Building Pipelines

Jim Harner

1/12/2021

Load the `dplyr` and `sparklyr` libraries and establish the Spark connection.

```r
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(sparklyr)
# start the sparklyr session locally or to the master container
if(system("test \"/bin/spark-class/\" && echo 1 || echo 0") == 1){
  master <- "spark://master:7077"
} else{
  master <- "local"
}
sc <- spark_connect(master = master)

# path.data <- paste("hdfs://hadoop:9000/user/", Sys.getenv("USER"), "/data/slump.csv", sep = "")
# For data knitted on the local filesystem:
path.data <- paste0("file:///home/", Sys.getenv("USER"), "/rspark-book/data/slump.csv")
path.data
```

```
## [1] "file:///home/rstudio/rspark-book/data/slump.csv"
```

6.9 Creating Pipelines

So far we have been building machine learning (ML) workflows using an interface to Spark based on `dplyr` verbs and the R pipe operator. In order to productionize machine learning code, we need to formalize the steps comprising ML Pipelines, i.e., use the Spark API.

8.1 ML Pipelines

An ML pipeline consists of a sequence of transformers and estimators, each forming a pipeline stage, which define the ML workflow. The building blocks of the *pipeline* consists of:

- *transformers*: algorithms for converting an input DataFrame to an output DataFrame;
- *estimators*: algorithms for fitting a model to an input DataFrame to produce a transformer.

Transformers are implemented by a `transform` function and are of two types:

- feature transformers convert the input DataFrame by altering the features or by creating new featues in the output DataFrame;

- learned models convert the input DataFrane containng features to another DataFrame appending the predicted values (or labels) based on the fitted model.

Estimators are learning algorithms that train (or fit) data by implementing a `fit` function. It inputs a DataFrame and produces a Model, which is a Transformer.

A pipeline model is a pipeline that has been fitted to the data with all estimators being converted to transformers.

6.9.1 Concrete Slump Pipeline

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

```
slump_sdf <- spark_read_csv(sc, "slump_sdf", path = path.data)
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
```

Machine learning algorithms and feature transformers generally require the input to be a vector. The training input variables are combined into a feature vector and then passed to `ft_vector_assembler` as the `input_col`. The `output_col` is a list assembled by rows, i.e., observations. The training assembled features are then standardized to have mean 0 and standard deviation 1 using `ft_standard_scaler`.

```
features <- c("cement", "slag", "fly_ash", "water", "sp", "coarse_aggr", "fine_aggr")

slump_train_sdf %>%
  ft_vector_assembler(input_col = features,
                      output_col = "features_assembled") %>%
  ft_standard_scaler(input_col = "features_assembled",
                     output_col = "features_scaled",
                     with_mean = TRUE) %>%
  glimpse()
```

```
## Rows: ??
## Columns: 12
## Database: spark_connection
## $ cement               <dbl> 137.0, 140.0, 140.0, 140.0, 140.1, 140.2, 140....
```

```
## $ slag                   <dbl> 167.0, 1.4, 128.0, 128.0, 11.8, 30.5, 44.8, 0....
## $ fly_ash                <dbl> 214.0, 198.1, 164.0, 164.0, 226.1, 239.0, 234....
## $ water                  <dbl> 226.0, 174.9, 183.0, 237.0, 207.8, 169.4, 171....
## $ sp                     <dbl> 6.0, 4.4, 12.0, 6.0, 4.9, 5.3, 5.5, 4.6, 11.0,...
## $ coarse_aggr            <dbl> 708.0, 1049.9, 871.0, 869.0, 1020.9, 1028.4, 1...
## $ fine_aggr              <dbl> 757.0, 780.5, 775.0, 656.0, 683.8, 742.7, 704....
## $ slump                  <dbl> 27.50, 16.25, 23.75, 24.00, 21.00, 21.25, 23.5...
## $ flow                   <dbl> 70.0, 31.0, 53.0, 65.0, 64.0, 46.0, 52.5, 53.0...
## $ compressive_strength <dbl> 34.45, 30.83, 33.38, 29.50, 26.28, 36.32, 33.7...
## $ features_assembled   <list> [<137, 167, 214, 226, 6, 708, 757>, <140.0, 1...
## $ features_scaled      <list> [<-1.1666834, 1.4965032, 0.7718295, 1.4980383...
```

