FAQs

- Make an appointment for your term project

Today’s topics

- In-Memory cluster computing
  - Apache Spark
  - Case Study: Random Forest Model

Why partitioning?

- Consider an application that keeps a large table of user information in memory
  - An RDD of (UserID, UserInfo) pairs
  - The application periodically combines this table with a smaller file representing events that happened in the last five minutes

Using partitionBy()

- Transforms userData to hash-partitioned RDD
Large Scale Data Analytics
In-Memory Cluster Computing: Apache Spark

Software stack

Spark stack of libraries
- SQL and DataFrames
- Machine learning
- GraphX
- Spark Streaming

Large Scale Data Analytics
In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests

Regressions and Classifications
- Regression analysis
  - A statistical process for estimating the relationships among variables
  - How the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed
  - Predicting numerical values
    - Income, temperature, size
- Classifications
  - Identifying which of a set of categories a new observation belongs
  - Predicting a label or category
    - Spam, picture of a cat

Vectors and Features
- Input and output to regression and classification
- Example
  - Predicting tomorrow's temperature given today's weather
  - Features (dimensions, predictors, or variables) of today's weather
    - Today's high temperature
    - Today's low temperature
    - Today's average humidity
    - Whether it's cloudy, rainy, or clear today
    - The number of weather forecasters predicting a cold snap tomorrow
  - Each of these features can be quantified
    - 13.1, 19.0, 0.73, cloudy, 1
- Feature vector
  - These features together, in order
  - (13.1, 19.0, 0.73, cloudy, 1)
  - Features do NOT need to be the same type
- Categorical features
  - Features that have no ordering
    - Cloudy, clear
  - Numeric features
    - Features that can be quantified by a number and have a meaningful ordering
      - 23°F, 56°F (23°F < 56°F)
Training Examples

- A learning algorithm needs to train on data in order to make predictions
- Inputs
  - Correct outputs (from historical data)
  - "One day, the weather was between 12 and 16 degrees Celsius, with 10% humidity, clear, with no forecast of a cold snap, and the following day the high temperature was 17.2 degrees"
- Target (output)
  - 17.2 degrees
- Feature vector often includes the target value
  - (13.1, 19.0, 0.73, cloudy, 1, 17.2)

Decision Trees and Forests

- Decision trees can naturally handle both categorical and numeric features
- Easy to parallelize
- Robust to outliers
  - A few extreme and possibly erroneous data points may not affect predictions at all

Random Decision Forests

Example: Finding a good pet

<table>
<thead>
<tr>
<th>Name</th>
<th>Weight (kg)</th>
<th># of legs</th>
<th>color</th>
<th>Is it a good pet for you?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fido</td>
<td>20.5</td>
<td>4</td>
<td>Brown</td>
<td>Y</td>
</tr>
<tr>
<td>Mr. Slither</td>
<td>3.1</td>
<td>0</td>
<td>Green</td>
<td>N</td>
</tr>
<tr>
<td>Nemo</td>
<td>0.2</td>
<td>0</td>
<td>Tan</td>
<td>Y</td>
</tr>
<tr>
<td>Dumbo</td>
<td>139.8</td>
<td>4</td>
<td>Grey</td>
<td>N</td>
</tr>
<tr>
<td>Kitty</td>
<td>12.1</td>
<td>4</td>
<td>Grey</td>
<td>Y</td>
</tr>
<tr>
<td>Jim</td>
<td>150.9</td>
<td>2</td>
<td>Tan</td>
<td>N</td>
</tr>
<tr>
<td>McPigeon</td>
<td>1.0</td>
<td>2</td>
<td>Grey</td>
<td>Y</td>
</tr>
<tr>
<td>Spot</td>
<td>10.0</td>
<td>4</td>
<td>Brown</td>
<td>N</td>
</tr>
</tbody>
</table>

