Evaluation of Decision Tree in Spark

By the end of this activity, you will be able to perform the following in Spark:

1. Determine the accuracy of a classifier model
2. Display the confusion matrix for a classifier model

In this activity, you will be programming in a Jupyter Python Notebook. If you have not already started the Jupyter Notebook server, see the instructions in the Reading *Instructions for Starting Jupyter*.

Step 1. **Open Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:

Navigate to *localhost:8889/tree/Downloads/big-data-4*:

Open the model evaluation notebook by clicking on *model-evaluation.ipynb:*

Step 2. **Load predictions.** Execute the first cell to load the classes used in this activity:

```
In [1]: from pyspark.sql import SQLContext
        from pyspark.ml.evaluation import MulticlassClassificationEvaluator
        from pyspark.mllib.evaluation import MulticlassMetrics
```

Execute the next cell to load the predictions CSV file that we created at the end of the Week 3 Hands-On *Classification in Spark* into a DataFrame:

```
In [2]: sqlContext = SQLContext(sc)
        predictions = sqlContext.read.load('file:///home/cloudera/Downloads/big-data-4/predictions.csv',
                                    format='com.databricks.spark.csv',
                                    header='true',inferSchema='true')
```

Step 3. **Compute accuracy.** Let's create an instance of *MulticlassClassificationEvaluator* to determine the accuracy of the predictions:

```
In [3]: evaluator = MulticlassClassificationEvaluator(
            labelCol="label", predictionCol="prediction", metricName="precision")
```

The first two arguments specify the names of the label and prediction columns, and the third argument specifies that we want the overall precision.

We can compute the accuracy by calling *evaluate()*:

```
In [4]: accuracy = evaluator.evaluate(predictions)
        print("Accuracy = %g " % (accuracy))

        Accuracy = 0.809524
```

Step 4. **Display confusion matrix.** The *MulticlassMetrics* class can be used to generate a confusion matrix of our classifier model. However, unlike *MulticlassClassificationEvaluator*, *MulticlassMetrics* works with RDDs of numbers and not DataFrames, so we need to convert our *predictions* DataFrame into an RDD.

If we use the *rdd* attribute of *predictions,* we see this is an *RDD* of *Rows:*

```
In [5]: predictions.rdd.take(2)
Out[5]: [Row(prediction=1.0, label=1.0), Row(prediction=1.0, label=1.0)]
```

Instead, we can map the RDD to *tuple* to get an RDD of numbers:

```
In [6]: predictions.rdd.map(tuple).take(2)
Out[6]: [(1.0, 1.0), (1.0, 1.0)]
```

Let's create an instance of *MulticlassMetrics* with this RDD:

```
In [7]: metrics = MulticlassMetrics(predictions.rdd.map(tuple))
```

NOTE: the above command can take longer to execute than most Spark commands when first run in the notebook.

The *confusionMatrix()* function returns a Spark *Matrix*, which we can convert to a Python Numpy array, and transpose to view:

```
In [8]: metrics.confusionMatrix().toArray().transpose()
Out[8]: array([[ 87.,  26.],
               [ 14.,  83.]])
```