We next define a pipeline based on the two stages above, i.e., a `vector_assembler` (a Transformer), a `standard_scaler` (an Estimator), and a third stage: `linear_regression` (an Estimator). The `pipeline` itself is an Estimator.

```
slump_pipeline <- ml_pipeline(sc) %>%
  ft_vector_assembler(input_col = features,
                      output_col = "features_assembled") %>%
  ft_standard_scaler(input_col = "features_assembled",
                     output_col = "features_scaled",
                     with_mean = TRUE) %>%
  ml_linear_regression(features_col = "features_scaled",
               label_col = "compressive_strength")
class(slump_pipeline)
```

```
## [1] "ml_pipeline"       "ml_estimator"      "ml_pipeline_stage"
```

```
slump_pipeline
```

```
## Pipeline (Estimator) with 3 stages
## <pipeline__f9ea1d1c_99b6_49c2_806b_e64c6c89fb1a>
##   Stages
##   |--1 VectorAssembler (Transformer)
##   |    <vector_assembler__344ddb98_841b_41c3_8523_89a853102b94>
##   |     (Parameters -- Column Names)
##   |      input_cols: cement, slag, fly_ash, water, sp, coarse_aggr, fine_aggr
##   |      output_col: features_assembled
##   |--2 StandardScaler (Estimator)
##   |    <standard_scaler__af257ec1_fa32_4f05_837b_48abeaa1087c>
##   |     (Parameters -- Column Names)
##   |      input_col: features_assembled
##   |      output_col: features_scaled
##   |     (Parameters)
##   |      with_mean: TRUE
##   |      with_std: TRUE
##   |--3 LinearRegression (Estimator)
##   |    <linear_regression__e4f35928_0706_4fd5_8da0_732487072bfb>
##   |     (Parameters -- Column Names)
##   |      features_col: features_scaled
##   |      label_col: compressive_strength
##   |      prediction_col: prediction
##   |     (Parameters)
##   |      aggregation_depth: 2
##   |      elastic_net_param: 0
##   |      epsilon: 1.35
```

```
##   |      fit_intercept: TRUE
##   |      loss: squaredError
##   |      max_iter: 100
##   |      reg_param: 0
##   |      solver: auto
##   |      standardization: TRUE
##   |      tol: 1e-06
```

The pipeline can then be fit (trained) on the training data: `slump_train_sdf`.

```
slump_full_model <- slump_pipeline %>%
  ml_fit(slump_train_sdf)
class(slump_full_model)
```

```
## [1] "ml_pipeline_model" "ml_transformer"    "ml_pipeline_stage"
```

```
slump_full_model
```

```
## PipelineModel (Transformer) with 3 stages
## <pipeline__f9ea1d1c_99b6_49c2_806b_e64c6c89fb1a>
##   Stages
##   |--1 VectorAssembler (Transformer)
##   |    <vector_assembler__344ddb98_841b_41c3_8523_89a853102b94>
##   |     (Parameters -- Column Names)
##   |      input_cols: cement, slag, fly_ash, water, sp, coarse_aggr, fine_aggr
##   |      output_col: features_assembled
##   |--2 StandardScalerModel (Transformer)
##   |    <standard_scaler__af257ec1_fa32_4f05_837b_48abeaa1087c>
##   |     (Parameters -- Column Names)
##   |      input_col: features_assembled
##   |      output_col: features_scaled
##   |     (Transformer Info)
##   |      mean:  num [1:7] 230.3 75.4 148.7 196.4 8.2 ...
##   |      std:  num [1:7] 79.99 61.22 84.6 19.74 2.25 ...
##   |--3 LinearRegressionModel (Transformer)
##   |    <linear_regression__e4f35928_0706_4fd5_8da0_732487072bfb>
##   |     (Parameters -- Column Names)
##   |      features_col: features_scaled
##   |      label_col: compressive_strength
##   |      prediction_col: prediction
##   |     (Transformer Info)
##   |      coefficients:  num [1:7] 5.66 -1.123 4.814 -4.338 -0.105 ...
##   |      intercept:  num 35.9
##   |      num_features:  int 7
##   |      scale:  num 1
```

