Decision trees

Lesson Objectives

- After completing this lesson, you should be able to:
  - Understand the API for Decision Trees
  - 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

Mllib’s implementation

- Supports binary and multiclass classification
- Supports regression
- Supports continuous and categorical features
- Partitions data by rows, allowing distributed training
- Used by the Pipelines API for Decision Trees

Loading data

```
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

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

data = MLUtils.loadLibSVMFile(sc, 'sample_libsvm_data.txt')

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

```
--2016-09-24 20:08:54--  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.8’

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

2016-09-24 20:08:55 (1,09 MB/s) - ‘sample_libsvm_data.txt.8’ saved [104736/104736]
```

Decision Tree Classifier

```
model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                     impurity='gini', maxDepth=5, maxBins=32)

predictions = model.predict(testData.map(lambda x: x.features))

labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)

testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())

print('Test Error = ' + str(testErr))
print('Learned classification tree model:')
print(model.toDebugString())
```

```
Test Error = 0.0
Learned classification tree model:
DecisionTreeModel classifier of depth 2 with 5 nodes
  If (feature 434 <= 0.0)
   If (feature 100 <= 165.0)
    Predict: 0.0
   Else (feature 100 > 165.0)
    Predict: 1.0
  Else (feature 434 > 0.0)
   Predict: 1.0
```

DecisionTreeRegressor

```python
model = DecisionTree.trainRegressor(trainingData, categoricalFeaturesInfo={},
                                    impurity='variance', maxDepth=5, maxBins=32)

predictions = model.predict(testData.map(lambda x: x.features))

labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)

testMSE = labelsAndPredictions.map(lambda (v, p): (v - p) * (v - p)).sum() / float(testData.count())

print('Test Mean Squared Error = ' + str(testMSE))
print('Learned regression tree model:')
print(model.toDebugString())
```

```
Test Mean Squared Error = 0.0
Learned regression tree model:
DecisionTreeModel regressor of depth 2 with 5 nodes
  If (feature 434 <= 0.0)
   If (feature 100 <= 165.0)
    Predict: 0.0
   Else (feature 100 > 165.0)
    Predict: 1.0
  Else (feature 434 > 0.0)
   Predict: 1.0
```

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
  - Perform classification and regression with Decision Trees
  - Understand and use Decision Trees' parameters