FAQs

• TP1 scores have been posted
• Programming Assignment 3 has been posted

Today's topics

• In-Memory cluster computing
  • Apache Spark
  • Random Forests
• Distributed File Systems

Large Scale Data Analytics

In-Memory Cluster Computing: Apache Spark

Predicting Forest Cover with Decision Trees and Forests

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(line =>
  val values = line.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

```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

```
<table>
<thead>
<tr>
<th></th>
<th>0.0 1.0 2.0 3.0 4.0 5.0 6.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>14019.0 8630.0 18.0 0.0 0.0 1.0 891.0</td>
</tr>
<tr>
<td>1.0</td>
<td>5413.0 22299.0 438.0 16.0 0.0 50.0</td>
</tr>
<tr>
<td>2.0</td>
<td>0.0 657.0 2999.0 73.0 0.0 12.0 0.0</td>
</tr>
<tr>
<td>3.0</td>
<td>0.0 1.0 141.0 117.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>4.0</td>
<td>0.0 97.0 99.0 8.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>5.0</td>
<td>0.0 103.0 1139.0 36.0 0.0 86.0 0.0</td>
</tr>
<tr>
<td>6.0</td>
<td>1891.0 81.0 0.0 0.0 0.0 0.0 0.0</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
    - \( \text{PPV} = \frac{TP}{TP+FP} \)
  - Recall is the fraction of all examples that are actually positive that the classifier marked positive
    - \( \text{TPR} = \frac{TP}{P=TP+FN} \)

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

```
metrics.precision
```

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**
  - \( \text{feature <= value} \)
  - \( \text{feature in (value1, value2, ...)} \)
  - A larger number of bins requires more processing time
  - More optimal decision rule
Decision Tree Hyperparameters [2/2]

• Good rule should distinguish examples more meaningfully.
• Example:
  E.g. 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.

Example: Finding a good pet

<table>
<thead>
<tr>
<th>Name</th>
<th>Weight (kg)</th>
<th>Legs</th>
<th>Color</th>
<th>Good pet?</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>1390.8</td>
<td>4</td>
<td>Grey</td>
<td>N</td>
</tr>
<tr>
<td>Kitty</td>
<td>150.9</td>
<td>2</td>
<td>Grey</td>
<td>Y</td>
</tr>
<tr>
<td>Nemo</td>
<td>0.1</td>
<td>100</td>
<td>Brown</td>
<td>N</td>
</tr>
<tr>
<td>McPigeon</td>
<td>1.0</td>
<td>2</td>
<td>Grey</td>
<td>N</td>
</tr>
<tr>
<td>Spot</td>
<td>10.0</td>
<td>4</td>
<td>Brown</td>
<td>Y</td>
</tr>
</tbody>
</table>

Original Feature Set
(20.5, 4, Brown, Y)
(3.1, 0, Green, N)
(0.2, 0, Tan, Y)
(1390.8, 4, Grey, N)
(12.1, 4, Grey, Y)
(150.9, 2, Tan, N)
(0.1, 100, Brown, N)
(1.0, 2, Grey, N)
(10.0, 4, Brown, Y)

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.

Example 1

• Example 1
  • Calculate the Gini impurity of our entire dataset.
  • Random selection: Blue (50%), Green (50%)
  • Classification: it will be also “random”.
  • What is the probability we classify our datapoints incorrectly?
  • 50%
Gini impurity – Example 1: Random split

- If we have $C$ total classes and $p(i)$ is the probability of picking a datapoint with class $i$
- Gini Impurity
  \[ G = \sum_{i=1}^{C} p(i)(1-p(i)) \]
  \[ G = p(1)x(1-p(1)) + p(2)x(1-p(2)) \]
  \[ = 0.5 \times (1-0.5) + 0.5 \times (1-0.5) \]
  \[ = 0.5 \]

<table>
<thead>
<tr>
<th>Event</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pick blue and classify blue (correct)</td>
<td>25%</td>
</tr>
<tr>
<td>Pick blue and classify green (incorrect)</td>
<td>25%</td>
</tr>
<tr>
<td>Pick green and classify blue (incorrect)</td>
<td>25%</td>
</tr>
<tr>
<td>Pick green and classify green (correct)</td>
<td>25%</td>
</tr>
</tbody>
</table>

Gini impurity – Example 2: Perfect split

- Left branch are only blues and right branch are only green
- Gini Impurity
  \[ G_{left} = p(1)x(1-p(1)) + p(2)x(1-p(2)) \]
  \[ = 1 \times (1-1) + 0 \times (1-0) \]
  \[ = 0 \]

- Gini Impurity
  \[ G_{right} = p(1)x(1-p(1)) + p(2)x(1-p(2)) \]
  \[ = 0 \times (1-0) + 1 \times (1-1) \]
  \[ = 0 \]
- Both branches have 0 impurity! (Best Gini impurity)

Gini impurity – Example 3: Imperfect split

- Gini Impurity
  \[ G = \sum_{i=1}^{C} p(i)(1-p(i)) \]
  \[ G_{left} = p(1)x(1-p(1)) + p(2)x(1-p(2)) \]
  \[ = \frac{1}{6} \times (1-\frac{1}{6}) + \frac{5}{6} \times (1-\frac{5}{6}) \]
  \[ = \frac{5}{18} = 0.278 \]

