PART 1. LARGE SCALE DATA ANALYTICS
IN-MEMORY CLUSTER COMPUTING

PART 2. LARGE SCALE DATA STORAGE SYSTEMS
DISTRIBUTED FILE SYSTEMS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

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

```
metrics.confusionMatrix
...
14019.0 6630.0 15.0 0.0 0.0 1.0 391.0
5413.0 22399.0 438.0 16.0 0.0 3.0 50.0
0.0 457.0 2999.0 73.0 0.0 12.0 0.0
0.0 1.0 163.0 117.0 0.0 0.0 0.0
0.0 872.0 40.0 0.0 0.0 0.0 0.0
0.0 500.0 1138.0 36.0 0.0 48.0 0.0
1091.0 41.0 0.0 0.0 0.0 0.0 891.0
```

```
metrics.precision
...
0.7030630195577938
```

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
    • \( PPV = \frac{TP}{TP+FP} \)
  • Recall is the fraction of all examples that are actually positive that the classifier marked positive
    • \( TPR = \frac{TP}{TP+FN} = \frac{TP}{P} \)

• 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 [1/2]

• 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 [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 (hg)</th>
<th># of 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>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>Millie</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)
Decision tree for “Finding a good pet” example

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)

Weight >= 100kg?

Yes

Not suitable

No

Is color green

Yes

Not suitable

No

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

This split was not good. How can we quantify that?
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

Gini impurity – 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%

<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 1: Random split

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

<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
- $G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$
- $G_{left} = p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  - $= 1 \times (1-1) + 0 \times (1-0)$
  - $= 0$
- $G_{right} = p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  - $= 0 \times (1-0) + 1 \times (1-1)$
  - $= 0$
- Both branches have 0 impurity! (Best Gini impurity)
Gini impurity – Example 3: Imperfect split

- $G = \sum_{i=1}^c p(i) \times (1 - p(i))$
- $G_{\text{left}} = p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  - $= 1 \times (1-1) + 0 \times (1-0)$
  - $= 0$
- $G_{\text{right}} = p(1) \times (1 - p(1)) + p(2) \times (1 - p(2))$
  - $= \frac{1}{6} \times (1-\frac{1}{6}) + \frac{5}{6} \times (1-\frac{5}{6})$
  - $= \frac{5}{18} = 0.278$

- **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

- The 3 class dataset
  - Red/blue/green with two features $x$ and $y$
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

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

<table>
<thead>
<tr>
<th>Split</th>
<th>Left Branch</th>
<th>Right Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 0.4</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>
Training a Decision Tree with Gini Impurity

: Root Node

- Gini Impurity of the entire dataset

\[ G = \sum_{i=1}^{c} p(i)(1 - p(i)) \]

- \( G_{\text{initial}} = (1/3 \times 2/3) + (1/3 \times 2/3) + (1/3 \times 2/3) = 2/3 \)

- \( G_{\text{left}} = 0 \times 1 + 1 \times 0 + 0 \times 1 = 0 \)

- \( G_{\text{right}} = 3/8 \times 5/8 + 2/8 \times 6/8 + 3/8 \times 5/8 = 21/32 \)

- \( \text{Gain} = G_{\text{initial}} - 1/9G_{\text{left}} - 8/9G_{\text{right}} = 2/3 - (1/9 \times 0) - (8/9 \times 21/32) = 0.083 \)
Training a Decision Tree with Gini Impurity: Second Node

- Do The Same Thing..

Training a Decision Tree with Gini Impurity: 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?

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

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) ...
```
Tuning Decision Trees

• continued

(((entropy, 20,300), 0.9125545571245186)
(((gini, 20,300), 0.9042533162173727)
(((gini, 20,10), 0.8854428754813863)
(((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
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 = 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 - 1
  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]

```scala
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 val cvAccuracy = getMetrics(model, cvData).
      precision ((impurity, depth, bins), (trainAccuracy, cvAccuracy))
    }
```

• 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]

```
(( entropy, 30,300), ( 0.9996922984231909, 0.9438383977425239)) (( entropy, 30,40), ( 0.9994469978654548, 0.93893481368939)) (( gini, 30,300), ( 0.999822874061833, 0.937127912178671)) (( gini, 30,40), ( 0.9995180059216415, 0.9329467634811934)) (( entropy, 20,40), ( 0.9725865867933623, 0.9280773598540899)) (( gini, 20,300), ( 0.9702347139020864, 0.9249630062975326)) (( entropy, 20,300), ( 0.964394392205467, 0.923191307340239)) (( gini, 20,40), ( 0.9679344832334917, 0.9223820503114354)) (( gini, 10,300), ( 0.7953203539213661, 0.7946763481193434)) (( gini, 10,40), ( 0.7880624698753701, 0.7860215423792973)) ...
```

• 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**
  1. Sample $n$ data points with replacement
  2. Train with $n$ data points
  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

- `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 = "2709,125,28,67,23,3224,253,207,61,6094,0,29"
val vector = Vectors.dense(input.split(',').map(_.toDouble))
forest.predict(vector)
```
Part 2. Large Scale Data Storage Systems

Distributed File System

Distributed file systems

• Hadoop vs. Google File System?
Part 1. Large scale data analysis using MapReduce

Distributed File System

Google File System (GFS)

Google Stanford Hardware (1998)
A server room in Council Bluffs, Iowa

• Photo: Google/Connie Zhou

Data Center

• [https://youtube.googleapis.com/v/avP5d16wEp0%26fs=1%26autoplay=1](https://youtube.googleapis.com/v/avP5d16wEp0%26fs=1%26autoplay=1)
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