Example: Finding a good pet

Decision tree for “Finding a good pet” example

Covertype dataset

- Dataset with records of the types of forest covering parcels of land in Colorado, USA
- Each example contains several features describing
  - Each parcel of land
    - Elevation, slope, distance to water, shade, and soil type
  - The forest cover type is to be predicted
    - From the rest of features
  - Total 54 features
  - Used in Kaggle competition
  - Includes categorical and numeric features
- 581,012 examples

Attribute information

- Elevation / quantitative (metres) / Elevation in metres
- Aspect / quantitative / azimuth / Aspect in degrees azimuth
- Slope / quantitative / degrees / Slope in degrees
- Horizontal_Distance_To_Hydrology / quantitative / meters / Horz Dist to nearest surface water features
- Vertical_Distance_To_Hydrology / quantitative / meters / Vert Dist to nearest surface water features
- Horizontal_Distance_To_Roadways / quantitative / meters / Horz Dist to nearest roadway
- Hillshade_9am / quantitative / 0 to 255 index / Hillshade index at 9am, summer solstice
- Hillshade_Noon / quantitative / 0 to 255 index / Hillshade index at noon, summer solstice
- Hillshade_3pm / quantitative / 0 to 255 index / Hillshade index at 3pm, summer solstice
- Horizontal_Distance_To_Fire_Points / quantitative / meters / Horz Dist to nearest wildfire ignition points
- Wilderness_Area / binary (columns) / (presence), 0 (absence) / Wilderness area designation
- Soil_Type / qualitative / 0 (absence) or 1 (presence) / Soil Type designation
- Cover_Type / qualitative / 1 to 7 / Forest Cover Type designation
Preparing data

- The `covtype.data` file should be extracted and copied into HDFS
  - File is available at `/user/ds/`

- **LabeledPoint**
  - The Spark MLlib abstraction for a feature vector
  - Consists of a Spark MLlib Vector of features, and a target value (label)
  - It can be used with categorical features, with appropriate encoding

Using **LabeledPoint** for the categorical features

- **One-hot coding**
  - One categorical feature that takes on N distinct values becomes N numeric features, each taking on the value 0 or 1
  - Exactly one of the N values have value 1 and the others are 0
  - Cloudy: 1,0,0
  - Rainy: 0,1,0
  - Clear: 0,0,1

- **1-of-n coding**
  - Cloudy: 1
  - Rainy: 2
  - Clear: 3

Categorical values in **Covtype** data set

- The `covtype.info` file says that four of the columns are actually a one-hot encoding of a single categorical feature
  - `Wilderness_Type`, with four values
  - Likewise, 40 of the columns are really one `Soil_Type` categorical feature
- The target itself is a categorical value encoded as the values 1 to 7
- The remaining features are numeric features in various units, like meters, degrees, or a qualitative “index” value

A First Decision Tree

- Spark MLlib requires input in the form of LabeledPoint objects

```scala
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression._
val rawData = sc.textFile("hdfs:///user/ds/covtype.data")
val data = rawData.map{x => val values = x.split(',').map(_.toDouble)
  val featureVector = Vectors.dense(values.init)
  val label = values.last - 1
  LabeledPoint(label, featureVector)}
```

Splitting data

- Training, cross-validation, and test
  - 80% of data for training and 10% each for cross-validation and test
  - Training and CV sets are used to choose a good setting of hyperparameters for this data set
- Test set is used to produce an unbiased evaluation of the expected accuracy of a model built with those hyperparameters

```scala
val Array(trainData, cvData, testData) = data.randomSplit(Array(0.8, 0.1, 0.1))
trainData.cache()
cvData.cache()
testData.cache()
```

Building a **DecisionTreeModel** on the training set

- Building a DecisionTreeModel on the training set with some default arguments
  - Compute some metrics about the resulting model using the CV set
Building a `DecisionTreeModel` on the training set

```python
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.tree._
import org.apache.spark.mllib.tree.model._
import org.apache.spark.rdd._

def getMetrics(model: DecisionTreeModel, data: RDD[LabeledPoint]): MulticlassMetrics = {
  val predictionsAndLabels = data.map(example => (model.predict(example.features), example.label))
  new MulticlassMetrics(predictionsAndLabels)
}

val model = DecisionTree.trainClassifier(trainData, 7, Map[Int, Int](), "gini", 4, 100)
val metrics = getMetrics(model, cvData)
```

