CS535 BIG DATA

PART 1. BATCH COMPUTING MODELS FOR BIG DATA ANALYTICS

1. DISTRIBUTED MODEL FOR SCALABLE BATCH COMPUTING – DISTRIBUTED FILE SYSTEMS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

- Questions about PA1
  - Send an email to cs535@cs.colostate.edu
- FAQ for PA1 page is available at:
  - http://www.cs.colostate.edu/~cs535/FAQ_PA1.html
- Wikibomb
  - Please see the description of PA1
- No plan for extension
- Difference between original PageRank formula vs. spark example
- Sign up sheet will be sent out today
- Term Project
  - Google computing cluster credit is available
  - Optional

Objectives

- Distributed File System
  - GFS2 (Colossus)
- Large scale data analysis using Spark with case study
  - Decision tree/Random Forest

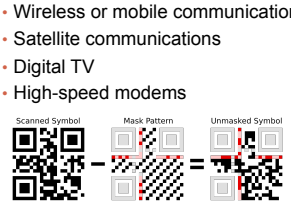
Distributed File Systems

Colossus: Google File System II

Reed-Solomon Codes

Reed-Solomon Codes

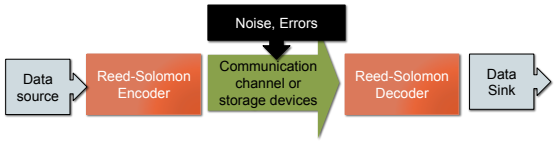
- Block-based error correcting codes
  - Digital communication and storage
- Storage devices (including tape, CD, DVD, barcodes, etc)
- Wireless or mobile communications
- Satellite communications
- Digital TV
- High-speed modems



SOURCE: https://en.wikiversity.org/wiki/Reed–Solomon_codes_for_coders

What does the R-S code do?

- Takes a block of digital data
- Adds extra "redundant" bits
- If an error happens, the R-S decoder processes each block and recovers original data



Reed-Solomon Codes

Quick overview with an example

A Quick Example of the R-S encoding

- 4+2 coding
  - Original files are broken into 4 pieces
  - 2 parity pieces are added
- First piece of data "ABCD", second piece of data "EFGH"…

Original Data

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

A Quick Example of the R-S encoding

- Applying coding matrix

01	00	00	00
00	01	00	00
00	00	01	00
00	00	00	01
1b	1c	12	14
1c	1b	14	12

X

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

=

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P
51	52	53	49
55	56	57	25

A Quick Example of the R-S encoding

- Data loss
  - 2 of 6 rows are lost

01	00	00	00
00	01	00	00
00	00	01	00
00	00	00	01
1b	1c	12	14
1c	1b	14	12

X

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

=

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P
51	52	53	49
55	56	57	25

A Quick Example of the R-S encoding

- Without 2 rows

01	00	00	00
00	01	00	00
1b	1c	12	14
1c	1b	14	12

X

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

=

A	B	C	D
E	F	G	H
51	52	53	49
55	56	57	25

A Quick Example of the R-S encoding

- Multiplying each side with the inverted matrix

01	00	00	00
00	01	00	00
8d	f6	7b	01
f6	8d	01	7b

X

01	00	00	00
00	01	00	00
1b	1c	12	14
1c	1b	14	12

X

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

=

01	00	00	00
00	01	00	00
8d	f6	7b	01
f6	8d	01	7b

X

A	B	C	D
E	F	G	H
51	52	53	49
55	56	57	25

A Quick Example of the R-S encoding

- *The Inverse Matrix and the Coding Matrix Cancel Out*

01	00	00	00
00	01	00	00
8d	f6	7b	01
f6	8d	01	7b

X

01	00	00	00
00	01	00	00
1b	1c	12	14
1c	1b	14	12

X

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

=

01	00	00	00
00	01	00	00
8d	f6	7b	01
f6	8d	01	7b

X

A	B	C	D
E	F	G	H
51	52	53	49
55	56	57	25

A Quick Example of the R-S encoding

- *Reconstructing the Original Data*

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

=

01	00	00	00
00	01	00	00
8d	f6	7b	01
f6	8d	01	7b

X

A	B	C	D
E	F	G	H
51	52	53	49
55	56	57	25

Reed-Solomon Codes

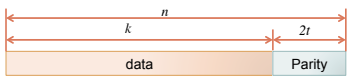
Terminology

Properties of Reed-Solomon codes

- *RS(n,k)* with *s*-bit symbols
  - Encoder takes *k* data symbols (blocks) of *s* bits each
  - Adds parity symbols to make *n* symbol **code word**
  - There are *n-k* parity symbols of *s* bits each
- A Reed-Solomon decoder can correct up to *t* symbols that contain errors in a code word, where $2t = n-k$.
- $t = (n-k)/2$ *for n-k even*
- $t = (n-k-1)/2$ *for n-k odd*

Example

- *RS(255,223)* with 8 bit symbols
  - Each code word contains 255 code word bytes
  - 223 bytes are data and 32 bytes are parity
  - *n=255, k=223, s=8, 2t = 32, t=16*
- *The decoder can correct any 16 symbol errors in the code word*
  - Errors in up to 16 bytes anywhere in the codeword can be automatically corrected.

