Today's topics

- FAQs
- Predictive Analysis
- Linear Regression with Gradient Descent using MapReduce
- Clustering
  - k-Means algorithm with Canopy algorithm using MapReduce

FAQs

- Corrected midterm scores will be available in this week

Predictive Analysis

Fitting Linear Regression Model to a Large Dataset

--Continued

Summary1:

Fitting the linear regression model  [1/2]

- The structure

\[ h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 \ldots \]

How big is the error of the fitted model?

We would like to minimize this error

Summary2:

Fitting the linear regression model  [2/2]

- How big is the error of the fitted model?
  - We would like to minimize this error

- The model that fits the data best
  - The model with the minimum sum of errors on the training data
  - e.g. The sum or mean of the squares of the errors
  - Least squares regression
Summary 3:
Root Mean Squared Error
- Measures the differences between values predicted by model estimator and the values actually observed

\[ \text{RMSD} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2} \]

Summary 4:
Objective function (Cost function)
- For a given training set, how do we pick, or learn, the parameter \( \theta \)?
- Make \( h(x) \) close to \( y \)
- Make your prediction close to the real observation
- We define the objective (cost) function
- Using Mean Squared Error and multiplying \( \frac{1}{2} \) for convenience

\[ J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \]

Summary 5:
Minimization problem
- We have a function \( J(\theta_0, \theta_1) \)
- We want to find \( \min_{\theta_0, \theta_1} J(\theta_0, \theta_1) \)
- Goal: Find parameters to minimize the cost (output of the objective function)
- Outline of our approach:
  - Start with some \( \theta_0, \theta_1 \)
  - Keep changing \( \theta_0, \theta_1 \) to reduce \( J(\theta_0, \theta_1) \) until we end up at a minimum

Summary 6:
Concept of Gradient descent algorithm
Repeat until convergence {
\[ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \]
}(for \( j = 0 \) and \( j = 1 \))

Correct: Simultaneous update
\[ \text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \]
\[ \theta_0 := \text{temp0} \]
\[ \text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \]
\[ \theta_1 := \text{temp1} \]

Incorrect:
\[ \text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \]
\[ \theta_0 := \text{temp0} \]
\[ \text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \]
\[ \theta_1 := \text{temp1} \]
Using Gradient Descent Algorithm for Linear Regression Model

Gradient descent algorithm

Repeat until convergence {
  \( \theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \)
  (for \( j = 0 \) and \( j = 1 \))
}

Linear Regression Model

\[ h_\theta(x) = \theta_0 + \theta_1 x_1 \]
\[ J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \]

Gradient descent for Linear Regression

Repeat until convergence (for \( j = 0 \) and \( j = 1 \))

\[ \theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \]

Update \( \theta_0 \) and \( \theta_1 \) simultaneously

Fitting \( h_\theta(x) \)

\[ J(\theta_0, \theta_1) \]

“Batch” Gradient Descent

- Batch
- Each step of gradient descent uses all of the training example

\[ \theta_0 \leftarrow \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} h_\theta(x_i) - y_i \]
\[ \theta_1 \leftarrow \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)x_i \]
Running with MapReduce

- For the sample size 1,000 (m=1,000)
- Batch gradient descent:
  \[ \theta_0 \leftarrow \theta_0 - \alpha \frac{1}{m} \sum_i (h_0(x^{(i)}) - y^{(i)}) \]
  \[ \theta_1 \leftarrow \theta_1 - \alpha \frac{1}{m} \sum_i (h_0(x^{(i)}) - y^{(i)})x^{(i)} \]
- Using 4 machines

For \( \theta_0 \)

- Step 1. 4 input splits
  \[ \text{temp1} = \sum_{i=1}^{251} (h_0(x^{(i)}) - y^{(i)}) \]
  \[ \text{temp2} = \sum_{i=252}^{501} (h_0(x^{(i)}) - y^{(i)}) \]
  \[ \text{temp3} = \sum_{i=502}^{751} (h_0(x^{(i)}) - y^{(i)}) \]
  \[ \text{temp4} = \sum_{i=752}^{1000} (h_0(x^{(i)}) - y^{(i)}) \]
- Step 2. Calculate temp1 ~ 4
- Step 3. Calculate final results
  \[ \theta_0 \leftarrow \theta_0 - \alpha \frac{1}{1,000} (\text{temp1} + \text{temp2} + \text{temp3} + \text{temp4}) \]

