Linear Regression

Jim Harner

1/12/2021

`sparklyr` requires a `dplyr` compatible back-end to Spark.

```r
library(dplyr, warn.conflicts = FALSE)

library(sparklyr)
# master <- "spark://master:7077"
master <- "local"
sc <- spark_connect(master = master)
```

6.1 Linear Regression

Linear regression models the linear relationship between an outcome variable (dependent or response variable) and one or more explanatory variables (predictors, independent variables, or features). Both the outcome and predictor variables are numeric. *Linearity* is an assumption that should be checked. In some cases it is difficult to assume linearity except locally.

6.1.1 Linear Regression Basics

The simple linear regression can be expressed as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon.$$

where ϵ is the *error term* or *noise* variable and the x_j are the predictors or features. For the standard regression model, $\epsilon \sim N(0, \sigma^2)$, i.e., the variability is assumed to be constant over the range of x.

The strategy is to minimize:

$$RSS(\beta_0, \beta_1, \dots, \beta_p) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip})^2$$

with respect to the β's. RSS is the basic loss function for regression. Later this loss will be generalized by constraining (regularizing) the coefficients, i.e., shrinking the coefficients towards 0. This often reduce the coefficient variances without appreciably increasing the bias.

The observed errors or *residuals* are given by:

$$e_i = y_i - \hat{y}_i,$$

where $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}$ is the predicted value for the i^{th} observation.

The residual sum of squares is given by

$$RSS = \sum e_i^2.$$

and the estimated variance of ϵ is

$$\hat{\sigma}^2 = \frac{RSS}{n - p - 1}.$$

The residual standard error (RSE) is simply the square root of the estimated variance:

$$RSE = \sqrt{\frac{RSS}{n-p-1}},$$

which estimates σ.

R^2, is called the *coefficient of determination*— the proportion of the variability explained by the model. It is given by:

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2}.$$

The term on the right is the proportion of unexplained variability, i.e., the residual sum of squares divided by the total error.

The principal hypothesis of interest is:

$$H_o : \beta_j = 0 \text{ vs. } H_a : \beta_j \neq 0,$$

i.e., the coefficient for the j^{th} predictor is 0. In order to test this hypothesis, we compute a t-test as follows:

$$t = \frac{\hat{\beta}_j}{\text{s.e.}(\hat{\beta}_j)},$$

where s.e.$(\hat{\beta}_j)$ is the estimated standard error of $\hat{\beta}_j$. The estimated variances of the estimators are given by the diagonal elements of:

$$\hat{\text{Var}}(\hat{\beta}) = \hat{\sigma}^2 (X^t X)^{-1}$$

$(X^t X)^{-1}$ is called the unscaled covariance matrix.

The p-value is the probability of getting a test statistics as extreme or more extreme than the observed value under the *null hypothesis*. This is computed as $P(t \geq |t_{\text{obs}}|)$ or by `Pr( >|t|)` in R.

6.1.2 Determining relevant predictors

How do we select which predictors (features) are important?

Stepwise selection **Forward stepwise selection** begins with a model containing no predictors, and then adds predictors to the model one-at-a-time until all of the predictors are in the model.

The algorithm is simple:

1. Start with M_0, the null model.

2. For $k = 0, 1, \ldots, p-1$ augment the predictors in M_k with one additional predictor and then pick the one with the highest R^2 or lowest RSS. Call this M_{k+1}.

3. Select the best model from $M_0, M_1, \ldots, M_p$ using cross validation prediction error, C_p, AIC, BIC, or the adjusted R^2.

The method has far fewer models $(1 + p(p+1)/2)$ than best subset selection (2^p). Also, this method can be used for the high-dimensional cases when $p > n$.

Backward stepwise selection begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time.

The algorithm follows:

1. Start with M_p, the model with all predictors.

2. For $k = p, p-1, \ldots, 1$ consider all k models that contain all but one of the predictors in M_k, i.e., each containing k_1 predictors, and then pick the one with the highest R^2 or lowest RSS. Call this M_{k-1}.

3. Select the best model from $M_0, M_1, \ldots, M_p$ using cross validation prediction error, C_p, AIC, BIC, or the adjusted R^2.

The backward selection approach searches $1 + p(p+1)/2$ models. In this case, n must be larger than p.

Hybrid versions of forward and backward stepwise selection: variables are added to the model sequentially, but after adding each new variable, the method may also remove any variables that no longer provide an improvement in the model fit. Such an approach attempts to more closely mimic best subset selection while retaining the computational advantages of forward and backward stepwise selection.

