Linear Regression Example

Jim Harner

6/7/2019

6.3 Concrete Slump Test Regression

Load `slump.csv` into Spark with `spark_read_csv` from the local filesystem.

```
slump_sdf <- spark_read_csv(sc, "slump_sdf",
                path =  "file:///home/rstudio/rspark-tutorial/data/slump.csv")
head(slump_sdf)
```

```
## # Source: spark<?> [?? x 10]
##   cement  slag fly_ash water    sp coarse_aggr fine_aggr slump  flow
##    <dbl> <dbl>   <dbl> <dbl> <dbl>       <dbl>     <dbl> <dbl> <dbl>
## 1    273    82     105   210     9         904       680    23  62
## 2    163   149     191   180    12         843       746     0  20
## 3    162   148     191   179    16         840       743     1  20
## 4    162   148     190   179    19         838       741     3  21.5
## 5    154   112     144   220    10         923       658    20  64
## 6    147    89     115   202     9         860       829    23  55
## # ... with 1 more variable: compressive_strength <dbl>
```

First we need to split `slump_sdf` into a training and a test Spark DataFrame.

```
slump_partition <- tbl(sc, "slump_sdf") %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 2)
slump_train_sdf <- slump_partition$training
slump_test_sdf <- slump_partition$test
```

The full model is now run.

```
slump_lr_full_model <- slump_partition$training %>%
  ml_linear_regression(compressive_strength ~ cement + slag + fly_ash + water
                       + sp + coarse_aggr + fine_aggr)
summary(slump_lr_full_model)
```

```
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -5.6280 -1.6192 -0.3183  0.9372  7.1920
##
## Coefficients:
##  (Intercept)       cement         slag      fly_ash        water
## 219.36232986   0.03777496  -0.06065688   0.02819246  -0.31892157
##           sp  coarse_aggr    fine_aggr
##  -0.12983604  -0.08744781  -0.06805072
##
## R-Squared: 0.8987
## Root Mean Squared Error: 2.507
```

Notice that the model summary does not provide much useful information. We can get p-values from a `tidy` summary.

```
tidy(slump_lr_full_model)
```

```
## # A tibble: 8 x 5
##   term         estimate std.error statistic p.value
##   <chr>           <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)  219.       92.4        2.37  0.0207
## 2 cement         0.0378    0.0290     1.30  0.198
## 3 slag          -0.0607    0.0409    -1.48  0.143
## 4 fly_ash        0.0282    0.0301     0.938 0.352
## 5 water         -0.319     0.0927    -3.44  0.00104
## 6 sp            -0.130     0.183     -0.708 0.481
## 7 coarse_aggr   -0.0874    0.0355    -2.46  0.0166
## 8 fine_aggr     -0.0681    0.0379    -1.79  0.0778
```

The performance metrics on the training data can be extracted from the `ml_model` object:

```
data.frame(lambda = 0,
           r2 = slump_lr_full_model$summary$r2,
           rmse = slump_lr_full_model$summary$root_mean_squared_error,
           mae = slump_lr_full_model$summary$mean_absolute_error)
```

```
##   lambda        r2    rmse      mae
## 1      0 0.8987442 2.50723 1.896407
```

However, we actually want these metrics on the test data set.

Performance metrics for regression are now obtained by getting predictions using the training data based on the full model and then using the `ml_regression_evaluator` to get specific metrics.

```
slump_lr_full_predict <- ml_predict(slump_lr_full_model, slump_train_sdf)
slump_lr_metrics <-
  data.frame(lambda = 0,
             rmse = ml_regression_evaluator(slump_lr_full_predict,
                                            label_col = "compressive_strength",
                                            metric_name = "rmse"),
             mae = ml_regression_evaluator(slump_lr_full_predict,
                                           label_col = "compressive_strength",
                                           metric_name = "mae"))
slump_lr_coef <- as.data.frame(slump_lr_full_model$coefficients)
```

This is done initially for $\lambda = 0$.

The model function for the lasso with varying values of the regularization parameter λ is defined by:

```
slump_lr_model <- function(l) {
  slump_train_sdf %>%
    ml_linear_regression(compressive_strength ~ cement + slag + fly_ash +
                         water + sp + coarse_aggr + fine_aggr,
                         elastic_net_param = 1, reg_param = l)
}
```

We now calculate the `rmse` and `mae` for each of the models.