Now get the fitted values on the test data.

```
(slump_fitted_full_test <- ml_transform(slump_full_model, slump_test_sdf))
```

```
## # Source: spark<?> [?? x 13]
##    cement  slag fly_ash water    sp coarse_aggr fine_aggr slump  flow
##     <dbl> <dbl>   <dbl> <dbl> <dbl>       <dbl>     <dbl> <dbl> <dbl>
##  1   140.   4.2    216.  194.   4.7       1050.      710.  24.5  57
##  2   140.  61.1    239.  182.   5.7       1018.      681.  24.5  60
##  3   145    0      227   240    6          750       853   14.5  58.5
##  4   145  177      227   209   11          752       715    2.5  20
```

```
## 5    148  109      139   193    7          768       902   23.8  58
## 6    152  139      178   168   18          944       695    0    20
## 7    154  141      181   234   11          797       683   23    65
## 8    158    0      246   174    7         1035       706   19    43
## 9    159    0      187   176   11          990       789   12    39
## 10   159  116      149   175   15          953       720   23.5  54.5
## # ... with more rows, and 4 more variables: compressive_strength <dbl>,
## #   features_assembled <list>, features_scaled <list>, prediction <dbl>
```

```
class(slump_fitted_full_test)
```

```
## [1] "tbl_spark" "tbl_sql"   "tbl_lazy"  "tbl"
```

```
slump_fitted_full_test %>%
  summarize(mean(abs(compressive_strength - prediction)))
```

```
## Warning: Missing values are always removed in SQL.
## Use `mean(x, na.rm = TRUE)` to silence this warning
## This warning is displayed only once per session.
```

```
## # Source: spark<?> [?? x 1]
##   `mean(abs(compressive_strength - prediction))`
##                                            <dbl>
## 1                                           2.04
```

6.9.2 Hyperparameter Tuning

Model selection is a critical, but difficult task in the effort to find the "best model." Rather than doing variable selection using optimal statistical criteria, Spark uses regularization. The process of selecting a model is done by hyperparameter tuning, or for short "tuning." In addition to tuning the regression regularization parameter, λ, and the elastic net parameter, α, other hyperparameters, e.g., whether or not to substract the mean when standardizing, can be included.

Two tuning approaches are available for tuning in Spark:

- train-validation split using `ml_train_validation_split`;
- k-fold cross-validation using `ml_cross_validator`.

In both cases you need three arguments for tuning:

- `estimator`: the algorithm or pipeline to tune;
- `estimator_param_map`: the parameter grid to search over;
- `evaluator`: the metric that measure how well the fitted model does on held-out data.

Model selection is done by:

- splitting the input data into separate training and test datasets;
- iterating through the parameter grid for each training-test pair;
  - using `estimator` for getting the fitted model;
  - evaluating the fitted model using the `evaluator`;
- selecting the best performing model.

We consider each in turn for the `slump` data.

Train-validation split The `ml_train_validation_split` function evaluates the training-test pair once for each element of the parameter grid. The data is split into training and test datasets by the `train_ratio` argument.

```
slump_tv <- ml_train_validation_split(sc,
  estimator = slump_pipeline,
  estimator_param_map = list(
    linear_regression = list(
      reg_param = c(0.0, 0.0025, 0.005, 0.0075, 0.01, 0.02, 0.04, 0.06, 0.1, 0.15),
      elastic_net_param = c(0.0, 1.0)
    )
  ),
  evaluator = ml_regression_evaluator(sc, label_col = "compressive_strength",
                                      metric_name = "mae"),
  train_ratio = 0.7,
  parallelism = 3,
  seed = 2)
class(slump_tv)
```

```
## [1] "ml_train_validation_split" "ml_tuning"
## [3] "ml_estimator"              "ml_pipeline_stage"
```

```
slump_tv
```

```
## TrainValidationSplit (Estimator)
## <train_validation_split__28549b4f_8d63_4a12_9041_847366ef40c9>
##  (Parameters -- Tuning)
##   estimator: Pipeline
##              <pipeline__f9ea1d1c_99b6_49c2_806b_e64c6c89fb1a>
##   evaluator: RegressionEvaluator
##              <regression_evaluator__13674130_8cb2_4298_96c3_e89dfaf56db9>
##     with metric mae
##   train_ratio: 0.7
##   [Tuned over 20 hyperparameter sets]
```

