FAQs

- PA0 has been posted
  - Feb. 6, 5:00PM via Canvas
  - Individual submission (No team submission)

- Accommodation request, honor student
  - Contact me by Jan 26 2018

This Material is Built Based on,


Types of Analytics

- Analytics is a term that is often used interchangeably with
  - Data science
  - Data mining
  - Knowledge discovery

- Predictive analytics
  - A target variable is typically available
  - E.g. linear/logistic regression, decision trees, neural networks, support vector machines

- Descriptive analytics
  - No target variable
  - E.g. Clustering, association rules
Types of Data Sources

- **Transactions**
  - Structured, low-level, detailed information
  - Customer transactions
  - Bank account, card transfers, credit card payments
  - Can be summarized over longer time horizons (e.g. averages, relative trends, Max/Min values)
  - *Unstructured data embedded in text documents*
    - Requires extensive preprocessing

- **Qualitative, expert-based data**
  - Requires expert matter expert (SME) analysis
  - Scientific data

Sampling

- Taking a *subset* of data for analytics
  - Generations hypothesis
  - Model selection
  - Feature selection
  - Speculative process
  - Building analytics model

- **Stratified sampling**
  - Taking samples according to predefined strata
  - e.g. Fraud detection with very skewed (99 percent non-fraud customers, 1 percent fraud customers)
  - Sample should contain the same percentage of fraud customers as in the original data

Types of Data Elements

- **Continuous**
  - Data elements that are defined on an interval that can be limited or unlimited
    - e.g. income, sales, temperature
  - Categorical Nominal
    - Data elements that can only take on a limited set of values with no meaningful ordering between them
  - e.g. marital status, profession, purpose of loan

- **Ordinal**
  - Data elements that can only take on a limited set of values with a meaningful ordering between them
  - e.g. credit rating, age coded as young, middle age and old

- **Binary**
  - Data elements that can only take on two values
  - e.g. having child, allowed to drive

Missing Values

- **Missing values** can occur because of various reasons
  - The information can be non-applicable
  - The information can be undisclosed
  - The information can be unavailable

Missing Values --continued

- **Replace (impute)**
  - Replaces the missing value with a computed/selected value
    - Hot deck: replaces with a randomly selected similar records
    - Cold deck: selects replacement from another dataset
    - Mean substitution: replaces with the mean of that variable for all other cases
    - Regression: predicts missing values of a variable based on other variables.

- **Delete**
  - Deletes observations with lots of missing values
  - Assumes that information is missing at random and has no meaningful interpretation and/or relationship to the target

- **Keep**
  - Missing values can be meaningful
    - e.g. a customer did not disclose the income for current condition

Outliers of Dataset

- **Outliers** are extreme observations that are very dissimilar to the rest of the population
  - Valid observation
    - Salary of boss
  - Invalid observation
    - Age is 100

- **Multivariate outliers**
  - Observations that are outlying in multiple dimensions
    - e.g. Temperature in Fort Collins is 100 degrees but on a midnight in December
Identifying Outliers using Box Plots

- A box plot represents three key quartiles of the data
  - $Q_1$: 25% of the observations have a lower value
  - $Q_2$: 50% of the observations have a lower value
  - $Q_3$: 75% of the observations have a lower value
- The minimum and maximum values are added
- Too far away is now quantified as more than $1.5 \times $IQR $(= Q_3 - Q_1)$

Identifying Outliers using Z-Score

- Measuring how many standard deviations an observation is away from the mean
  - $z = \frac{x - \mu}{\sigma}$ where $\mu$ represents the average of the variable and $\sigma$ its standard deviation
- A practical rule of thumb then defines outliers when the absolute value of the z-score $|z|$ is bigger than 3

Dealing with Outliers

- Treat outliers as missing values
- Popular schemes
  - Truncation: Taking only values that are within the limits
  - Winsorizing: Limiting extreme values to reduce the effect of possible spurious outliers

