y_hat = predict(fit, test_set)
mean((y_hat - test_set$son)^2) # This returns the same R^2 from earlier
```

```
## [1] 5.8
```

One caveat with `predict` is that it does not necessarily always return objects of the same type. In these scenarios it is useful to look at the help file for the type of fitted object that is being used. For example: `?predict.lm` and `?predict.glm`. This will be useful in the future when we use functions such as k-NN (`?predict.knn`).