Decision trees

Lesson Objectives

- After completing this lesson, you should be able to:
  - Understand the Pipelines API for Decision Trees
  - Describe Pipeline's Input and Output columns
  - Perform classification and regression with Decision Trees
  - Understand and use Decision Trees' parameters

Decision trees

- Popular method for classification an regression
- Easy to interpret
- Handle categorical features
- Extend to multiclass classification
- Do NOT require feature scaling
- They capture non-linearities and feature interactions

SPark.ML API for Decision Trees

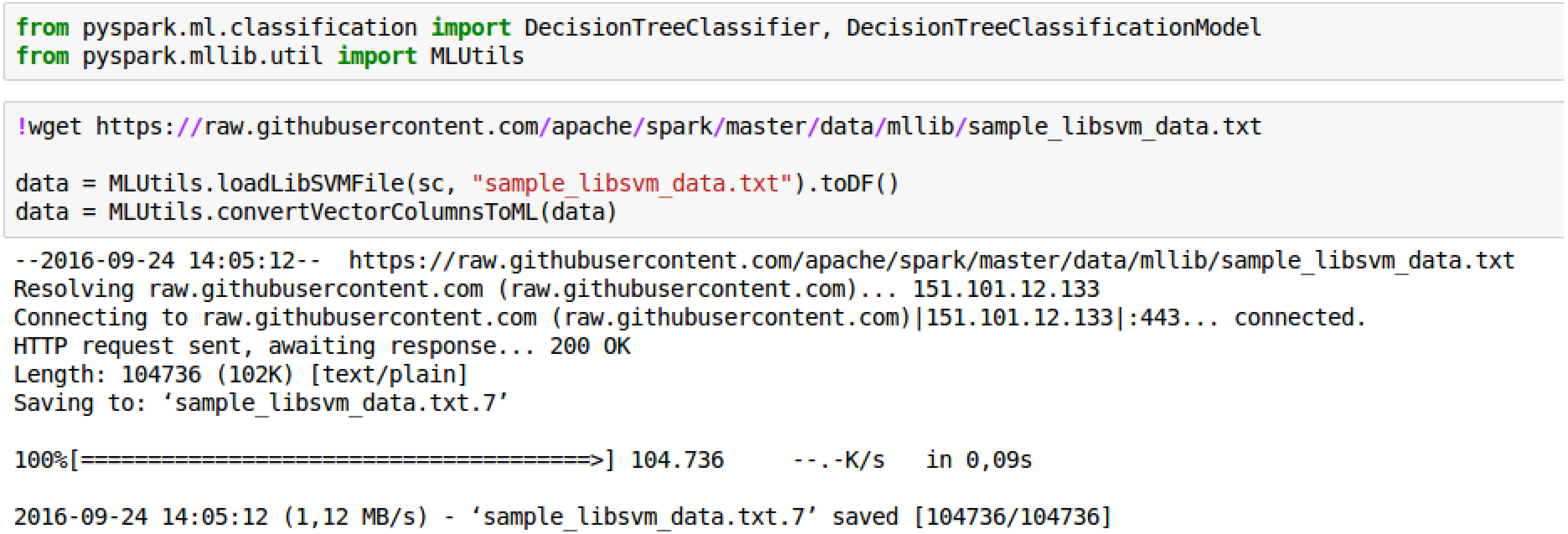
- More functionalities than the original MLlib API
- Separation of Decision Trees for classification and regression
- Use of DF metadata to distinguish between continuous and categorical features
- For classification trees
  - class conditional probabilities, that is, predicted probabilities of each class, made available
  - estimates of feature importance

Inputs and outputs

Param name	Type(s)	Default	Description
labelCol	Double	"label"	Label to predict
featuresCol	Vector	"features"	Feature vector

Param name	Type(s)	Default	Description	Notes
predictionCol	Double	"prediction"	Predicted label	
rawPredictionCol	Vector	"rawPrediction"	Vector of length # classes, with the counts of training instance labels at the tree node which makes the prediction	Classification only
probabilityCol	Vector	"probability"	Vector of length # classes equal to rawPrediction normalized to a multinomial distribution	Classification only

Loading data



```python
from pyspark.ml.classification import DecisionTreeClassifier, DecisionTreeClassificationModel
from pyspark.mllib.util import MLUtils
```

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

data = MLUtils.loadLibSVMFile(sc, "sample_libsvm_data.txt").toDF()
data = MLUtils.convertVectorColumnsToML(data)
```

```
--2016-09-24 14:05:12--  https://raw.githubusercontent.com/apache/spark/master/data/mllib/sample_libsvm_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: 104736 (102K) [text/plain]
Saving to: ‘sample_libsvm_data.txt.7’