Fit the models over the parameter grid and choose the best model. The best model is displayed as part of the output, but it can be found directly from: `slump_tv_model$best_model`.

```
slump_tv_model <- ml_fit(slump_tv, slump_train_sdf)
class(slump_tv_model)
```

```
## [1] "ml_train_validation_split_model" "ml_tuning_model"
## [3] "ml_transformer"                  "ml_pipeline_stage"
```

```
slump_tv_model # slump_tv_model$best_model
```

```
## TrainValidationSplitModel (Transformer)
## <train_validation_split__28549b4f_8d63_4a12_9041_847366ef40c9>
##  (Parameters -- Tuning)
##   estimator: Pipeline
##              <pipeline__f9ea1d1c_99b6_49c2_806b_e64c6c89fb1a>
##   evaluator: RegressionEvaluator
##              <regression_evaluator__13674130_8cb2_4298_96c3_e89dfaf56db9>
##     with metric mae
##   train_ratio: 0.7
##   [Tuned over 20 hyperparameter sets]
##  (Best Model)
```

```
##   PipelineModel (Transformer) with 3 stages
##   <pipeline__f9ea1d1c_99b6_49c2_806b_e64c6c89fb1a>
##     Stages
##     |--1 VectorAssembler (Transformer)
##     |    <vector_assembler__344ddb98_841b_41c3_8523_89a853102b94>
##     |     (Parameters -- Column Names)
##     |      input_cols: cement, slag, fly_ash, water, sp, coarse_aggr, fine_aggr
##     |      output_col: features_assembled
##     |--2 StandardScalerModel (Transformer)
##     |    <standard_scaler__af257ec1_fa32_4f05_837b_48abeaa1087c>
##     |     (Parameters -- Column Names)
##     |      input_col: features_assembled
##     |      output_col: features_scaled
##     |     (Transformer Info)
##     |      mean:  num [1:7] 230.3 75.4 148.7 196.4 8.2 ...
##     |      std:  num [1:7] 79.99 61.22 84.6 19.74 2.25 ...
##     |--3 LinearRegressionModel (Transformer)
##     |    <linear_regression__e4f35928_0706_4fd5_8da0_732487072bfb>
##     |     (Parameters -- Column Names)
##     |      features_col: features_scaled
##     |      label_col: compressive_strength
##     |      prediction_col: prediction
##     |     (Transformer Info)
##     |      coefficients:  num [1:7] 6.6 0 5.85 -3.1 0 ...
##     |      intercept:  num 35.9
##     |      num_features:  int 7
##     |      scale:  num 1
```

Inspect the evaluation metric over the parameter grid.

```r
ml_validation_metrics(slump_tv_model) %>%
  arrange(elastic_net_param_1, reg_param_1)
```

```
##         mae elastic_net_param_1 reg_param_1
## 1  2.853767                   0      0.0000
## 2  2.838023                   0      0.0025
## 3  2.824923                   0      0.0050
## 4  2.813762                   0      0.0075
## 5  2.804067                   0      0.0100
## 6  2.774540                   0      0.0200
## 7  2.736175                   0      0.0400
## 8  2.708406                   0      0.0600
## 9  2.663937                   0      0.1000
## 10 2.616364                   0      0.1500
## 11 2.853767                   1      0.0000
## 12 2.788731                   1      0.0025
## 13 2.774220                   1      0.0050
## 14 2.758700                   1      0.0075
## 15 2.750158                   1      0.0100
## 16 2.701110                   1      0.0200
## 17 2.633763                   1      0.0400
## 18 2.596387                   1      0.0600
## 19 2.542572                   1      0.1000
## 20 2.482346                   1      0.1500
```