Standardizing Data

- Scaling variables to a similar range
  - e.g. two variables: education and income
  - Elementary school (1), middle school (2), high school (3), college (4), graduate school (5)
  - Income: 0 ~ $10M
  - When building logistic regression models, the coefficient for education might become very small.
  - Min/Max standardization
    - $X_{new} = \frac{X_{old} - min(X)}{max(X) - min(X)} \cdot (newmax - newmin) + newmin$
    - Where newmin and newmax are the newly imposed maximum and minimum (e.g. 1 and 0)

Standardizing Data. -- continued

- Z-Score based
- Calculate the z-scores
- Decimal scaling
  - $X_{new} = \frac{X_{old}}{10^p}$
  - Dividing by a power of 10
- Standardization is useful for regression-based approaches
- It is not needed for decision trees

Part 2: Introduction

Big Data Analytics
- Big Data Technology Stack
In a nutshell

Data Layer
- Apache HDFS, Amazon AWS's S3, IBM GPFS, Microsoft Azure

Data Processing Layer
- Apache Hadoop MapReduce, Pig, Apache Spark, Cassandra, Storm, Mahout, MLlib

Data Integration Layer
- Apache Flume, Apache Kafka, Apache Sqoop

Operations and Scheduling Layer
- Apache Ambari, Apache Oozie, Apache Zookeeper

Security and Governance
- Apache HDFS, Amazon AWS's S3, IBM GPFS, Microsoft Azure

Part 1. Large Scale Data Analytics
Introduction to MapReduce

What is MapReduce?

[1/2]

What is MapReduce?

MapReduce

- MapReduce is inspired by the concepts of map and reduce in Lisp.

- "Modern" MapReduce
  - Developed within Google as a mechanism for processing large amounts of raw data.
    - Crawler documents or web request logs
    - Distributes these data across thousands of machines
    - Some computations are performed on each CPU with different datasets

This material is developed based on,

  - Download this chapter from the CS435 schedule page


- MapReduce Design Patterns, Donald Miner and Adam Shook, O'Reilly, 2013

This material is developed based on,
MapReduce

- **MapReduce** provides an abstraction that allows engineers to perform simple computations while hiding the details of parallelization, data distribution, load balancing and fault tolerance.

Mapper

- **Mapper** maps input key/value pairs to a set of intermediate key/value pairs.
  - Maps are the individual tasks that transform input records into intermediate records.
  - The transformed intermediate records do not need to be of the same type as the input records.
  - A given input pair may map to zero or many output pairs.
  - The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job.

Reducer

- **Reducer** reduces a set of intermediate values which share a key to a smaller set of values.

Example 1: WordCount

- For text files stored under `usr/joe/wordcount/input`, count the number of occurrences of each word.

Example 1: WordCount

- Run the MapReduce application.
Example 1: WordCount [3/5]

Mappers
1. Read a line
2. Tokenize the string
3. Pass the <key, value> output to the reducer

Reducers
1. Collect <key, value> pairs sharing same key
2. Aggregate total number of occurrences

What do you have to pass from the Mappers?

Example 1: WordCount [4/5]

```java
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```

Example 1: WordCount [5/5]

```java
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values,
    Context context)
    throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

MapReduce Example 2

NCDC data example
- A national climate data center record
- Find the maximum temperature of a year (1900 ~ 1999)

```
0057
320130 # USAF weather station identifier
99999 # NAO weather station identifier
1950001 # observation data
1600 # observation time
+ 51317 # latitude (degrees x 1000)
+ 028783 # longitude (degrees x 1000)
FM-12
+ 0171 # elevation (meters)
9999 #
0020
320 # wind direction (degrees)
1 # quality code
```

The first entries for 1990

```
1990.raw/ 1990 | Read
E1010-9999-1990.gz
E1011-9999-1990.gz
E1012-9999-1990.gz
E1013-9999-1990.gz
E1014-9999-1990.gz
E1015-9999-1990.gz
E1016-9999-1990.gz
E1017-9999-1990.gz
E1018-9999-1990.gz
E1019-9999-1990.gz
E1020-9999-1990.gz
E1021-9999-1990.gz
E1022-9999-1990.gz
E1023-9999-1990.gz
E1024-9999-1990.gz
```
Analyzing the data with Unix Tools (1/2)

• A program for finding the maximum recorded temperature by year from NCDC weather records

```bash
#!/usr/bin/env bash
for year in all/*
do
  echo -ne `basename $ year .gz` 
  gunzip -c $ year |
  awk '{
    temp = substr( $0, 88, 5) + 0;
    q = substr( $0, 93, 1);
    if (temp != 9999 && q ~ /[01459]/ && temp > max)
      max = temp
    }'
  END { print max }
Done
```

Results?