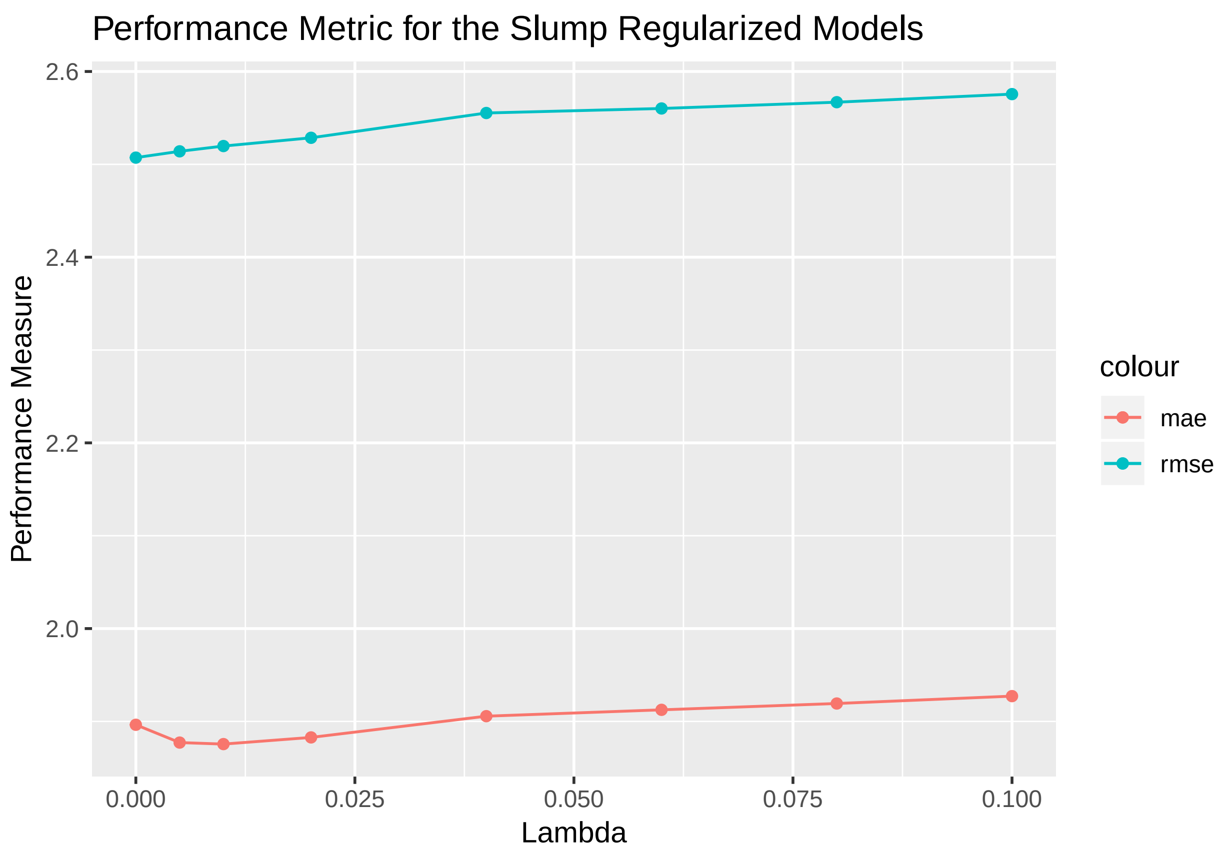
```
reg_parm <- c(0.005, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1)
for(l in reg_parm) {
  slump_lr_predict <- slump_lr_model(l) %>%
    ml_predict(slump_train_sdf)
  slump_lr_metrics <-
    data.frame(lambda = l,
               rmse = ml_regression_evaluator(slump_lr_predict,
                                              label_col = "compressive_strength",
```

```
                                                metric_name = "rmse"),
               mae = ml_regression_evaluator(slump_lr_predict,
                                            label_col = "compressive_strength",
                                            metric_name = "mae")) %>%
    rbind(slump_lr_metrics, .)
  slump_lr_coef <-
    as.data.frame(slump_lr_model(l)$coefficients) %>%
    cbind(slump_lr_coef, .)
}
slump_lr_metrics
```

```
##   lambda     rmse      mae
## 1  0.000 2.507230 1.896407
## 2  0.005 2.514022 1.877238
## 3  0.010 2.519696 1.875591
## 4  0.020 2.528594 1.882784
## 5  0.040 2.555291 1.905668
## 6  0.060 2.560160 1.912491
## 7  0.080 2.566941 1.919315
## 8  0.100 2.575633 1.927251
```

