FAQs

- Programming Assignment 2
  - Content Based Authorship Detection using TF/IDF Scores and Cosine Similarity
  - This assignment will require more than 40 hours of work
  - March 23 5:00PM via Canvas
  - Individual submission
  - Please do not submit your output file!

Assignment 2

- Term $i$ appears in $n_i$ sub-collections within the corpus
- Inverted Document Frequency
  - $IDF_i = \log_{N/n_i}$
  - where, $N$ is the total number of sub-collections (number of authors)
- TF-IDF
  - $TF_i \times IDF_i$

Assignment 2

- Author Attribute Vector (AAV)
  - $AAV_m = (TF.IDF_{word1}, TF.IDF_{word2}, TF.IDF_{word3}, ..., TF.IDF_{wordn})$
- CosSimilarity
Assignment 2 [4/5]

- Off-Line computing
  - This phase includes multiple steps:
    - You should calculate the TF, IDF, and TF-IDF values for all terms for all of the sub-collections in your corpus. You are required to use MapReduce(s) for this step. Custom implementations without using MapReduce is disallowed.
    - You should create the AAVs (author attribute vectors) for every author.
    - You should store the results (author attribute vectors) in a HDFS file.

Command-Line software
- Read a document with unknown authorship and create an attribute vector for it. Make sure the dimensionality of this vector should be identical to the one used in the off-line computing (section 2.1).
- You should calculate the Cosine Similarity between the author attribute vector for this document and all of the author attribute vectors calculated in the section 2.1, and select top 10 authors. You are required to use MapReduce for this step. Also, you are required to use a Combiner for this step.

Help session
- March 3, 3:00PM – 3:50PM
- CSB130

Link Analysis
Link spam

Architecture of a Spam Farm

- **Spam Farm**
  - A collection of pages whose purpose is to increase the PageRank of a certain page or pages

- From the point of view of the view of spammer, the Web is divided into two parts
  - Inaccessible pages
    - The pages that the spammer cannot affect
  - Accessible pages
    - Those pages that, while they are not controlled by the spammer, can be affected by the spammer

Understanding Spam Farm (1/2)

- Setting the links to the target page
  - Without link from outside, the spam farm is not useful
  - e.g. Blogs or news papers
  - Comments like “I agree. Please see my article at www.mySpamFarm.com”
Understanding Spam Farm (2/2)
- There is one page $t$, the target page
  - Spammer attempts to place as much PageRank as possible
- There are a large number of $m$ of supporting pages
  - Accumulate the portion of the PageRank that is distributed equally to all pages
  - The fraction $1 - \beta$ of the PageRank that represents surfers going to a random page
  - Prevent the PageRank of $t$ from being lost
  - Note that all of the supporting pages links only to $t$

Analysis of a Spam Farm (1/6)
- A taxation parameter $\beta$
  - The fraction of a page’s PageRank that gets distributed to its successors at the next round
- Let there be,
  - $n$ pages on the Web in total
  - A target page $t$
  - $m$ supporting pages

Analysis of a Spam Farm (2/6)
- Let $x$ be the amount of PageRank contributed by the accessible pages
  - $x$ is the sum over all accessible page $p$ with a link to $t$, of the PageRank of $p$ times $\beta$ divided by the number of successors of $p$
  - Finally, let $y$ be the unknown PageRank of $t$

Analysis of a Spam Farm (3/6)
- The PageRank of each supporting page
  - $\beta y / m + (1 - \beta) / n$
- First term represents the contribution from $t$
  - $\beta y$ is distributed to $t$’s successors
- Second term is the supporting page’s share of the fraction $1 - \beta$ of the PageRank that is divided equally among all pages on the Web

Analysis of a Spam Farm (4/6)
- PageRank of $y$ of target page $t$ is $(1) + (2) + (3)$
  1. Contribution $x$ from outside
  2. $\beta$ times the PageRank of every supporting page
    - $\frac{\text{PageRank of each supporting page}}{m}$
  3. $(1 - \beta) / n$, the share of the fraction $1 - \beta$ of the PageRank that belongs to $t$
    - This amount is negligible

Analysis of a Spam Farm (5/6)
- From (1) and (2),
  $$y = x + \beta \left( \frac{\beta y}{m} + \frac{1 - \beta}{n} \right) = x + \beta y + \beta(1 - \beta) \frac{m}{n}$$
  $$y = x \left( \frac{1 - \beta}{1 - \beta} + \frac{\beta}{n} \right)$$
  $$y = x \left( \frac{\beta}{1 - \beta} + \frac{\beta}{n} \right)$$
- Where
  $$c = \frac{\beta(1 - \beta)}{(1 - \beta^2)} = \frac{\beta}{1 + \beta}$$
Analysis of a Spam Farm (6/6)

- If we choose $\beta = 0.85$, then $1/(1 - \beta^2) = 3.6$
- $c = \beta/(1 + \beta) = 0.46$

- The structure has amplified the external PageRank contribution by 360%.
- Also, it obtained an amount of PageRank that is 46% of the fraction of the Web, $m/n$, that is in the spam farm.