• The complete run for the century took 42 minutes
• To speed up the processing
  • We need to run parts of the program in parallel
  • Process different years in different processes
  • What will be the problems?

Map and Reduce

• MapReduce works by breaking the processing into two phases
  • The map phase
  • The reduce phase
• Each phase has key-value pairs as input and output
• Programmers should specify
  • Types of input/output key-values
  • The map function
  • The reduce function

Analyzing the data with Unix Tools (2/2)

• The script loops through the compressed year files
• Printing the year
• Processing each file using awk
  • Extracts two fields
  • Air temperature and the quality code
• Check if it is greater than the maximum value seen so far

```
<table>
<thead>
<tr>
<th>Year</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1901</td>
<td>317</td>
</tr>
<tr>
<td>1902</td>
<td>244</td>
</tr>
<tr>
<td>1903</td>
<td>289</td>
</tr>
<tr>
<td>1904</td>
<td>256</td>
</tr>
<tr>
<td>1905</td>
<td>283</td>
</tr>
</tbody>
</table>
```

Challenges

• Dividing the work into equal-size pieces
  • Data size per year?
• Combining the results from independent processes
  • Combining results and sorting by year?
• You are still limited by the processing capacity of a single machine (the worst one!)

Visualizing the way the MapReduce works (1/3)

Sample lines of input data
```
006710169099999194902507006.....99999999 + 00001 + 99999999999...
0042011690999991949032412004.....99999999 + 00221 + 99999999999...
0042011690999991949032418004.....99999999 - 00111 - 99999999999...
0042012690999991949032418004.....99999999 + 01111 + 99999999999...
0042012690999991949032418004.....99999999 + 00781 + 99999999999...
```

These lines are presented to the map function as the key-value pairs
```
(8, 006710169099999194902507006.....99999999 + 00001 + 99999999999...)
(106, 0042011690999991949032412004.....99999999 + 00221 + 99999999999...)
(112, 0042011690999991949032418004.....99999999 - 00111 - 99999999999...)
(212, 0042012690999991949032418004.....99999999 + 01111 + 99999999999...)
```

The keys are the file offsets within the file (optional)
Visualizing the way the MapReduce works (2/3)

The map function extracts the year and the air temperature and emit them as its output:

- (1949, 0)
- (1950, 22)
- (1949, 111)
- (1949, 78)

This output key-value pairs will be sorted and grouped by key. Our reduce function will see the following input:

- (1949, [111, 78])
- (1950, [0, 22, -11])

Comparison with other systems

- MPI vs. MapReduce
  - MapReduce tries to collocate the data with the compute node
  - Data access is fast
    - Data is local!

- Volunteer computing vs. MapReduce
  - SETI@home
    - Using donated CPU time

MapReduce Example 3

Retrieve records with “220.4.5.20” as the source

- Apply a condition C to each record in the relation and provide as output only those records that satisfy C:

- 220.4.5.20, 160.33.1.3
- 220.4.5.20, 100.33.1.5
- 160.33.1.3, 100.33.1.5
- 160.33.1.3, 79.45.66.9
- 100.33.1.5, 79.45.66.9
Retrieve records with "220.4.5.20" as the source

Map function:
For each record, check the condition \( \text{source} = "220.4.5.20" \) and produce the key-value pair \((t, t)\).

Reduce function:
For the shuffled pairs \((t, [t, t, \ldots, t])\)
Returns \((t, t)\) as the result of eliminating duplications.

Questions?