For \( \theta_1 \)

- Step 1. 4 input splits
  \[ \text{temp1} = \sum_{i=1}^{251} (h_0(x^{(i)}) - y^{(i)})x^{(i)} \]
  \[ \text{temp2} = \sum_{i=252}^{501} (h_0(x^{(i)}) - y^{(i)})x^{(i)} \]
  \[ \text{temp3} = \sum_{i=502}^{751} (h_0(x^{(i)}) - y^{(i)})x^{(i)} \]
  \[ \text{temp4} = \sum_{i=752}^{1000} (h_0(x^{(i)}) - y^{(i)})x^{(i)} \]
- Step 2. Calculate temp1 ~ 4
- Step 3. Calculate final results
  \[ \theta_1 \leftarrow \theta_1 - \alpha \frac{1}{1,000} (\text{temp1} + \text{temp2} + \text{temp3} + \text{temp4}) \]

Clustering: Core concept

- Set of N-dimensional vectors
  - Can be in the order of millions
- Group (or cluster) them based on their proximity (or similarity) to each other in an N-dimensional space
- Vectors or objects in a cluster (or group) are more similar to each other than in any other group

Clustering: Applications

- Anomaly detection
- Fraud detection
- Recommendation systems
- Medical imaging
- Market research
- Human genetic clustering
Clustering

$k$-Means Clustering using MapReduce

$k$-Means Clustering

- Unlabeled dataset
- Aims to partition $n$ observations into $k$ clusters
- Each observation belongs to the cluster with the nearest mean

Concept: $k$-Means Clustering (1/4)

Concept: $k$-Means Clustering (2/4)

Concept: $k$-Means Clustering (3/4)

Concept: $k$-Means Clustering (4/4)
k-Means algorithm

Input
- $k$ (number of clusters)
- Training set $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$

repeat{ $c^{(i)}$ = index (from $K$) of cluster centroid closest to $x^{(i)}$
for $k = 1$ to $K$ $\mu_k$ = average (mean) of points assigned to cluster $k$
}

Cost function
- The objective is to find:
  $$ \arg\min_k \sum_{i=1}^{m} \sum_{x \in S_i} (x - \mu_k)^2 $$
- Where $\mu_k$ is the mean of points in $S_i$

How to choose the number of clusters
- Value $k$ in the algorithm

Choosing the value $K$

Elbow Method
Choosing the value $K$ (2/2)

Distance Measures
- Euclidean Distance
- Manhattan Distance
- Cosine Distance
- Hamming Distance
- Jaccard Dissimilarity
- Edit Distance
- Smith Waterman Similarity
- Image Distance
- Etc.

$k$-Means using MapReduce
- Computing the Euclidean distance between the sample vectors and the centroids can be parallelized
  - By splitting the data into individual subgroups and clustering samples in each subgroup separately
  - By the mapper
- Recalculating new centroid vectors
  - Divide the sample vectors into subgroups
  - Compute the sum of vectors in each subgroup in parallel
  - Reducer will add up the partial sums and compute the new centroids
- Question
  - How much of data should be transferred for this step?
  - How do we effectively parallelize this process?

Steps
1. Get data into a form you can use
2. Picking Canopy Centers (MR job)
3. Assign Data Points to Canopies (MR job)
4. Pick $k$-Means cluster Centers
5. $k$-Means algorithm (MR jobs)
   - Iterate

Canopy clustering algorithm
- Unsupervised pre-clustering algorithm
- Often used as preprocessing step for $k$-Means or Hierarchical clustering
- Major goal of this algorithm is to speed up clustering operations on large datasets
General Canopy Clustering Algorithm

- Using two thresholds $T_1$ (the loose distance) and $T_2$ (the tight distance), where $T_1 > T_2$

1. Begin with the set of data points to be clustered
2. Remove a point from the set, beginning a new "canopy"
3. For each point left in the set, assign it to the new canopy if the distance is less than the loose distance $T_1$
4. If the distance of the point is less than the tight distance $T_2$, remove it from the original set
5. Repeat steps 2-3-4, until there are no more data points in the set to cluster

Canopy Clustering using MapReduce

- Each mapper performs canopy clustering on the points in its input set
  - Non-overlapping sampled points

- Reducer clusters the canopy centers to produce the final canopy centers
  - Performs canopy clustering over the canopy centers

Generating Input data

- Generate samples
  - green and red

Generating Canopy centers (Red)

- For the red data performed in a Mapper

Generating Canopy centers (Green)

- For the green data performed in a Mapper
Collecting Canopy Centers (Reducer)

Perform Canopy Clustering (Reducer)

Final Canopy centers

Creating Canopies

- Is this good enough?