Finally, we plot the performance measures.

```
library(ggplot2)
slump_lr_metrics %>%
  ggplot(aes(x = lambda)) +
  geom_point(aes(y = rmse, color = 'rmse')) +
  geom_line(aes(y = rmse, color = 'rmse')) +
  geom_point(aes(y = mae, color = 'mae')) +
  geom_line(aes(y = mae, color = 'mae')) +
  ggtitle("Performance Metric for the Slump Regularized Models") +
  xlab("Lambda") + ylab("Performance Measure")
```



Based on the performance metrics, it is clear we want `lambda` to be small, e.g., $\lambda = 0.005$ or 0.01. However, we also want parsimony.

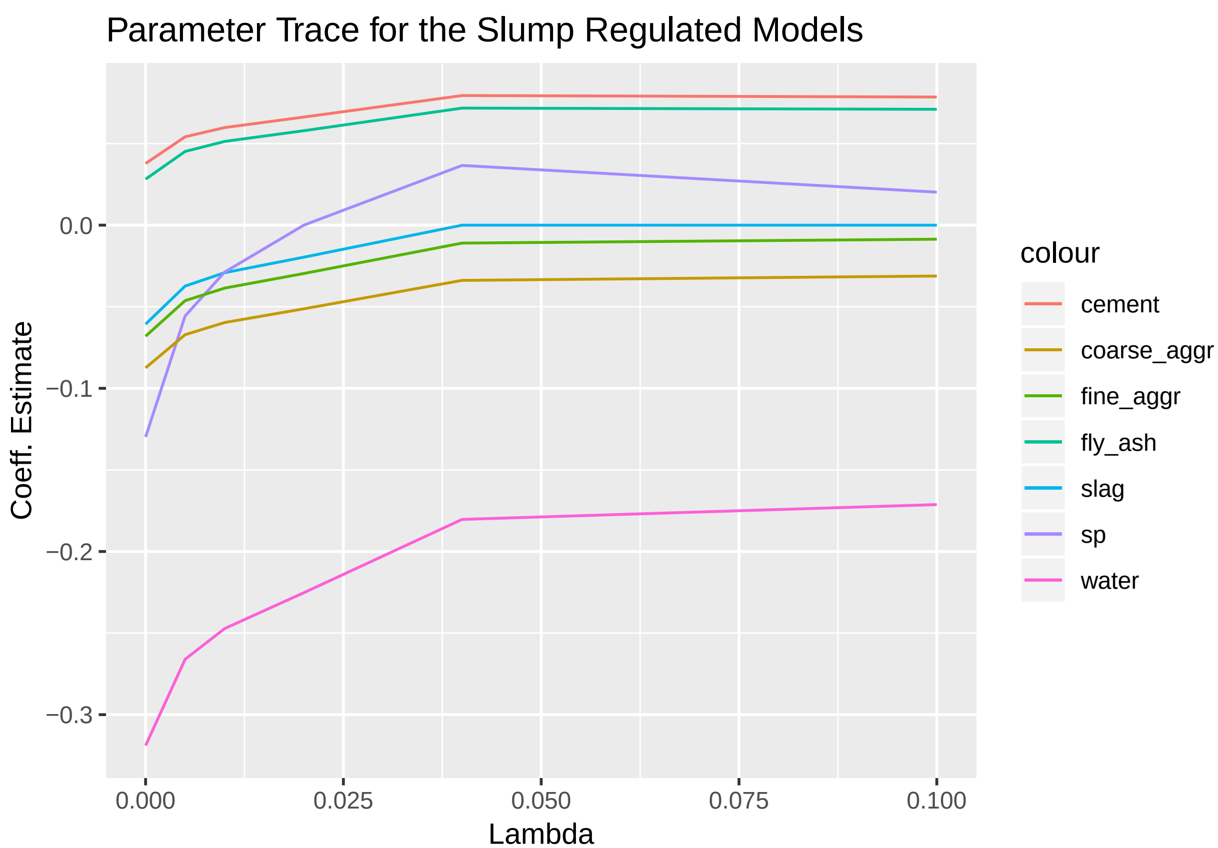
We now get the parameter estimates as `lambda` increases.

```
names(slump_lr_coef) <- as.character(rbind(c(0.0, reg_parm)))
slump_lr_coef <- t(slump_lr_coef)
slump_lr_coef
```

```
##       (Intercept)     cement        slag    fly_ash      water          sp
## 0       219.36233 0.03777496 -0.06065688 0.02819246 -0.3189216 -0.12983604
## 0.005   166.09655 0.05416211 -0.03732442 0.04517294 -0.2660410 -0.05562986
## 0.01    147.03917 0.05981071 -0.02912546 0.05131044 -0.2472031 -0.02873438
## 0.02    125.22235 0.06625311 -0.01963970 0.05789226 -0.2252704  0.00000000
## 0.04     80.23832 0.07943291  0.00000000 0.07174224 -0.1803432  0.03659909
## 0.06     78.41981 0.07912863  0.00000000 0.07149383 -0.1772967  0.03114239
## 0.08     76.60486 0.07882439  0.00000000 0.07125072 -0.1742739  0.02567742
## 0.1      74.78988 0.07852014  0.00000000 0.07100761 -0.1712511  0.02021238
##       coarse_aggr    fine_aggr
## 0     -0.08744781 -0.068050721
## 0.005 -0.06710968 -0.046269057
## 0.01  -0.05966800 -0.038586854
## 0.02  -0.05126774 -0.029635158
## 0.04  -0.03384715 -0.010993751
## 0.06  -0.03296010 -0.010196193
## 0.08  -0.03207201 -0.009399424
## 0.1   -0.03118391 -0.008602646
```