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

2016-09-24 14:05:12 (1,12 MB/s) - ‘sample_libsvm_data.txt.7’ saved [104736/104736]
```

Creating the tree model

```python
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, IndexToString, VectorIndexer

labelIndexer = StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(data)

labelConverter = IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels(labelIndexer.labels)

featureIndexer = VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").setMaxCategories(4).fit(data)

dtC = DecisionTreeClassifier().setLabelCol("indexedLabel").setFeaturesCol("indexedFeatures")

pipelineClass = Pipeline().setStages([labelIndexer, featureIndexer, dtC, labelConverter])

trainingData, testData = data.randomSplit([0.7, 0.3])
```

DecisionTreeClassifier (2)

```
modelClassifier = pipelineClass.fit(trainingData)
treeModel = modelClassifier.stages[2]
predictionsClass = modelClassifier.transform(testData)
```

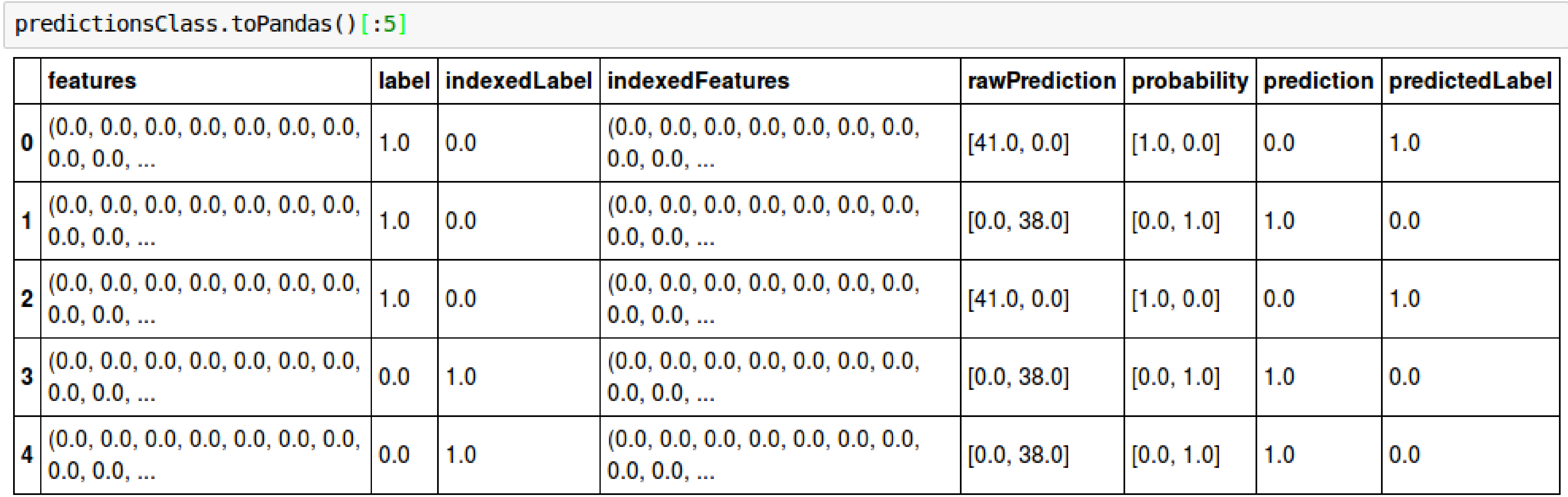
```
modelClassifier.stages
```

```
[StringIndexer_4f85a22d1a5692dc73c7,
 VectorIndexer_46698a2405ae191eed0a,
 DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4975bf032b3d5861e573) of depth 1 with 3 nodes,
 IndexToString_4246ad93c8b3ff0b5f09]
```