Running Canopies using MapReduce

- Each mapper performs canopy clustering on the points in its input set and outputs its canopies' centers

- The reducer clusters the canopy centers to produce the final canopy centers

- You can use these Canopy clustering centroids for input of k-Means

- What if the centroid includes multiple canopies?
  - Your computation should consider the merging and selection process

MapReduce Design Patterns

Input and Output Patterns
This material is built based on,

- MapReduce Design Patterns
  - Building Effective Algorithms and Analytics for Hadoop and Other Systems
  - By Donald Miner, Adam Shook
  - November, 2012

Customizing input and output

- Do we always want to load or store data the way Hadoop MR does out of the box?
  - Injecting data from original source without storing data in HDFS
  - Feeding the MapReduce output to the next process

Patterns discussed in this section

1. Generating data
2. External source input
3. Partition pruning

Modify the way data is loaded on disk

- Approach 1: Configuring how contiguous chunks of input are generated from blocks in HDFS
  - InputFormat

- Approach 2: Configuring how records appear in the map phase
  - RecordReader

Modify the way data is stored on disk

- Approach 1: Configuring how contiguous chunks of output are generated from blocks in HDFS
  - OutputFormat

- Approach 2: Configuring how records are stored after the map phase
  - RecordWriter
Roles of InputFormat in Hadoop

1. Validate the input configuration for the job (i.e., checking that the data is there).
2. Split the input blocks and files into logical chunks of type InputSplit, each of which is assigned to a map task for processing.
3. Create the RecordReader implementation to be used to create key/value pairs from the raw InputSplit. These pairs are sent one by one to their mapper.

Accessing your input file in MapReduce

STEP 1. Validates the input for the job by checking whether all of the input paths exist
STEP 2. Splits each input file logically based on the total size of the file in bytes
   - Block size is the upper bound
   - E.g. 160MB in HDFS will generates three blocks
     - 2 x 64MB and 1x38MB
STEP 3. Each map task will be assigned exactly one of these input splits
STEP 4. RecordReader will generate key/value pairs for Mapper input

Methods of the InputFormat abstract

- `getSplits()`: retrieves the configured input using the JobContext object
  - returns a List of InputSplit objects
  - `getLocations()` of InputSplit returns the list of hostnames where the input split is located
    - This provides clue to the system to determine where to process the map task
    - Good place to throw any necessary exceptions
- `createRecordReader()`: Called by framework and generates RecordReader

RecordReader (1/2)

- Generates key/value pairs
- Fixing boundaries
- Input split boundary might not exactly match the record boundary
  - E.g. TextInputFormat reads text files using a LineRecordReader to create key/value pairs
  - Will the chunk of bytes for each input split be lined up with a new line character, to mark the line for the LineRecordReader?
  - Those bits that are stored on a different node are streamed from a DataNode hosting the block
    - Handled by the FSDataInputStream class

RecordReader (2/2)

- Reads Bytes from the input source
- Generates WritableComparable key and Writable value
  - An object-oriented way to present information to a mapper
- Example
  - TextInputFormat grabs each line
    - “<?xml version=”1.0”?>” and “<quiz>” will be injected to the different Mappers
    - Customized RecordReader can read lines after the input split boundary
    - Each RecordReader should starts at the beginning of an XML element

Methods of the RecordReader (abstract)

- `initialize()`
- `getCurrentKey()` and `getCurrentValue()`
- `nextKeyValue()`
- `getProgress()`
- `close()`
Schema on read
- `InputSplit` represents a byte-oriented view of the split
- `RecordReader` prepares data for a mapper
  - Only the `RecordReader` maintains the schema

OutputFormat
- Similar to an input format
- Tasks
  - Validate the output configuration for the job
  - Create the `RecordWriter` implementation that will write the output of the job
- `FileOutputFormat`
  - File based output
  - Most output from MapReduce job is written to HDFS
- `TextOutputFormat` (extended `FileOutputFormat`)
  - Stores key/value pairs to HDFS at a configured output directory with a tab delimiter
  - Validates the output file directory