Optimal models In the above stepwise procedures, how do we select the best model in step 3? We need the model with the lowest test error. To estimate the test error, we need to:

- adjust the training error to account for bias due to overfitting, or

- estimate the test error directly using a validation set or by cross validation.

C_p, the *Akaike information criterion (AIC)*, the *Bayesian information criterion(BIC)*, and the *adjusted* R^2 are methods for adjusting the training error for model complexity.

Mallow's C_p is computed as:

$$C_p = \frac{RSS_k}{\hat{\sigma}^2} + 2k - n,$$

where RSS_k is the RSS based on k predictors in the model and $\hat{\sigma}^2 = RSS_p/(n-p)$ is an estimate of $\text{Var}(\epsilon)$ for the full model. If $\hat{\sigma}^2$ is unbiased, then $\hat{\sigma}^2(C_p + n) = RSS_k + 2k\hat{\sigma}^2$ is an unbiased estimate of $n \times MSE$. Notice that $2k\hat{\sigma}^2$ is a model complexity penalty term.

For $k = p$, $C_p = p$. If the k predictor model fits, then $E(RSS_k) = (n-k)\sigma^2$ and $E(C_p) \approx k$. If it is a bad fit, then $C_p > k$. Thus, we want the smallest k with $C_p \leq k$.

The *AIC criterion* is given by:

$$AIC = -2 \times \text{log-likelihood} + 2k,$$

where $-2 \times \text{log-likelihood} = n \log(RSS_k/n)$ is called the deviance.

The *BIC criterion* is given by:

$$BIC = -2 \times \text{log-likelihood} + \log(n)k,$$

If $n > 7$, then the penalty term for BIC exceeds that of AIC.

These statistics tend to take on small values for models with a low test error. We choose k to minimize the AIC or BIC.

The adjusted R^2 statistic is calculated as

$$\text{Adjusted } R^2 = 1 - \frac{RSS/(n-k-1)}{TSS/(n-1)}.$$

A large value of the adjusted R^2 indicates a model with a small test error or equivalents we could minimize $RSS/(n-k-1)$.

Performance Metrics We use different performance metrics for different kinds of models, and in different contexts. For linear regression we typically use:
* **Mean squared error (MSE)**: This is the average squared distance between the predicted and actual

values.

$$\text{MSE} = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

* **Root mean squared error (RMSE)**: The square root of the mean squared error.

$$\text{RMSE} = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{n}}$$

* **Mean absolute error (MAE)**: The average of the absolute value of the difference between the predicted and actual values.

$$\text{MAE} = \frac{\sum|y_i - \hat{y}_i|}{n}$$

The latter two are most often used since they are in the same scale as the response variable.

Cross-validation of these performance metrics was discussed in Section 1.6.

6.1.3 Concrete Slump Test Data

The `slump.csv` file was loaded into Hadoop. Assuming you have not removed it with the `hdfs.rm` function, you can load the data into Spark from Hadoop using the `sparklyr`'s `spark_read_csv` function, which creates a Spark DataFrame.

Alternately, you can load `slump.csv` into Spark directly with `spark_read_csv` from the local filesystem.

```r
# slump_sdf <- spark_read_csv(sc, "slump_sdf",
#                    path = "hdfs://hadoop:9000/user/rstudio/data/slump.csv")
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>
```

`header = TRUE` is the default for `spark_read_csv`.

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

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

Initially, we fit a model with just `fly_ash`, which is thought to be the best single predictor of `compressive_strength`. This is difficult to check since their is no automatic selection method in Spark other than regularization.

```r
slump_lr_p1_fit <- slump_partition$training %>%
  ml_linear_regression(compressive_strength ~ cement)
summary(slump_lr_p1_fit)
```

```
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -14.030  -4.622   0.540   4.920  16.905
##
## Coefficients:
## (Intercept)      cement
##  23.0537235   0.0559365
##
## R-Squared: 0.2826
## Root Mean Squared Error: 7.082
```

```
tidy(slump_lr_p1_fit)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  23.1       2.54        9.07 1.37e-13
## 2 cement        0.0559    0.0104      5.36 9.23e- 7
```

`cement` is highly significant, but the R^2 is low.

The full model is now run.