```
print treeModel.toDebugString
```

```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4975bf032b3d5861e573) of depth 1 with 3 nodes
  If (feature 434 <= 0.0)
   Predict: 1.0
  Else (feature 434 > 0.0)
   Predict: 0.0
```

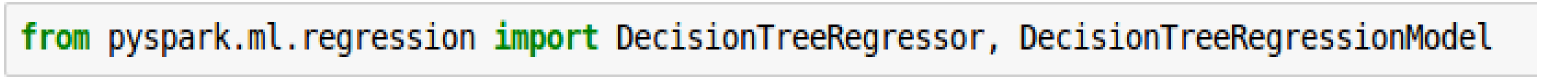
DecisionTreeClassifer (3)



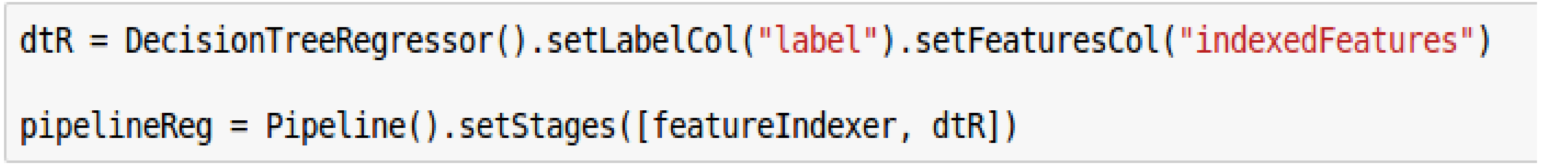
```
predictionsClass.toPandas()[:5]
```

	features	label	indexedLabel	indexedFeatures	rawPrediction	probability	prediction	predictedLabel
0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	1.0	0.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[41.0, 0.0]	[1.0, 0.0]	0.0	1.0
1	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	1.0	0.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 38.0]	[0.0, 1.0]	1.0	0.0
2	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	1.0	0.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[41.0, 0.0]	[1.0, 0.0]	0.0	1.0
3	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.0	1.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 38.0]	[0.0, 1.0]	1.0	0.0
4	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.0	1.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 38.0]	[0.0, 1.0]	1.0	0.0

DecisionTreeRegressor



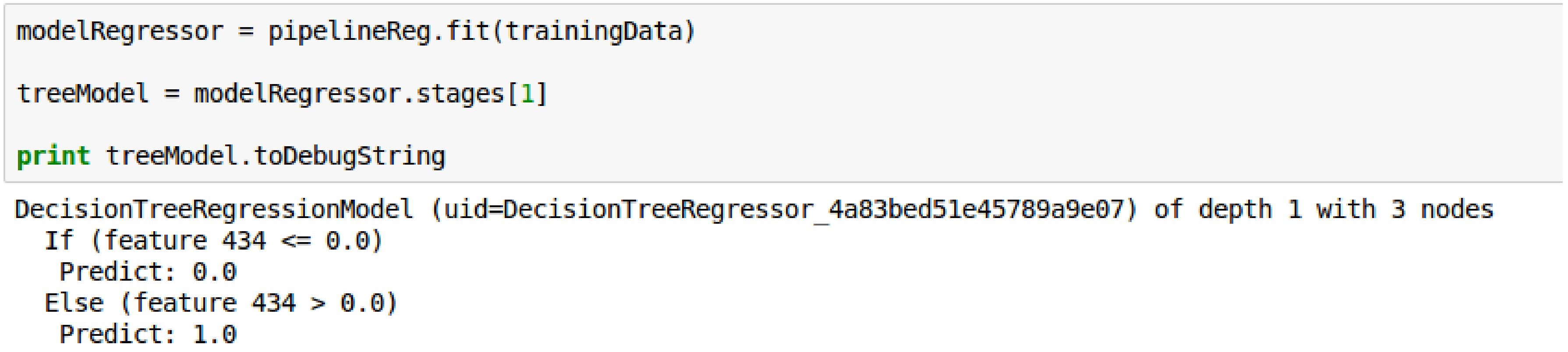
```
from pyspark.ml.regression import DecisionTreeRegressor, DecisionTreeRegressionModel
```



```
dtR = DecisionTreeRegressor().setLabelCol("label").setFeaturesCol("indexedFeatures")