Confusion matrix
- 7 x 7 matrix
- The row number corresponds to an actual correct value
- The column number corresponds to a predicted value

```plaintext
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14019.0</td>
<td>6630.0</td>
<td>15.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>391.0</td>
</tr>
<tr>
<td>1</td>
<td>5413.0</td>
<td>22399.0</td>
<td>438.0</td>
<td>16.0</td>
<td>0.0</td>
<td>3.0</td>
<td>50.0</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>457.0</td>
<td>2999.0</td>
<td>73.0</td>
<td>0.0</td>
<td>12.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
<td>1.0</td>
<td>163.0</td>
<td>117.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
<td>872.0</td>
<td>40.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
<td>500.0</td>
<td>1138.0</td>
<td>36.0</td>
<td>0.0</td>
<td>48.0</td>
<td>0.0</td>
</tr>
<tr>
<td>6</td>
<td>1091.0</td>
<td>41.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>891.0</td>
</tr>
</tbody>
</table>
```

Precision in multiclass metrics
- Binary classification
  - Positive vs. negative class
  - Precision is the fraction of examples that the classifier marked positive that are actually positive.
  - $PPV = TP/(TP+FP)$
  - Recall is the fraction of all examples that are actually positive that the classifier marked positive.
  - $TPR = TP/(TP+FN)$

- Multiclass problem
  - Positive class vs. negative (all else)

Is 70% accuracy good?
- Classifier that classifies at random in proportion to its prevalence in the training set
- What is the baseline?
  - A broken clock will be correct twice a day
  - Randomly guessing a classification would also be occasionally correct

Decision Tree Hyperparameters
- **Hyperparameters**
  - Values we have to choose by building models
  - Maximum depth, maximum bins, and impurity measure
- **Maximum depth**
  - Limits the number of levels in the decision tree
  - Useful to avoid overfitting the training data
- **Maximum bins**
  - `feature <= value`
  - `feature in [value1, value 2, ...]`
  - A larger number of bins requires more processing time
  - More optimal decision rule

Decision Tree Hyperparameters
- **Good rule should distinguish examples more meaningfully**
- **Example**
  - A rule that divides the Covtype set into only 1-3 category and 4-7 category would be a great rule
  - A good rule divides the training data's target values into relatively homogeneous or “pure” subsets
- **Minimizing the impurity of the two subsets**
Gini impurity

- Gini impurity
  - Measuring impurity degree
  - Within a subset, it is the probability that a randomly chosen classification of a randomly chosen example is incorrect
  - Includes the sum of products of proportions
    \[ I_G(p) = 1 - \sum_{i=1}^{N} p_i^2 \]
  - If the subset contains only one class
    - The value is 0

Entropy

- Borrowed from information theory
- How much uncertainty does the collection of target values in the subset contain?

\[ I_E(p) = -\sum_{i=1}^{N} p_i \log(p_i) \]

Tuning Decision Trees

- Spark tries a number of combinations of impurity measure, maximum depth or number of bins and reports the results

```scala
val evaluations = for {
  impurity <- Array("gini", "entropy");
  depth <- Array(1, 20);
  bins <- Array(10, 300)
} yield {
  val model = DecisionTree.trainClassifier(trainData, 7, Map[Int, Int]()(), impurity, depth, bins)
  val predictionsAndLabels = cvData.map(example => (model.predict(example.features), example.label))
  val accuracy = new MulticlassMetrics(predictionsAndLabels).precision((impurity, depth, bins), accuracy)
}
evaluations.sortBy(_._2).reverse.foreach(println)
```