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

```
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7501 -1.6642 -0.2428  1.2498  6.9504
##
## Coefficients:
##  (Intercept)       cement          slag      fly_ash         water            sp
## 117.20416259   0.07075548  -0.01835116   0.05690188  -0.21973660  -0.04664274
##  coarse_aggr    fine_aggr
##  -0.04619533  -0.02701631
##
## R-Squared: 0.9074
## Root Mean Squared Error: 2.545
```

```
tidy(slump_lr_full_fit)
```

```
## # A tibble: 8 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept) 117.       82.0         1.43  0.158
## 2 cement        0.0708    0.0265      2.67  0.00946
## 3 slag         -0.0184    0.0368     -0.498 0.620
## 4 fly_ash       0.0569    0.0266      2.14  0.0358
## 5 water        -0.220     0.0829     -2.65  0.00998
## 6 sp           -0.0466    0.180      -0.259 0.796
## 7 coarse_aggr  -0.0462    0.0318     -1.45  0.151
## 8 fine_aggr    -0.0270    0.0331     -0.815 0.418
```

$R^2 = 0.907$, but some of the variables are not significant.

We eliminate `sp` since it has the largest p-value.

```
slump_lr_p6_fit <- slump_partition$training %>%
  ml_linear_regression(compressive_strength ~ cement + slag + fly_ash + water
                       + coarse_aggr + fine_aggr)
summary(slump_lr_p6_fit)
```

```
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7398 -1.6676 -0.2371  1.2032  7.0510
##
## Coefficients:
##  (Intercept)       cement         slag      fly_ash        water  coarse_aggr
## 105.00323631   0.07453852  -0.01328025   0.06089636  -0.20758187  -0.04157104
##    fine_aggr
##  -0.02232803
##
## R-Squared: 0.9073
## Root Mean Squared Error: 2.546
```

```
tidy(slump_lr_p6_fit)
```

```
## # A tibble: 7 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept) 105.       66.8        1.57  0.120
## 2 cement        0.0745    0.0220     3.40  0.00115
## 3 slag         -0.0133    0.0310    -0.428 0.670
## 4 fly_ash       0.0609    0.0215     2.83  0.00604
## 5 water        -0.208     0.0679    -3.06  0.00318
## 6 coarse_aggr  -0.0416    0.0261    -1.59  0.116
## 7 fine_aggr    -0.0223    0.0276    -0.810 0.421
```

$R^2 = 0.907$ is as high as for the full model.

We next remove slag.

```
slump_lr_p5_fit <- slump_partition$training %>%
  ml_linear_regression(compressive_strength ~ cement + fly_ash + water + coarse_aggr
                       + fine_aggr)
summary(slump_lr_p5_fit)
```

```
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8517 -1.7021 -0.2716  1.2789  7.0615
##
## Coefficients:
## (Intercept)      cement     fly_ash       water coarse_aggr   fine_aggr
## 76.90231240  0.08372863  0.06989374 -0.17995726 -0.03064797 -0.01084040
##
## R-Squared: 0.907
## Root Mean Squared Error: 2.549
```

```
tidy(slump_lr_p5_fit)
```

```
## # A tibble: 6 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  76.9     12.4          6.22 3.40e- 8
## 2 cement         0.0837   0.00463     18.1  0.
## 3 fly_ash        0.0699   0.00451     15.5  0.
## 4 water         -0.180    0.0211      -8.55 1.99e-12
## 5 coarse_aggr   -0.0306   0.00569     -5.38 9.44e- 7
## 6 fine_aggr     -0.0108   0.00641     -1.69 9.52e- 2
```

The $R^2 = 0.907$ is still very high.

We next remove `fine_aggr` since its statistic is the smallest.

```
slump_lr_p4_fit <- slump_partition$training %>%
  ml_linear_regression(compressive_strength ~ cement + fly_ash + water + coarse_aggr)
summary(slump_lr_p4_fit)
```

```
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -5.4384 -1.8189 -0.0654  1.3288  6.4798
##
## Coefficients:
## (Intercept)      cement     fly_ash       water coarse_aggr
## 60.21708437  0.08587463  0.07308202 -0.16677057 -0.02490884
##
## R-Squared: 0.9032
## Root Mean Squared Error: 2.602
```

```
tidy(slump_lr_p4_fit)
```

```
## # A tibble: 5 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  60.2      7.57         7.96 2.19e-11
## 2 cement        0.0859   0.00451     19.0  0.
## 3 fly_ash       0.0731   0.00415     17.6  0.
## 4 water        -0.167    0.0198      -8.42 3.11e-12
## 5 coarse_aggr  -0.0249   0.00463     -5.38 9.35e- 7
```

The $R^2 = 0.903$ barely drops.