pipelineReg = Pipeline().setStages([featureIndexer, dtR])
```

DecisionTreeRegressor (2)



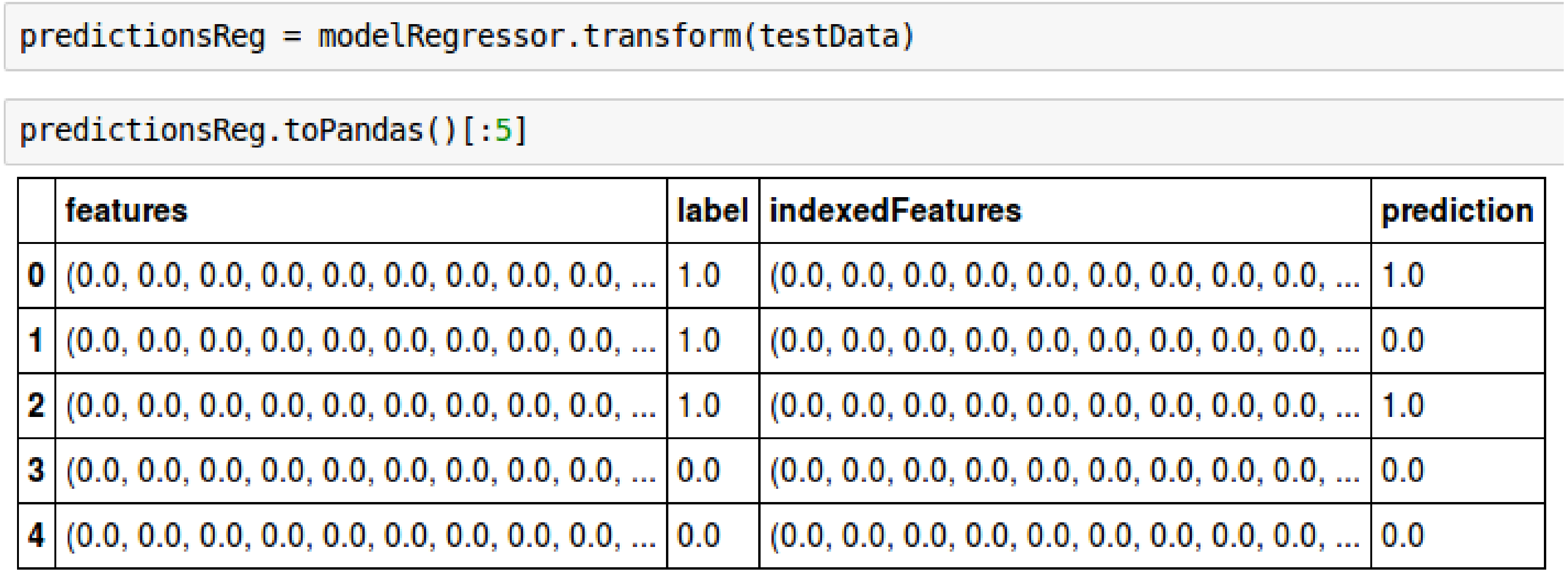
```
modelRegressor = pipelineReg.fit(trainingData)

treeModel = modelRegressor.stages[1]

print treeModel.toDebugString
```

```
DecisionTreeRegressionModel (uid=DecisionTreeRegressor_4a83bed51e45789a9e07) of depth 1 with 3 nodes
  If (feature 434 <= 0.0)
   Predict: 0.0
  Else (feature 434 > 0.0)
   Predict: 1.0
```

DecisionTreeClassifer (3)



```
predictionsReg = modelRegressor.transform(testData)
```

```
predictionsReg.toPandas()[:5]
```

	features	label	indexedFeatures	prediction
0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	1.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	1.0
1	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	1.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.0
2	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	1.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	1.0
3	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.0
4	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.0	(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.0

Problem specification parameters

- Describe the problem and the dataset
- Should be specified
- Do not require tuning
- Parameters:
  - **numClasses**: number of classes (classification)
  - **categoricalFeaturesInfo**: specifies which features are categorical and how many categorical values each of those features can take
    - optional: if not specified, algorithm may still get reasonable results
    - BUT performance should be better if categorical features are designated
    - map from feature indices to number of categories
    - features not in the map are treated as continuous

Stopping criteria

- Determine when the tree stops building
- May lead to overfitting
- Need to be validate on held-out test data

Stopping criteria, parameters

- **maxDepth**: maximum depth of a tree
  - if it increases (deeper trees):
    - more expressive, potentially higher accuracy
    - more costly to train
    - more likely to overfit
- **minInstancesPerNode**: each child must receive at least this number of instances for a node to be split further
  - commonly used in Random Forests as its trees are deeper and may overfit

Stopping criteria, parameters

–**minInfoGain**: the split must improve this much, in terms of information gain, for a node to be split further

- •The information gain is the difference between the parent node impurity and the weighted sum of the two child node impurities
- •Node impurity is a measure of the homogeneity of the labels at the node

Tunable parameters (1)

- **maxBins**: number of bins used when discretizing continuous features
  - must be at least the maximum number of categories for any categorical feature
  - if it increases:
    - allows the consideration of more split candidates and fine-grained split decisions
    - increases computation and communication

Tunable parameters (2)

maxMemoryInMB: amount of memory to be used for collecting sufficient statistics

- default = 256 MB, works in most scenarios
- if it increases:
  - can lead to faster training by allowing fewer passes over the data
  - there may be decreasing returns since amount of communication on each interaction also increases

Tunable parameters (3)

–**subsamplingRate**: fraction of the training data used for learning the decision tree

- more relevant for training ensemblers of trees (see next Lesson)

–**impurity**: impurity measure used to choose between candidate splits

- classification: Gini Impurity and Entropy
- regression: Variance

Lesson Summary

- Having completed this lesson, you should be able to:
  - Understand the Pipelines API for Decision Trees
  - Describe default's Input and Output columns
  - Perform classification and regression with Decision Trees
  - Understand and use Decision Trees' parameters