DATA WAREHOUSE & MINING

TA COURSE #07

CLASSIFICATION

TA COURSE #07

CLASSIFICATION

SPARK MLLIB

WHAT IS MLLIB

- **MLlib** is Apache Spark's scalable machine learning library.
- Ease of Use
  - Usable in Java, Scala, Python, and R.
- Performance
  - High-quality algorithms, 100x faster than MapReduce.
- Runs Everywhere
  - Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud, against diverse data sources.

TA COURSE #07

CLASSIFICATION

DECISION TREE

Access here for more detailed info

DECISION TREE CLASSIFIER

- A popular family of classification and regression methods
- Decision trees are widely used since they are
  - Easy to interpret, handle categorical features, extend to the multiclass classification setting
  - Do not require feature scaling
  - Be able to capture non-linearities and feature interactions.
- The spark.ml implementation supports decision trees for binary and multiclass classification and for regression, using both continuous and categorical features.

SPLIT CANDIDATES

- Continuous features
  - Range for one value
- Categorical features
  - Bin for tags

STOPPING RULE

- The recursive tree construction is stopped at a node when one of the following conditions is met:
  - The node depth is equal to the maxDepth training parameter.
  - No split candidate leads to an information gain greater than minInfoGain.
  - No split candidate produces child nodes which each have at least minInstancesPerNode training instances.

TA COURSE #07

CLASSIFICATION

USAGE TIPS

Access here for more detailed info

PROBLEM SPECIFICATION PARAMETERS

- algo: Type of decision tree, either Classification or Regression.
- numClasses: Number of classes (for Classification only).
- categoricalFeaturesInfo: Specifies which features are categorical and how many categorical values each of those features can take. This is given as a map from feature indices to feature arity (number of categories). Any features not in this map are treated as continuous.

PROBLEM SPECIFICATION PARAMETERS

- For example, Map(0 -> 2, 4 -> 10) specifies that feature 0 is binary (taking values 0 or 1) and that feature 4 has 10 categories (values {0, 1, ..., 9}). Note that feature indices are 0-based: features 0 and 4 are the 1st and 5th elements of an instance's feature vector.
- Note that you do not have to specify categoricalFeaturesInfo. The algorithm will still run and may get reasonable results. However, performance should be better if categorical features are properly designated.

STOPPING CRITERIA

- maxDepth: Maximum depth of a tree. Deeper trees are more expressive (potentially allowing higher accuracy), but they are also more costly to train and are more likely to overfit.
- minInstancesPerNode: For a node to be split further, each of its children must receive at least this number of training instances. This is commonly used with RandomForest since those are often trained deeper than individual trees.
- minInfoGain: For a node to be split further, the split must improve at least this much (in terms of information gain).

TUNABLE PARAMETERS

- maxBins: Number of bins used when discretizing continuous features.
  - Increasing maxBins allows the algorithm to consider more split candidates and make fine-grained split decisions. However, it also increases computation and communication.
- Note that the maxBins parameter must be at least the maximum number of categories for any categorical feature.

TUNABLE PARAMETERS

- maxMemoryInMB: Amount of memory to be used for collecting sufficient statistics.
- subsamplingRate: Fraction of the training data used for learning the decision tree.
- impurity: Impurity measure used to choose between candidate splits.

TA COURSE #07

CLASSIFICATION

EXAMPLE CODE

Access here for more detailed info

DECISION TREE CLASSIFIER

```scala
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.util.MLUtils

// Load and parse the data file.
val data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")
// Split the data into training and test sets (30% held out for testing)
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

// Train a DecisionTree model.
//  Empty categoricalFeaturesInfo indicates all features are continuous.
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 5
val maxBins = 32

val model = DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,
  impurity, maxDepth, maxBins)

// Evaluate model on test instances and compute test error
val labelAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
println(s"Test Error = $testErr")
println(s"Learned classification tree model:\n ${model.toDebugString}")

// Save and load model
model.save(sc, "target/tmp/myDecisionTreeClassificationModel")
val sameModel = DecisionTreeModel.load(sc, "target/tmp/myDecisionTreeClassificationModel")
```

THX!