Evaluation

Lesson Objectives

- After completing this lesson, you should be able to:
  - Evaluate binary classification algorithms using area under the ROC curve
  - Evaluate multiclass classification algorithms using several metrics
  - Evaluate regression algorithms using several metrics

Evaluators

- Computes metrics from predictions
- Available Evaluators
  - BinaryClassificationEvaluator
  - MultiClassClassificationEvaluator
  - RegressionEvaluator

BinaryClassificationMetrics

- Evaluator for binary classification

Expects two input columns: predictions and labels

- Supported metric: areaUnderROC

BinaryClassificationMetrics

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.util import MLUtils

!wget https://raw.githubusercontent.com/apache/spark/master/data/mllib/sample_binary_classification_data.txt
```

```
--2016-09-26 09:12:21--  https://raw.githubusercontent.com/apache/spark/master/data/mllib/sample_binary_classification_da
ta.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.12.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.12.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 104736 (102K) [text/plain]
Saving to: ‘sample_binary_classification_data.txt.2’

100%[======================================>] 104.736     --.-K/s   in 0,09s

2016-09-26 09:12:21 (1,09 MB/s) - ‘sample_binary_classification_data.txt.2’ saved [104736/104736]
```

BinaryClassificationMetrics

```
data = MLUtils.loadLibSVMFile(sc, "sample_binary_classification_data.txt")

training, test = data.randomSplit([0.6, 0.4], seed=11L)
```

```
data.take(1)
```

```
[LabeledPoint(0.0, (692,[127,128,129,130,131,154,155,156,157,158,159,181,182,183,184,185,186,187,188,189,207,208,209,210,
211,212,213,214,215,216,217,235,236,237,238,239,240,241,242,243,244,245,262,263,264,265,266,267,268,269,270,271,272,273,2
89,290,291,292,293,294,295,296,297,300,301,302,316,317,318,319,320,321,328,329,330,343,344,345,346,347,348,349,356,357,35
8,371,372,373,374,384,385,386,399,400,401,412,413,414,426,427,428,429,440,441,442,454,455,456,457,466,467,468,469,470,482
,483,484,493,494,495,496,497,510,511,512,520,521,522,523,538,539,540,547,548,549,550,566,567,568,569,570,571,572,573,574,
575,576,577,578,594,595,596,597,598,599,600,601,602,603,604,622,623,624,625,626,627,628,629,630,651,652,653,654,655,656,6
57],[51.0,159.0,253.0,159.0,50.0,48.0,238.0,252.0,252.0,252.0,237.0,54.0,227.0,253.0,252.0,239.0,233.0,252.0,57.0,6.0,10.
0,60.0,224.0,252.0,253.0,252.0,202.0,84.0,252.0,253.0,122.0,163.0,252.0,252.0,252.0,253.0,252.0,252.0,96.0,189.0,253.0,16
7.0,51.0,238.0,253.0,253.0,190.0,114.0,253.0,228.0,47.0,79.0,255.0,168.0,48.0,238.0,252.0,252.0,179.0,12.0,75.0,121.0,21.
0,253.0,243.0,50.0,38.0,165.0,253.0,233.0,208.0,84.0,253.0,252.0,165.0,7.0,178.0,252.0,240.0,71.0,19.0,28.0,253.0,252.0,1
95.0,57.0,252.0,252.0,63.0,253.0,252.0,195.0,198.0,253.0,190.0,255.0,253.0,196.0,76.0,246.0,252.0,112.0,253.0,252.0,148.0
,85.0,252.0,230.0,25.0,7.0,135.0,253.0,186.0,12.0,85.0,252.0,223.0,7.0,131.0,252.0,225.0,71.0,85.0,252.0,145.0,48.0,165.0
,252.0,173.0,86.0,253.0,225.0,114.0,238.0,253.0,162.0,85.0,252.0,249.0,146.0,48.0,29.0,85.0,178.0,225.0,253.0,223.0,167.0
,56.0,85.0,252.0,252.0,252.0,229.0,215.0,252.0,252.0,252.0,196.0,130.0,28.0,199.0,252.0,252.0,253.0,252.0,252.0,233.0,145
.0,25.0,128.0,252.0,253.0,252.0,141.0,37.0]))]
```

BinaryClassificationMetrics

```
model = LogisticRegressionWithLBFGS.train(training)

predictionAndLabels = test.map(lambda lp: (float(model.predict(lp.features)), lp.label))

metrics = BinaryClassificationMetrics(predictionAndLabels)

print("Area under PR = %s" % metrics.areaUnderPR)

print("Area under ROC = %s" % metrics.areaUnderROC)
```

```
Area under PR = 0.993386243386
Area under ROC = 0.981481481481
```

BinaryClassificationMetrics

```
model = LogisticRegressionWithLBFGS.train(training)

predictionAndLabels = test.map(lambda lp: (float(model.predict(lp.features)), lp.label))

metrics = BinaryClassificationMetrics(predictionAndLabels)

print("Area under PR = %s" % metrics.areaUnderPR)

print("Area under ROC = %s" % metrics.areaUnderROC)
```

```
Area under PR = 0.993386243386
Area under ROC = 0.981481481481
```

MulticlassMetrics

- valuator for multiclass classification
- Expects two input columns: prediction and label
- Supported metrics:
  - F1 (default)
  - Precision
  - Recall

MulticlassMetrics

```
from pyspark.mllib.evaluation import MulticlassMetrics

!wget https://raw.githubusercontent.com/apache/spark/master/data/mllib/sample_multiclass_classification_data.txt
```