Combatting Link Spam

- Detecting and eliminating link spam have been critical for search engines.
  - Just as it was critical to eliminate term spam in the previous decade.
- Detecting particular structures:
  - Spam farm
    - One page links to a very large number of pages.
    - Each of which links back to it.

TrustRank

- TrustRank is a topic-sensitive PageRank.
  - "topic" is a set of pages believed to be trustworthy (not spam).
- Develop a suitable teleport set of trustworthy pages.
  - Let humans examine a set of pages and decide which of them are trustworthy.
- Pick a domain whose membership is controlled.
  - University pages.
  - .mil, .gov.

Calculating TrustRank (1/2)

- Then the topic-sensitive PageRank for $S$ is the limit of the iteration,

$$v' = \beta M v + (1 - \beta) e_S / |S|$$

- $M$ is the transition matrix of the Web, and $|S|$ is the size of set $S$.
Calculating TrustRank (2/2)
• Suppose we use $\beta=0.8$, and our trust rank is represented by the teleport set (trustworthy pages) $S=(B,D)$

\[
\beta M = \begin{bmatrix}
0 & 2/5 & 4/15 & 0 \\
0 & 0 & 0 & 2/5 \\
4/15 & 0 & 0 & 2/5 \\
4/15 & 2/5 & 0 & 0
\end{bmatrix}
\]

\[
v' = \left(\frac{0}{1/10} \right) v + \left(\frac{1}{1/10} \right)
\]

\[
\begin{bmatrix}
4/15 & 0 & 0 & 2/5 \\
4/15 & 0 & 0 & 2/5 \\
4/15 & 2/5 & 0 & 0 \\
0 & 2/5 & 5/4 & 0
\end{bmatrix}
\]

B and D get a higher PageRank than before

Spam Mass
• Measures the fraction of its PageRank that comes from spam for each page

For an arbitrary page $p$,
- PageRank $r$
  - Computing ordinary PageRank
  - TrustRank $t$
  - Computing the TrustRank based on some teleport set of trustworthy pages
  - The spam mass
    - $(r - t)/r$

A negative or small positive spam mass
- $p$ is probably not a spam page

Page with high spam mass score
- Should be eliminated

Example
• Suppose that both the PageRank and TrustRank were computed
  - Teleport set was page B and D
  - Which nodes are not the link spams?
  - Is there any link spam?

<table>
<thead>
<tr>
<th>Web Page</th>
<th>PageRank</th>
<th>TrustRank</th>
<th>SpamMass</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3/9</td>
<td>54/210</td>
<td>0.229</td>
</tr>
<tr>
<td>B</td>
<td>2/9</td>
<td>59/210</td>
<td>-0.264</td>
</tr>
<tr>
<td>C</td>
<td>2/9</td>
<td>38/210</td>
<td>0.186</td>
</tr>
<tr>
<td>D</td>
<td>2/9</td>
<td>59/210</td>
<td>-0.264</td>
</tr>
</tbody>
</table>

MapReduce Design Patterns
Filtering 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

Filtering pattern

- Providing an abstract of existing data
- Many data filtering do not require the "reduce" part of MapReduce
  - It does not produce an aggregation
- Known uses
  - Tracking a thread of events
  - Distributed grep
  - Data cleaning
  - Closer view of data
  - Simple random sampling
  - Removing low scoring data

Filtering patterns covered in this class

1. Simple Random Sampling
2. Bloom filter
3. Top 10
4. Distinct

The structure of the simple filter pattern

- Each record has an equal probability of being selected
- Useful for sizing down a data set
- For representative analysis
Writing a Simple Random Sampling filter

```java
public static class SRSMapper
extends Mapper < Object, Text, NullWritable, Text > {
    private Random rands = new Random();
    private Double percentage;

    protected void setup(Context context) throws IOException,
            InterruptedException {
        // Retrieve the percentage that is passed in via the configuration
        // like this: conf.set("filter_percentage", .5);
        // for .5%
        String strPercentage = context.getConfiguration().get("filter_percentage");
        percentage = Double.parseDouble(strPercentage) / 100.0;
    }

    public void map( Object key, Text value, Context context) throws
            IOException, InterruptedException {
        if (rands.nextDouble() < percentage) {
            context.write(NullWritable.get(), value);
        }
    }
}
```

Filtering Pattern 2. Bloom Filter

- Checking the membership of a set
- Known uses
  - Removing most of the non-membership values
  - Prefiltering a data set for an expensive set membership check

What is a Bloom Filter?