Make predictions on the test data uaing the “best model.”

```
(slump_fitted_tv_test <- ml_transform(slump_tv_model, slump_test_sdf))
```

```
## # Source: spark<?> [?? x 13]
##    cement  slag fly_ash water    sp coarse_aggr fine_aggr slump  flow
##     <dbl> <dbl>   <dbl> <dbl> <dbl>       <dbl>     <dbl> <dbl> <dbl>
##  1   140.   4.2    216.  194.   4.7       1050.      710.  24.5  57
##  2   140.  61.1    239.  182.   5.7       1018.      681.  24.5  60
##  3   145    0      227   240    6          750       853   14.5  58.5
##  4   145  177      227   209   11          752       715    2.5  20
##  5   148  109      139   193    7          768       902   23.8  58
##  6   152  139      178   168   18          944       695    0    20
##  7   154  141      181   234   11          797       683   23    65
##  8   158    0      246   174    7         1035       706   19    43
##  9   159    0      187   176   11          990       789   12    39
## 10   159  116      149   175   15          953       720   23.5  54.5
## # ... with more rows, and 4 more variables: compressive_strength <dbl>,
## #   features_assembled <list>, features_scaled <list>, prediction <dbl>
```

```
class(slump_fitted_tv_test)
```

```
## [1] "tbl_spark" "tbl_sql"   "tbl_lazy"  "tbl"
```

```
slump_fitted_tv_test %>%
  summarize(mean(abs(compressive_strength - prediction)))
```

```
## # Source: spark<?> [?? x 1]
##   `mean(abs(compressive_strength - prediction))`
##                                              <dbl>
## 1                                             2.12
```

Cross-validation Cross-validation splits the dataset into k folds, which provides the training and test datasets. This is done by `ml_cross_validator`. The evaluation metric is computed by averaging its value over the k models produced by the `estimator` over the k training-test pairs. Cross-validation is more compute intensive than the train-validation split, but potentially is more reliable.

```
slump_cv <- ml_cross_validator(sc,
  estimator = slump_pipeline,
  estimator_param_map = list(
    linear_regression = list(
      reg_param = c(0.0, 0.0025, 0.005, 0.0075, 0.01, 0.02, 0.04, 0.06, 0.1, 0.15),
      elastic_net_param = c(0.0, 1.0)
    )
  ),
  evaluator = ml_regression_evaluator(sc, label_col = "compressive_strength",
                                      metric_name = "mae"),
  num_folds = 10,
  parallelism = 3)

slump_cv
```

```
## CrossValidator (Estimator)
## <cross_validator__ae3b2771_cd42_4d91_aa1a_be0ec85130c9>
##  (Parameters -- Tuning)
##   estimator: Pipeline
##              <pipeline__f9ea1d1c_99b6_49c2_806b_e64c6c89fb1a>
##   evaluator: RegressionEvaluator
```

```
##              <regression_evaluator__82ebd731_32b1_42ab_b606_f94d352a1f2b>
##     with metric mae
##   num_folds: 10
##   [Tuned over 20 hyperparameter sets]
```