- Gini Gain
  - The quality of the split by weighting the impurity of each branch by how many elements it has
  \[ (0.4 \times 0) + (0.6 \times 0.278) = 0.167 \]
  - Higher Gini Gain means better split

Training a Decision Tree with Gini Impurity

- Root Node
  - Finding the Root Node
    - Which feature do we have to use? X or Y?
    - What will be the test threshold? ($x=2$ or $x=3$)
  - Example → see the picture
  - Using x feature with a test threshold of 2
  - Good split
    - Separating different classes as much as possible
  - Split
    - Left Branch
      - Red/blue/green with two features x and y
    - Right Branch
      - Red/blue/green with two features x and y

Training a Decision Tree with Gini Impurity

- Root Node
  - Find the threshold that provides highest Gini Gain (low Gini Impurity)
  - We will go over every possible split to find the best threshold
Training a Decision Tree with Gini Impurity:

- **Root Node**
  - Gini Impurity of the entire dataset
  - $G = \sum_{i=1}^{c} p(i)(1 - p(i))$
  - Gain = $G_{initial} - \frac{1}{9}G_{left} - \frac{8}{9}G_{right}$

- **Second Node**
  - Do the same thing...

- **When to stop?**
  - Until have a Gini Gain of 0
  - We trained a decision tree!

**Entropy**

- Borrowed from information theory
- How much uncertainty does the collection of target values in the subset contain?
  - $I(p) = \sum_{i} p \log(1/p) - \sum_{i} p \log(p)$

**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) ...
```
Tuning Decision Trees

• continued

```scala
[(entropy, 20, 300), 0.9125545571245186]  
[(gini, 20, 300), 0.9696597845406046]  
[(gini, 20, 10), 0.8848951647411211]  
[(entropy, 20, 10), 0.8848951647411211]  
[(gini, 1, 300), 0.6358065896448438]  
[(gini, 1, 10), 0.6355669661959777]  
[(entropy, 1, 300), 0.4861446298673513]  
[(entropy, 1, 10), 0.4861446298673513]  
```

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

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

```
val data = rawData.map(line => val values = line.split(',').map(_.toDouble)  
val wilderness = values.slice(10, 14).indexOf(1.0).toDouble  
val soil = values.slice(14, 54).indexOf(1.0).toDouble  
val featureVector = Vectors.dense(values.slice(0, 10) :+ wilderness :+ soil)  
val label = values.last-LabeledPoint(label, featureVector)  
```

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

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

```
val evaluations =  
for (impurity <- Array("gini", "entropy"); depth <- Array(10, 20, 30); bins <- Array(40, 300)) yield {  
  val model = DecisionTree.trainClassifier(trainData, 7, Map(10 -> 4, 11 -> 40), impurity, depth, bins)  
  val trainAccuracy = getMetrics(model, trainData).precision((impurity, depth, bins),  
  val cvAccuracy = getMetrics(model, cvData).precision((impurity, depth, bins),  
  ((impurity, depth, bins), (trainAccuracy, cvAccuracy)))  
```

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

```
val data = rawData.map(line => val values = line.split(',').map(_.toDouble)  
val wilderness = values.slice(10, 14).indexOf(1.0).toDouble  
val soil = values.slice(14, 54).indexOf(1.0).toDouble  
val featureVector = Vectors.dense(values.slice(0, 10) :+ wilderness :+ soil)  
val label = values.last-LabeledPoint(label, featureVector)  
```

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

• Tree-building process completes several times faster
• By treating categorical features as categorical features, it improves accuracy by almost 3%
Ensemble Approach: 1. Bagged Decision Tree

- Bagging
- Bootstrap Aggregating

Step 1) Sample \( n \) data points with replacement
Step 2) Train with \( n \) data points
Step 3) repeat step 1 and 2, \( t \) times

We have \( t \) decision trees
- Take the majority vote if our trees produce class labels (like colors)
- Take the average if our trees produce numerical values (e.g. when predicting temperature, price, etc)

Ensemble Approach: 2. Random Forests

- Number of trees and number of features to try when finding the best split
- Dataset with \( p \) features
- Select a subset of features for each tree
  - Usually \( \sqrt{p} \) or \( p/3 \)
  - Feature bagging
- This provides randomness that makes
  - Individual tree more unique
  - Reduces correlation between trees

RandomForest

```scala
val forest = RandomForest.trainClassifier(
  trainData,
  7,
  Map(10 -> 4, 11 -> 40),
  20,
  "auto",
  "entropy",
  30,
  300
)

• 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

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

  ```scala
  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
  ```

Distributed file systems

- Hadoop vs. Google File System?
Part 1. Large scale data analysis using MapReduce
Distributed File System
Google File System (GFS)

A server room in Council Bluffs, Iowa

Data Center

Data centers
- US data centers consumes about 70 billion kilowatt-hours of electricity in 2014
  - 2 percent of country's total energy consumption
  - 6.4 million average American homes