The lasso trace of the coefficient estimates provides a way of picking the strength of regulation.

```
library(ggplot2)
as.data.frame(cbind(lambda = c(0.0, reg_parm), slump_lr_coef)) %>%
  ggplot(aes(x = lambda)) +
  geom_line(aes(y = cement, color = 'cement')) +
  geom_line(aes(y = slag, color = 'slag')) +
  geom_line(aes(y = fly_ash, color = 'fly_ash')) +
  geom_line(aes(y = water, color = 'water')) +
  geom_line(aes(y = sp, color = 'sp')) +
  geom_line(aes(y = coarse_aggr, color = 'coarse_aggr')) +
  geom_line(aes(y = fine_aggr, color = 'fine_aggr')) +
  ggtitle("Parameter Trace for the Slump Regulated Models") +
  xlab("Lambda") + ylab("Coeff. Estimate")
```



Over the range of λ, we have 3 features (`cement`, `fly_ash`, and `water`) with consistently non-zero coefficient estimates. Arguably, `coarse_aggr` also deviates from 0. These agree with the model we found by *ad hoc* variable selection in Section 6.1.

At this point we pick a reasonable model to run on the test Spark DataFrame based on the above criteria.

```
slump_train_sdf %>%
  ml_linear_regression(compressive_strength ~ cement + fly_ash + water + coarse_aggr,
                       alpha = 1, lambda = 0.005) %>%
  ml_predict(slump_test_sdf) %>%
  ml_regression_evaluator(label_col = "compressive_strength", metric_name = "rmse")
```

```
## [1] 2.792653
```

The RMSE is somewhat above that obtained for the training data. Several other models could be run, e.g., removing the `coarse_aggr` feature or changing λ to 0.01. This would suggest that we need a training data set to narrow the field of possible models, a validation data set to hone into the "best" model, and a test data set for the final model.

The above approach uses a fitting process that involves human curation. Generally this is a good idea during model development. However, production models should be fully automated. This can be done using `ml_train_validation_split()` (or `ml_cross_validator` for k-fold cross validation) with arguments: an `ml_estimator` object (possibly an `ml_pipeline`), an `estimator_param_maps` object, and an `ml_evaluator` object. The resulting train-validation-split model could then be piped into `ml_validation_metrics()` to get a data frame of performance metrics for all combinations of hyperparameters.

```r
spark_disconnect(sc)
```

```
## NULL
```