Categorical Features Revisited

- Map[Int, Int]{}
  - Keys
    - Indices of features in the input Vector
  - Values
    - Distinct value counts
  - Empty Map()
    - No features should be treated as categorical
    - All are numeric
  - Numeric representation of categorical features
    - It can cause errors
    - The algorithm would be trying to learn from an ordering that has no meaning

Treating the categorical features with one-hot encoding

- Encodes the categorical features as several binary 0/1 values
- Any decision rule on the “numeric” features will choose thresholds between 0 and 1
  - All are equivalent since all values are 0 or 1
- Considers the values of the underlying categorical feature individually
  - Increases memory usage
Converting one-hot encoding to 1-n encoding [1/3]

```scala
val data = csvdata.map { line =>
  line.split(',') match{
    case Array(_):=
      FeatureVector(values.slice(10, 10).toDouble, wildeness >= wildeness)
  }
}
```

- 4 "wilderness" features
- 40 "soil" features
- Add derived features back to first 10

Converting one-hot encoding to 1-n encoding [2/3]

```scala
val evaluations = for (impurity <- Array("gini", "entropy"))
  yield {
    val model = DecisionTree.trainClassifier(trainData, Map(10 -> 4, 11 -> 40),
      impurity, depth, bins;
    val trainAccuracy = getMetrics(model, trainData);
    val testAccuracy = getMetrics(model, testData);
    val summary = Array((impurity, depth, bins),
      (trainAccuracy, testAccuracy, trainData, testData));
  }
```

- Specify value count for categorical features 10, 11
- Causes these features to be treated as categorical

Converting one-hot encoding to 1-n encoding [3/3]

```scala
\[\text{val} \text{model} = \text{DecisionTree.trainClassifier contraceptions, Map(10 -> 4, 11 -> 40), \text{impurity, depth, bins)}; \text{val} \text{accuracy} = \text{getMetrics(model, \text{trainData})}; \text{val} \text{precision} = \text{getPrecision(model, \text{trainData})}; \text{val} \text{speed} = \text{getAccuracy(model, \text{trainData})}; \text{val} \text{confusion} = \text{getConfusion(model, \text{trainData})}; \text{val} \text{predictions} = \text{getPredictions(model, \text{trainData})}; \]
```

- Tree-building process completes several times faster
- By treating categorical features as categorical features, it improves accuracy by almost 3%

Does decision tree algorithm build the same tree every time?

- Over N values
- There are 2^N-2 possible decision rules
- Decision trees use several heuristics to narrow down the rules to be considered
- The process of picking rules involves some randomness
- Only a few features, picked at random, are looked at each time
- Only values from a random subset of the training data are looked at
- Trades a bit of accuracy for a lot of speed
- Decision tree algorithm won’t build the same tree every time

RandomForest

```scala
\[\text{val} \text{forest} = \text{RandomForest.trainClassifier(\text{trainData}, Map(10 -> 4, 11 -> 40), \text{"auto"}, \text{"entropy"}, 20, 10)}; \]
```

- Number of trees to build
  - Here, 20
- "auto"
  - The strategy for choosing which features to evaluate at each level of the tree
  - The random decision forest implementation will NOT even consider every feature as the basis of a decision rule
  - Only a subset of all features

Making predictions

- Classification
  - Majority vote
  - Each tree’s prediction is counted as a vote for one class
  - The label is predicted to be the class which receives the most votes
- Regression
  - Averaging
  - Each tree predicts a real value
  - The label is predicted to be the average of the tree predictions
Making predictions [2/2]

- The results of the DecisionTree and RandomForest training
  - DecisionTreeModel and RandomForestModel objects

- predict() method
  - Accepts a Vector object

- We can classify a new example by converting it to a feature vector in the same way and predicting its target class

```scala
val input = "2709,135,28,23,224,853,207,61,6094,0,29"
val vector = Vectors.dense(input.split(',').map(_.toDouble))
forest.predict(vector)
```