Random Forests

Lesson Objectives

- After completing this lesson, you should be able to:
  - Understand the API for Random Forests
  - Perform classification and regression with RFs
  - Understand and use RF's parameters

Ensemble Method

- Learning algorithm which creates a model composed of a set of other base models
- ‘Random Forests’ and ‘Gradient-Boosted Trees’ are ensemble algorithms based on decision trees
- Among top performers for classification and regression problems

Random Forests (RFs)

- Ensembles of Decision Trees
- One of the most successful machine learning models for classification and regression
- Combine many decision trees in order to reduce the risk of overfitting
- Supports binary and multiclass classification
- Supports regression
- Supports continuous and categorical features

RF: Basic Algorithm

- RF trains a bunch of decision trees separately
- RF Injects randomness into the training process
  - –bootstrapping: subsamples the original dataset on each iteration to get a different training set
  - –considers different random subsets of features to split on at each tree node
- Combined predictions from several trees reduces the variance of the predictions and improves the performance on test data
  - –classification: majority vote - each tree's prediction is counted as a vote for one class and the predicted label is the class with largest number of votes
  - –regression: average - each tree predicts a real value and the predicted label is equal to the average of all predictions

Random Forest Parameters (1)

- Most important parameters in Spark.ml implementation
- Parameters that CAN be tuned to improve performance:
  - **numTrees**: number of trees in the forest. If it increases:
    - the variance of predictions decreases, improving test-time accuracy
    - training time increases roughly linearly
  - **maxDepth**: maximum depth of each tree in the forest. If it increases:
    - model gets more expressive and powerful
    - takes longer to train
    - is more prone to overfitting

Random Forest Parameters (2)

- Parameters that DO NOT require tuning but CAN be tuned to speed up training:
  - **subsamplingRate**: specifies the fraction of size of the original dataset to be used for training each tree in the forest
    - default = 1.0, but decreasing it can speed up training
  - **featureSubsetStrategy**: specifies the fraction of total number of features to use as candidates for splitting at each tree node
    - decreasing it will speed up training
    - if too low, can also impact performance

RF Classification (1)

```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils

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

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

labels = testData.map(lambda x: x.label)
features = testData.map(lambda x: x.features)
```

RF Classification (2)

```
model = RandomForest.trainClassifier(trainingData,
                                     numClasses=2,
                                     categoricalFeaturesInfo={},
                                     numTrees=3,
                                     featureSubsetStrategy="auto",
                                     impurity='gini',
                                     maxDepth=4,
                                     maxBins=32)

print(model.toDebugString())
```

```
TreeEnsembleModel classifier with 3 trees

  Tree 0:
    If (feature 412 <= 0.0)
     If (feature 378 <= 0.0)
      Predict: 0.0
     Else (feature 378 > 0.0)
      Predict: 1.0
    Else (feature 412 > 0.0)
     Predict: 0.0
```

RF Classification (3)

```
predictions = model.predict(features)

labelsAndPredictions = labels.zip(predictions)

testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
```

```
Test Error = 0.0
```

RF for regression

```
model = RandomForest.trainRegressor(trainingData,
                                    categoricalFeaturesInfo={},
                                    numTrees=3,
                                    featureSubsetStrategy="auto",
                                    impurity='variance',
                                    maxDepth=4,
                                    maxBins=32)

print(model.toDebugString())
```

```
TreeEnsembleModel regressor with 3 trees

  Tree 0:
    If (feature 406 <= 72.0)
     If (feature 293 <= 253.0)
      Predict: 0.0
     Else (feature 293 > 253.0)
      Predict: 1.0
    Else (feature 406 > 72.0)
     Predict: 1.0
```

RF for regression

```
predictions = model.predict(features)

labelsAndPredictions = labels.zip(predictions)

testMSE = labelsAndPredictions.map(lambda (v, p): (v - p) * (v - p)).sum() / float(testData.count())
print('Test Mean Squared Error = ' + str(testMSE))
```

```
Test Mean Squared Error = 0.0292397660819
```

Lesson Summary

- Having completed this lesson, you should be able to:
  - Understand how to run a random forest in Spark
  - Grasp most of the parameters and their effects
  - Understand how to use RF for regression and categorization