Storing data in an External DB
- MapReduce job is not restricted to storing data to HDFS
- MapReduce can do a parallel bulk write
  - Your storage should be able to handle the large number of connections from the many tasks
- E.g. `DBOutputFormat<K DBWritable, V>`
  - Objects that read from/written to a database should implement `DBWritable`
  - If we have the following table in the database:

  ```
  CREATE TABLE MyTable (
      counter INTEGER NOT NULL,
      timestamp BIGINT NOT NULL,
  );
  ```

Writing your output to a DB (1/2)
```java
public class MyWritable implements Writable, DBWritable {
    // Some data
    private int counter;
    private long timestamp;

    // Writable#write() implementation
    public void write(DataOutput out) throws IOException {
        out.writeInt(counter);
        out.writeLong(timestamp);
    }

    // Writable#readFields() implementation
    public void readFields(DataInput in) throws IOException {
        counter = in.readInt();
        timestamp = in.readLong();
    }

    // PreparedStatement implementation
    public void write(PreparedStatement statement) throws SQLException {
        statement.setInt(1, counter);
        statement.setLong(2, timestamp);
    }

    public void readFields(ResultSet resultSet) throws SQLException {
        counter = resultSet.getInt(1);
        timestamp = resultSet.getLong(2);
    }
}
```

**** `PreparedStatement` is an object that represents a precompiled SQL statement.

I/O Pattern 1: Generating Data
- Generates a lot of data from scratch
  - This pattern does not load data
- Use cases:
  - Generating random data
  - Generating artificial data as part of a benchmark
  - TeraGen/TeraSort and DFSIO
- This pattern is map-only
Structure

- The InputFormat creates the fake splits from nothing
- The RecordReader takes its fake split and generates random records
- The IdentifyMapper is used to just write the data out as it comes in

Identity Mapper

- Implements Mapper<K, V, K, V>
- conf.setMapperClass(IdentityMapper.class);
- Identity Mapper takes input key/value pair and returns without any processing
- Other implementations of Mapper
  - InverseMapper, TokenCountMapper, ChainMapper... Etc.

Identity Reducer

- Implements Reducer<K, V, K, V>
- Performs no reduction, writing all input values directly to the output.
- What is the difference between Identity Reducer and 0 reducer?
  - Identity reducer still sort and shuffle output data from the mappers
  - No aggregation

I/O Pattern 1: Generating Data: Example

- Goal
  - Generates random StackOverflow data
  - Take a list of 1,000 words and make random blurbs

Driver code

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    int numMapTasks = Integer.parseInt(args[0]);
    int numRecordsPerTask = Integer.parseInt(args[1]);
    Path wordList = new Path(args[2]);
    Path outputDir = new Path(args[3]);
    Job job = new Job(conf, "RandomDataGenerationDriver");
    job.setJarByClass(RandomDataGenerationDriver.class);
    job.setNumReduceTasks(0);
    job.setInputFormatClass(RandomStackOverflowInputFormat.class);
    RandomStackOverflowInputFormat.setNumMapTasks(job, numMapTasks);
    RandomStackOverflowInputFormat.setNumRecordPerTask(job, numRecordsPerTask);
    RandomStackOverflowInputFormat.setInputPath(job, wordList);
    TextOutputFormat.setOutputPath(job, outputDir);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(NullWritable.class);
    System.exit(job.waitForCompletion(true) ? 0 : 2);
}
```
InputSplit Code

```java
public static class FakeInputSplit extends InputSplit implements Writable {
    public void readFields(DataInput arg0) throws IOException {
    }
    public void write(DataOutput arg0) throws IOException {
    }
    public long getLength() throws IOException, InterruptedException {
        return 0;
    }
    public String[] getLocations() throws IOException, InterruptedException {
        return new String[0];
    }
}
```

InputFormat code