We next remove `coarse_aggr` for completeness.

```
slump_lr_p3_fit <- slump_partition$training %>%
  ml_linear_regression(compressive_strength ~  cement + fly_ash + water)
summary(slump_lr_p3_fit)
```

```
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -7.1067 -1.9986 -0.2108  2.0982  7.7070
##
## Coefficients:
## (Intercept)      cement     fly_ash       water
## 24.21081578  0.09287406  0.07469413 -0.10575013
##
## R-Squared: 0.8631
## Root Mean Squared Error: 3.093
```

```
tidy(slump_lr_p3_fit)
```

```
## # A tibble: 4 x 5
##   term        estimate std.error statistic     p.value
##   <chr>          <dbl>     <dbl>     <dbl>       <dbl>
## 1 (Intercept)  24.2      4.17         5.81 0.000000165
## 2 cement        0.0929   0.00510     18.2  0
## 3 fly_ash       0.0747   0.00489     15.3  0
## 4 water        -0.106    0.0192      -5.51 0.000000534
```

The $R^2 = 0.863$ now drops about 5%.

Now `water` has the largest p-value.

```
slump_lr_p2_fit <- slump_partition$training %>%
  ml_linear_regression(compressive_strength ~  cement + fly_ash)
summary(slump_lr_p2_fit)
```

```
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -8.12162 -2.12346 -0.09783  1.50662 11.61679
##
## Coefficients:
## (Intercept)      cement     fly_ash
##  3.10846980  0.09134122  0.07929089
##
## R-Squared: 0.8045
## Root Mean Squared Error: 3.697
```

```
tidy(slump_lr_p2_fit)
```

```
## # A tibble: 3 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)   3.11     1.96         1.58   0.118
## 2 cement        0.0913   0.00605     15.1    0
## 3 fly_ash       0.0793   0.00572     13.9    0
```

Judging the efficacy of models based on the R^2 and RMSE for the training data is not what we should do. We need to compute performance metrics, for regression the Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE), on the test data. Both of these measures are on the same scale as `compressive_strength`.

We will compute these metrics for the test data on all models and then plot them. This will provide information for selecting the "best" model. Best is in quotes because we have not looked at all possible models

First, we form a named list of the models and compute a list of Spark DataFrames containing `compressive_strength` and its prediction for each model (the components of the list).

```
# form a list of the fitted models above
slump_lr_models <- list(
  "lr_p1" = slump_lr_p1_fit,
  "lr_p2" = slump_lr_p2_fit,
  "lr_p3" = slump_lr_p3_fit,
  "lr_p4" = slump_lr_p4_fit,
  "lr_p5" = slump_lr_p5_fit,
  "lr_p6" = slump_lr_p6_fit,
  "lr_full" = slump_lr_full_fit
)
# the scoring function
slump_test_fnc <- function(model, data = slump_partition$test){
  ml_predict(model, data) %>%
  select(compressive_strength, prediction)
}
# compute predicted values
slump_test_scores <- lapply(slump_lr_models, slump_test_fnc)
# slump_test_scores
```

The name of the predicted `compressive_strength` is `prediction`.

We now define a function that computes `rmse` and `mae` on a Spark DataFrame.

```
calculate_errors <- function(data_scores) {
  data_scores %>%
    mutate(pred_diff2 = (compressive_strength - prediction)^2) %>%
    mutate(pred_abs = abs(compressive_strength - prediction)) %>%
    summarize(rmse = sqrt(mean(pred_diff2)), mae = mean(pred_abs)) %>%
    collect()
}
```

This is utility code for computing metrics for the null model, i.e., the model with only the intercept (the base model for comparison).

```
slump_test_df <- slump_partition$test %>%
collect()
y <- slump_test_df$compressive_strength
```

We initialize the summary `data.frame` for the metrics with the null model.

```
terms <- c(1, 2, 3, 4, 5, 6, 7)
# slump_lr_errors <- data.frame(rmse = sqrt(mean((y -mean(y))^2)),
#                                mae = mean(abs(y - mean(y))), model = "lr_null")
slump_lr_errors <- NULL
```