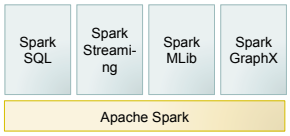


Large scale data analysis using Spark with case study

Spark software stack

Spark stack of libraries

- SQL and DataFrames
- Machine learning
- GraphX
- Spark Streaming



Large scale data analysis using Spark with case study

Predicting Forest Cover with Decision Trees

Regressions and Classifications

- **Regression analysis**
  - A statistical process for estimating the relationships among variables
  - Dependent variable vs. independent variable ("predictors")
  - 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 tomorrows 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`

Vectors and Features

- 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
    - 23F, 56F (23F < 56F)

Training Examples

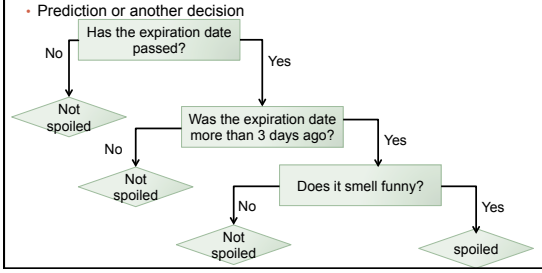
- A learning algorithm needs to train on data in order to make predictions
- Inputs
- Correct outputs (from historical data)
  - "One day, the weather was between 12 and 16 degrees Celsius, with 10% humidity, clear, with no forecast of a cold snap, and the following day the high temperature was **17.2 degrees**"
  - **Target (output)**
    - 17.2 degree
- 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
  - Extended decision tree algorithm

Decision tree

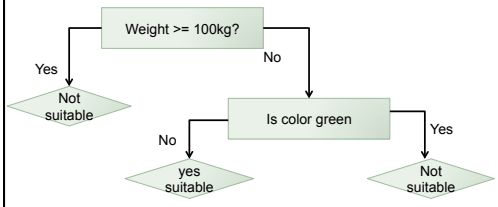
- Spark Mlib's DecisionTree and RandomForest implementation
- Each decision leads to one of two results
  - Prediction or another decision



Example: Finding a good pet

name	Weight (hg)	# of legs	color	Good pet?
Fido	20.5	4	Brown	Y
Mr. Slither	3.1	0	Green	N
Nemo	0.2	0	Tan	Y
Dumbo	1390.8	4	Grey	N
Kitty	12.1	4	Grey	Y
Jim	150.9	2	Tan	N
Millie	0.1	100	Brown	N
McPigeon	1.0	2	Grey	N
Spot	10.0	4	Brown	Y

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
- It includes categorical and numeric features
- 581,012 examples
  - https://archive.ics.uci.edu/ml/datasets/Covertype

Attribute information

Name/ Data Type/ Measurement/ Description

Elevation / **quantitative** /meters / Elevation in meters
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 soltice
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 (4 binary columns) / **qualitative** / 0 (absence) or 1 (presence) / Wilderness area designation
Soil_Type (40 binary columns) / **qualitative** / 0 (absence) or 1 (presence) / Soil Type designation
Cover_Type (7 types) / integer / **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)
  - `LabeledPoint` is only for numeric features
    - 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, rainy or clear`
    - `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 MLib requires input in the form of `LabeledPoint` objects

```
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**

```
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

```
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
    - $PPV = TP/(TP+FP)$
  - Recall is the fraction of all examples that are actually positive that the classifier marked positive
    - $TPR = TP/P=TP/(TP+FN)$
- 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 [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]

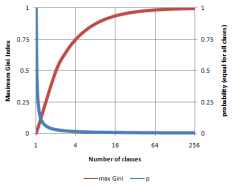
- Good rule should **distinguish examples more meaningfully**
- Example
  - A rule that divides the Covtype set into only 1-3 category and 4-7 category would be a great rule
- A good rule divides the training data's target values into relatively homogeneous or "pure" subsets
- **Minimizing the impurity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

$$I_G(p) = 1 - \sum_{i=1}^{N} p_i^2$$

- If the subset contains only one class
  - The value is 0



http://people.revoledu.com/kardi/tutorial/DecisionTree/how-to-measure-impurity.htm

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

```
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]

```
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]

```
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),( 0.9994469978654548,0.938934581368939))
(( gini, 30,300),( 0.9998622874061833,0.937127912178671))
(( gini, 30,40),( 0.9995180059216415,0.932946763481l934))
(( entropy, 20,40),( 0.9725865867933623,0.9280773598540899))
(( gini, 20,300),( 0.9702347139020864,0.9249630062975326))
(( entropy, 20,300),( 0.9643948392205467,0.9231391307340239))
(( 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%**

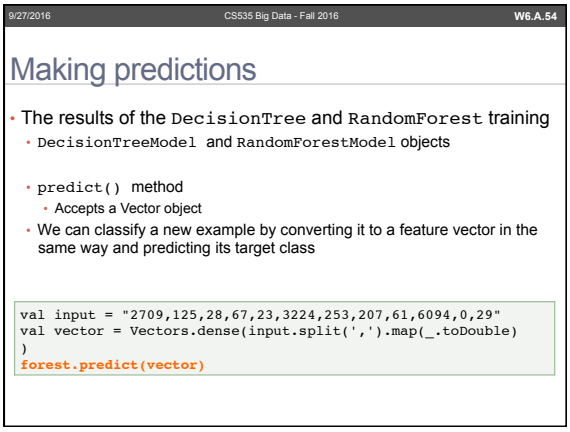
Does decision tree algorithm build the same tree every time?

- 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
  - Trades a bit of **accuracy** for a lot of **speed**
- Decision tree algorithm **won't build the same tree every time**

RandomForest

```
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


9/27/2016
CS535 Big Data - Fall 2016
W6.A.54
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
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)