```java
public static class RandomStackOverflowInputFormat extends InputFormat<Text, NullWritable> {
    public static final String NUM_MAP_TASKS = "random.generator.map.tasks";
    public static final String NUM_RECORDS_PER_TASK = "random.generator.num.records.per.map.task";
    public static final String RANDOM_WORD_LIST = "random.generator.random.word.file";
    public List<InputSplit> getSplits(JobContext job) throws IOException {
        // Get the number of map tasks configured for int
        int numSplits = job.getConfiguration().getInt(NUM_MAP_TASKS, -1);
        // Create a number of input splits equivalent to the number of tasks
        ArrayList<InputSplit> splits = new ArrayList<InputSplit>();
        for (int i = 0; i < numSplits; i++) {
            splits.add(new FakeInputSplit());
        }
        return splits;
    }
    public static void setNumMapTasks(Job job, int i) {
        job.getConfiguration().setInt(NUM_MAP_TASKS, i);
    }
    public static void setNumRecordPerTask(Job job, int i) {
        job.getConfiguration().setInt(NUM_RECORDS_PER_TASK, i);
    }
    public static void setRandomWordList(Job job, Path file) {
        DistributedCache.addCacheFile(file.toUri(), job.getConfiguration());
    }
}
```

I/O Pattern 2: External Source Output

- Writing MapReduce output to a nonnative location
- In a MapReduce approach, the data is written out in parallel

The Structure of the external source output pattern

- The OutputFormat verifies the output specification of the job configuration prior to job submission
- The RecordWriter writes all key/value pairs to the external source
Example

- Writing the results to a number of Redis instances
  - Redis is an open-source, in-memory, key-value store
  - Redis provides Jedis (Java client of Redis)
  - A Redis hash is a map between string fields and string values
    - Similar to a Java hashmap

```
// For each IOException
public void 
// Write the key/ value pair
jedisMap.size
j = jedisMap.get(substr)
// Get the RedisHosts instance
get(REDIS_HOSTS_CONF)
// throw new IOException();
// REDIS_HOSTS_CONF + " is not set in configuration.");
if (hosts == null | |
String hosts =
// Map an integer 0-15 to the instance
int i = 0;
for (String host : hosts.split(',')) {
  Jedis jedis = new Jedis(host);
  jedis.connect();
  jedisMap.put(i, jedis); + + i;
}
```

```
public static class RedisHashRecordWriter extends RecordWriter < Text, Text > {
  // Create a connection to Redis for each host
  // Map an integer 0-15 to the instance
  int i = 0;
  String host = hosts.split(',')[i] {
    Jedis jedis = new Jedis(host);
    jedis.connect();
    jedisMap.put(i, jedis); + + i;
  }
}
```

```
public static final String REDIS_HOSTS_CONF =
mapred.redishashoutputformat.hosts;
public static void setRedisHosts( Job job, String hosts) {
  job.getConfiguration(). set( REDIS_HOSTS_CONF, hosts);
}
```

```
public static class RedisHashOutputFormat extends OutputFormat < Text, Text > {
  public static final String REDIS_HASH_KEY_CONF =
mapred.redishashoutputformat.key;
  public static void setRedisHashKey( Job job, String hashKey) {
    job.getConfiguration(). set( REDIS_HASH_KEY_CONF, hashKey);
  }
```

```
public static void setRedisHashKey( Job job, String hashKey) {
  job.getConfiguration(). set( REDIS_HASH_KEY_CONF, hashKey);
} public RecordWriter < Text, Text >
getRecordWriter( TaskAttemptContext job) throw IOException,
InterruptedException {
  return new RedisHashRecordWriter( job.getConfiguration());
  get( REDIS_HASH_KEY_CONF), job.getConfiguration().
  get( REDIS_HOSTS_CONF));
}
```

```
public static class RedisHashOutputFormat extends OutputFormat < Text, Text > {
  public static final String REDIS_HASH_KEY_CONF =
mapred.redishashoutputformat.key;
  public static void setRedisHashKey( Job job, String hashKey) {
    job.getConfiguration(). set( REDIS_HASH_KEY_CONF, hashKey);
  }
```

```
public static void setRedisHashKey( Job job, String hashKey) {
  job.getConfiguration(). set( REDIS_HASH_KEY_CONF, hashKey);
} public RecordWriter < Text, Text >
getRecordWriter( TaskAttemptContext job) throw IOException,
InterruptedException {
  return new RedisHashRecordWriter( job.getConfiguration());
  get( REDIS_HASH_KEY_CONF), job.getConfiguration().
  get( REDIS_HOSTS_CONF));
}
```