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

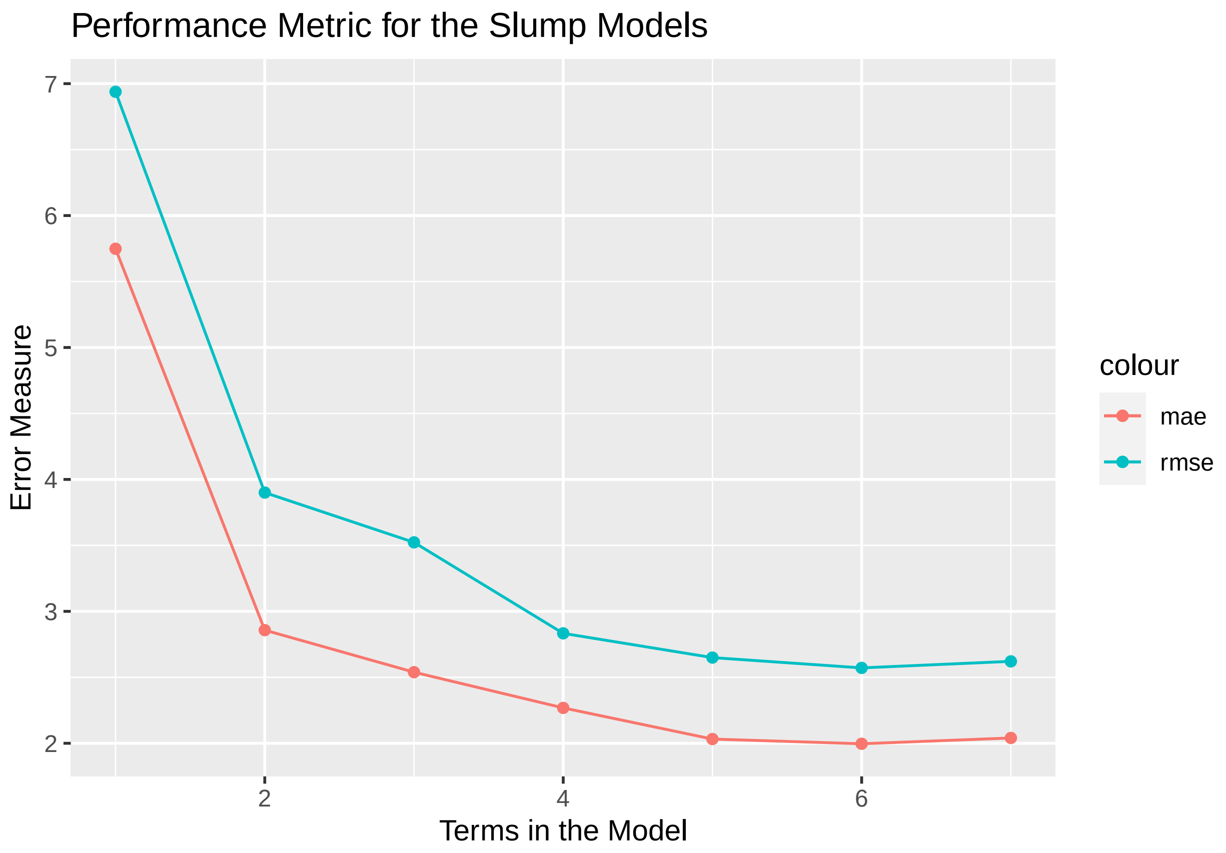
```
for(name in names(slump_test_scores)) {
  slump_lr_errors <- slump_test_scores[[name]] %>%
    calculate_errors %>%
    mutate(model = name) %>%
    rbind(slump_lr_errors, .)

}
cbind(terms, slump_lr_errors)
```

```
##   terms     rmse      mae   model
## 1     1 6.938355 5.748349   lr_p1
## 2     2 3.900348 2.857662   lr_p2
## 3     3 3.523081 2.538893   lr_p3
## 4     4 2.833356 2.268308   lr_p4
## 5     5 2.649555 2.032049   lr_p5
## 6     6 2.571492 1.996353   lr_p6
## 7     7 2.620981 2.040364 lr_full
```

The output is informative, but a plot is better.

```
library(ggplot2)
cbind(terms, slump_lr_errors) %>%
  ggplot(aes(x = terms)) +
  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 Models") +
  xlab("Terms in the Model") + ylab("Error Measure")
```



We want a parsimonious model so it is clear that the 3-term model or the 4-term model should be chosen. The variables in the final model are `cement`, `fly_ash`, and `water`. Arguably, `coarse_aggr` and `fine_aggr` could also be in the model.

The above approach is not guaranteed to be optimal since only a subset of the possible models are examined. Further, Spark depends on regularization for feature selection and does not support automatic variable selection based on optimality criteria. This example will be redone using regularization in Section 3 of this module.

```
spark_disconnect(sc)
```