Make predictions on the train data using the “best model.”

```r
slump_cv_model <- ml_fit(slump_cv, slump_train_sdf)
slump_cv_model
```

```
## CrossValidatorModel (Transformer)
## <cross_validator__ae3b2771_cd42_4d91_aa1a_be0ec85130c9>
##  (Parameters -- Tuning)
##   estimator: Pipeline
##              <pipeline__f9ea1d1c_99b6_49c2_806b_e64c6c89fb1a>
##   evaluator: RegressionEvaluator
##              <regression_evaluator__82ebd731_32b1_42ab_b606_f94d352a1f2b>
##     with metric mae
##   num_folds: 10
##   [Tuned over 20 hyperparameter sets]
##  (Best Model)
##   PipelineModel (Transformer) with 3 stages
##   <pipeline__f9ea1d1c_99b6_49c2_806b_e64c6c89fb1a>
##     Stages
##     |--1 VectorAssembler (Transformer)
##     |    <vector_assembler__344ddb98_841b_41c3_8523_89a853102b94>
##     |     (Parameters -- Column Names)
##     |      input_cols: cement, slag, fly_ash, water, sp, coarse_aggr, fine_aggr
##     |      output_col: features_assembled
##     |--2 StandardScalerModel (Transformer)
##     |    <standard_scaler__af257ec1_fa32_4f05_837b_48abeaa1087c>
##     |     (Parameters -- Column Names)
##     |      input_col: features_assembled
##     |      output_col: features_scaled
##     |     (Transformer Info)
##     |      mean:  num [1:7] 230.3 75.4 148.7 196.4 8.2 ...
##     |      std:  num [1:7] 79.99 61.22 84.6 19.74 2.25 ...
##     |--3 LinearRegressionModel (Transformer)
##     |    <linear_regression__e4f35928_0706_4fd5_8da0_732487072bfb>
##     |     (Parameters -- Column Names)
##     |      features_col: features_scaled
##     |      label_col: compressive_strength
##     |      prediction_col: prediction
##     |     (Transformer Info)
##     |      coefficients:  num [1:7] 6.67 0 5.9 -3.43 0 ...
##     |      intercept:  num 35.9
##     |      num_features:  int 7
##     |      scale:  num 1
```

Inspect the evaluation metric over the parameter grid.

```r
ml_validation_metrics(slump_cv_model) %>%
  arrange(elastic_net_param_1, reg_param_1)
```

```
##         mae elastic_net_param_1 reg_param_1
## 1  2.091564                   0      0.0000
```

```
## 2  2.088695                   0      0.0025
## 3  2.086237                   0      0.0050
## 4  2.084099                   0      0.0075
## 5  2.082218                   0      0.0100
## 6  2.076440                   0      0.0200
## 7  2.069205                   0      0.0400
## 8  2.064432                   0      0.0600
## 9  2.057642                   0      0.1000
## 10 2.061992                   0      0.1500
## 11 2.091564                   1      0.0000
## 12 2.080900                   1      0.0025
## 13 2.076835                   1      0.0050
## 14 2.072895                   1      0.0075
## 15 2.068517                   1      0.0100
## 16 2.055233                   1      0.0200
## 17 2.049943                   1      0.0400
## 18 2.052551                   1      0.0600
## 19 2.074921                   1      0.1000
## 20 2.114929                   1      0.1500
```

Make predictions on the test data uaing the "best model."

```
(slump_fitted_cv_test <- ml_transform(slump_cv_model, slump_test_sdf))
```

```
## # Source: spark<?> [?? x 13]
##    cement  slag fly_ash water    sp coarse_aggr fine_aggr slump  flow
##     <dbl> <dbl>   <dbl> <dbl> <dbl>       <dbl>     <dbl> <dbl> <dbl>
##  1   140.   4.2    216.  194.   4.7       1050.      710.  24.5  57
##  2   140.  61.1    239.  182.   5.7       1018.      681.  24.5  60
##  3   145     0     227   240    6          750       853   14.5  58.5
##  4   145   177     227   209   11          752       715    2.5  20
##  5   148   109     139   193    7          768       902   23.8  58
##  6   152   139     178   168   18          944       695    0    20
##  7   154   141     181   234   11          797       683   23    65
##  8   158     0     246   174    7         1035       706   19    43
##  9   159     0     187   176   11          990       789   12    39
## 10   159   116     149   175   15          953       720   23.5  54.5
## # ... with more rows, and 4 more variables: compressive_strength <dbl>,
## #   features_assembled <list>, features_scaled <list>, prediction <dbl>
```

```
slump_fitted_cv_test %>%
  summarize(mean(abs(compressive_strength - prediction)))
```

```
## # Source: spark<?> [?? x 1]
##   `mean(abs(compressive_strength - prediction))`
##                                            <dbl>
## 1                                           2.05
```

Pipelines can be serialized to disk and be accessed by other Spark APIs such as Python.

```
ml_save(slump_cv_model$best_model, path = getwd(), overwrite = TRUE)
```

```
spark_disconnect(sc)
```