```
--2016-09-26 09:14:48--  https://raw.githubusercontent.com/apache/spark/master/data/mllib/sample_multiclass_classification_data.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.12.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.12.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6953 (6,8K) [text/plain]
Saving to: ‘sample_multiclass_classification_data.txt’

100%[======================================>] 6.953       --.-K/s   in 0s

2016-09-26 09:14:48 (27,5 MB/s) - ‘sample_multiclass_classification_data.txt’ saved [6953/6953]
```

MulticlassMetrics

```
data = MLUtils.loadLibSVMFile(sc, "sample_multiclass_classification_data.txt")

training, test = data.randomSplit([0.6, 0.4], seed=11L)

data.take(1)
```

```
[LabeledPoint(1.0, (4,[0,1,2,3],[-0.222222,0.5,-0.762712,-0.833333]))]
```

```
model = LogisticRegressionWithLBFGS.train(training, numClasses=3)

predictionAndLabels = test.map(lambda lp: (float(model.predict(lp.features)), lp.label))

metrics = MulticlassMetrics(predictionAndLabels)
```

MulticlassMetrics

```
precision = metrics.precision()
recall = metrics.recall()
f1Score = metrics.fMeasure()
print("Summary Stats")
print("Precision = %s" % precision)
print("Recall = %s" % recall)
print("F1 Score = %s" % f1Score)
```

```
Summary Stats
Precision = 0.912280701754
Recall = 0.912280701754
F1 Score = 0.912280701754
```

MulticlassMetrics

```
labels = data.map(lambda lp: lp.label).distinct().collect()
for label in sorted(labels):
    print("Class %s precision = %s" % (label, metrics.precision(label)))
    print("Class %s recall = %s" % (label, metrics.recall(label)))
    print("Class %s F1 Measure = %s" % (label, metrics.fMeasure(label, beta=1.0)))
```

```
Class 0.0 precision = 0.913043478261
Class 0.0 recall = 0.875
Class 0.0 F1 Measure = 0.893617021277
Class 1.0 precision = 1.0
Class 1.0 recall = 1.0
Class 1.0 F1 Measure = 1.0
Class 2.0 precision = 0.8125
Class 2.0 recall = 0.866666666667
Class 2.0 F1 Measure = 0.838709677419
```

MulticlassMetrics

```
print("Weighted recall = %s" % metrics.weightedRecall)
print("Weighted precision = %s" % metrics.weightedPrecision)
print("Weighted F(1) Score = %s" % metrics.weightedFMeasure())
print("Weighted F(0.5) Score = %s" % metrics.weightedFMeasure(beta=0.5))
print("Weighted false positive rate = %s" % metrics.weightedFalsePositiveRate)
```

```
Weighted recall = 0.912280701754
Weighted precision = 0.914044622426
Weighted F(1) Score = 0.912762345122
Weighted F(0.5) Score = 0.913437018999
Weighted false positive rate = 0.044315333789
```

RegressionMetrics

- Evaluator for regression
- Expects two input columns: prediction and label
- Supported metrics:
  - rmse: root mean squared error (default)
  - mse: mean squared error
  - r2: R2, the coefficient of determination
  - mae: mean absolute error

A Simple Regression

```
from pyspark.mllib.regression import LinearRegressionWithSGD
from pyspark.mllib.evaluation import RegressionMetrics
from pyspark.mllib.linalg import DenseVector

!wget https://raw.githubusercontent.com/apache/spark/master/data/mllib/sample_linear_regression_data.txt
```

```
--2016-09-26 09:18:52--  https://raw.githubusercontent.com/apache/spark/master/data/mllib/sample_linear_regression_data.t
xt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.12.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.12.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 119069 (116K) [text/plain]
Saving to: ‘sample_linear_regression_data.txt.1’

100%[======================================>] 119.069     --.-K/s   in 0,09s

2016-09-26 09:18:53 (1,23 MB/s) - ‘sample_linear_regression_data.txt.1’ saved [119069/119069]
```

A Simple Regression

```
def parsePoint(line):
    values = line.split()
    return LabeledPoint(float(values[0]), DenseVector([float(x.split(':')[1]) for x in values[1:]]))

data = sc.textFile("sample_linear_regression_data.txt")
parsedData = data.map(parsePoint)

parsedData.take(1)
```

```
[LabeledPoint(-9.49000987882, [0.455127360066,0.36644694352,-0.382561089335,-0.445843019852,0.331097903589,0.806744529344
,-0.262434173177,-0.448503861117,-0.0726928483817,0.56580355758])]
```

```
model = LinearRegressionWithSGD.train(parsedData)

valuesAndPreds = parsedData.map(lambda p: (float(model.predict(p.features)), p.label))

metrics = RegressionMetrics(valuesAndPreds)
```

A Simple Regression

```
print("MSE = %s" % metrics.meanSquaredError)
print("RMSE = %s" % metrics.rootMeanSquaredError)

print("R-squared = %s" % metrics.r2)

print("MAE = %s" % metrics.meanAbsoluteError)

print("Explained variance = %s" % metrics.explainedVariance)
```

```
MSE = 103.309686818
RMSE = 10.1641372884
R-squared = 0.0276391109678
MAE = 8.14869190795
Explained variance = 2.88839520172
```

Lesson Summary

- Having completed this lesson, you should be able to:
  - Evaluate binary classification algorithms using area under the ROC curve
  - Evaluate multiclass classification algorithms using several metrics
  - Evaluate regression algorithms using several metrics