```
public static void checkOutputSpecs( JobContext job) throws IOException {
  String reputations =
// Get the Jedis instance that this key/ value pair
j = jedisMap.get(substr)
// Write the key/ value pair
j.hash( hashKey, key, toString(), value, toString());
public void close( TaskAttemptContext context) throws IOException,
InterruptedException {
  // For each jedis instance, disconnect it for (Jedis jedis : jedisMap.values());
  jedis.disconnect();
}
```

```
public static class RedisHashRecordWriter extends RecordWriter < Text, Text > {
  // code in next section }
}
```

```
public void write( Text key, Text value) throws IOException,
InterruptedException {
  // Get the Jedis instance that this key/ value pair
j = jedisMap.get(Math.abs(key.hashCode()) %
jedisMap.size());
  // Write the key/ value pair
j.hash( hashKey, key, toString(), value, toString());
  public void close( TaskAttemptContext context) throws IOException,
InterruptedException {
    // For each jedis instance, disconnect it for (Jedis jedis : jedisMap.values());
    jedis.disconnect();
  }
```

```
private static class RedisHashRecordWriter extends RecordWriter < Text, Text > {
  private String hashKey =
// Create a connection to Redis for each host
// Map an integer 0-15 to the instance
int i = 0;
for (String host : hosts.split(',')) {
  Jedis jedis = new Jedis(host);
  jedis.connect();
  jedisMap.put(i, jedis); + + i;
}
```

```
private static class RedisHashRecordWriter extends RecordWriter < Text, Text > {
  private String hashKey =
// Create a connection to Redis for each host
// Map an integer 0-15 to the instance
int i = 0;
for (String host : hosts.split(',')) {
  Jedis jedis = new Jedis(host);
  jedis.connect();
  jedisMap.put(i, jedis); + + i;
}
```
Driver Code

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String hosts = args[1];
    Job job = new Job(conf); job.setJarByClass(RedisOutputDriver.class);
    job.setInputFormatClass(RedisLastAccessInputFormat.class);
    job.setOutputValueClass(Text.class);
    job.setOutputKeyClass(Text.class);
    job.setNumReduceTasks(0);
    job.setOutputFormatClass(RedisHashOutputFormat.class);
    job.setInputPathClass(TextInputFormat.class);
    job.setOutputPaths(job.getConfiguration().get(REDIS_SELECTED_MONTHS_CONF, "files from being loaded into MapReduce based on the name of the file.
    Partition Pruning

- Configures the way the framework picks input splits and drops files from being loaded into MapReduce based on the name of the file.
- Partitions data by a predetermined value
- Use cases
  - Organizing your data based on your analysis patterns
  - Change analytics? Or, change data input format?

Writing InputSplit

```java
public static class RedisLastAccessInputSplit extends InputSplit {
    private String location = null;
    private List<String> hashKeys = new ArrayList<String>();
    public RedisLastAccessInputSplit() {
        // Default constructor for reflection
    }
    public RedisLastAccessInputSplit(String redisHost) {
        InputFormat inputFormat = new RedisLastAccessInputFormat(redisHost);
        String key = inputFormat.getHashKey();
        hashKeys.add(key);
        public void removeHashKey(String key) {
            hashKeys.remove(key);
            public List<String> getHashKeys() {
                return hashKeys;
            }
        }
    }
```
continued

```java
public List < InputSplit > getSplits( JobContext job) throws IOException {
    String months = job.getConfiguration().
        get( REDIS_SELECTED_MONTHS_CONF);
    if (months == null || months.isEmpty()) throw
        new IOException(REDIS_SELECTED_MONTHS_CONF + " is
        null or empty.");

    // Create input splits from the input months
    HashMap < String,
        RedisLastAccessInputSplit > instanceToSplitMap = new HashMap < String,
            RedisLastAccessInputSplit > ();
    for (String month :
        months.split(",")) {
        String host = MONTH_TO_INST_MAP.get( month);
        RedisLastAccessInputSplit split =
            instanceToSplitMap.get( host);
        if (split == null) {
            split = new RedisLastAccessInputSplit( host);
            split.addHashKey( month);
            instanceToSplitMap.put( host, split);
        } else {
            split.addHashKey(month);
        }
    }
    return new ArrayList < InputSplit > (instanceToSplitMap.values());
}
```

continued

```java
public static class RedisLastAccessRecordReader extends RecordReader
    < RedisKey, Text > {
    …
}
```