- Burton Howard Bloom in 1970
- Probabilistic data structure used to test whether a member is an element of a set
- Strong space advantage

Building a Bloom filter

- \( m \): The number of bits in the filter
- \( n \): The number of members in the set
- \( p \): The desired false positive rate
- \( k \): The number of different hash functions used to map some element to one of the \( m \) bits with a uniform random distribution
Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{5, 10, 15\} \)
- \( k = 3 \)
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)
  - \( h_1(5) = 7 \), \( h_2(5) = 5 \), \( h_3(5) = 5 \)

<table>
<thead>
<tr>
<th>Initial Bloom filter</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>After ( h_1(5) = 7 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_2(5) = 5 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>After ( h_3(5) = 5 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{5, 10, 15\} \)
- \( k = 3 \)
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)
  - \( h_1(10) = 6 \), \( h_2(10) = 7 \), \( h_3(10) = 2 \)

| After \( h_1(10) = 6 \) | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| After \( h_2(10) = 7 \) | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| After \( h_3(10) = 2 \) | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |

Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{5, 10, 15\} \)
- \( k = 3 \)
  - \( h_1(x) = 3x \mod 8 \)
  - \( h_2(x) = (2x + 3) \mod 8 \)
  - \( h_3(x) = x \mod 8 \)
  - \( h_1(15) = 6 \), \( h_2(15) = 7 \), \( h_3(15) = 7 \)

| After \( h_1(10) = 6 \) | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| After \( h_2(15) = 1 \) | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| After \( h_3(15) = 7 \) | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

Applying a Bloom filter

- Is 5 part of set \( T \)?
- \( h_1(5), h_2(5), h_3(5) \)'s bits are 1
- 5 is probably a part of set \( T \)

Check \( h_1(5) = 7 \) | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
Check \( h_2(5) = 5 \) | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
Check \( h_3(5) = 5 \) | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

Applying a Bloom filter

- Is 8 part of set \( T \)?
- \( h_1(8), h_2(8), h_3(8) \)
- 8 is NOT a part of set \( T \)

Check \( h_1(8) = 0 \) | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
Check \( h_2(8) = 3 \) | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
Check \( h_3(8) = 0 \) | 1 | 1 | 0 | 0 | 1 | 0 | 0 |

Applying a Bloom filter

- Is 9 part of set \( T \)?
- \( h_1(9), h_2(9), h_3(9) \)
- 9 is NOT a part of set \( T \)

Check \( h_1(9) = 3 \) | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
Check \( h_2(9) = 5 \) | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
Check \( h_3(9) = 1 \) | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
Applying a Bloom filter

- Is 7 part of set T?
  - $h_1(7), h_2(7), h_3(7)$th bits are 1
  - 7 is probably a part of set T

Check $h_1(7) = 711100100$
Check $h_2(7) = 111100110$
Check $h_3(7) = 711100110$

False positive rate (1/2)

$$fpr = (1 - \left(1 - \frac{1}{m}\right)^n)^k = \left(1 - e^{-kn/m}\right)^k$$

$m$ = number of bits in the filter
$n$ = number of elements
$k$ = number of hashing functions

Hash functions

- $k$ Hash functions
  - Uniform random distribution in $[1..m)$

- Cryptographic hash functions
  - MD5, SHA-1, SHA-256, Tiger, Whirlpool...

- Murmur Hashes (non-cryptographic)

False positive rate (2/2)

- A bloom filter with an optimal value for $k$ and 1% error rate only needs 9.6 bits per key.
- Add 4.8 bits/key and the error rate decreases by 10 times
- 10,000 words with 1% error rate and 7 hash functions
  - ~12KB of memory
- 10,000 words with 0.1% error rate and 11 hash functions
  - ~18KB of memory

Use cases

- Representing a very large dataset
- Reduce queries to external database
- Google BigTable

 Downsides  

- **False positive rate**

  - **Hard to remove elements** from a Bloom filter set
    - Setting bits to zero
    - Often more than one element hashed to a particular bits
    - Use a Counting Bloom filter
      - Instead of bit, it stores count of occurrences
      - Requires more memory
Building and running Bloom Filtering

Building Bloom filter (Non MapReduce or with MapReduce)

Input
Split
Bloom Filter
Building
Output
Split

Writing a Bloom filter

public class BloomFilterDriver {
  public static void main(String[] args) throws Exception {
    // Parse command line arguments
    Path inputpath = new Path(args[0]);
    int numMembers = Integer.parseInt(args[1]);
    float falsePosRate = Float.parseFloat(args[2]);
    Path bfpath = new Path(args[3]);
    // Calculate our vector size and optimal K value based on approximations
    int vectorSize = getOptimalBloomFilterSize(numMembers, falsePosRate);
    int nbHash = getOptimalK(numMembers, vectorSize);
    // Create new Bloom filter
    BloomFilter filter = new BloomFilter(vectorSize, nbHash, Hash.MURMUR_HASH);
    // Open file for read
    String line = null;
    FileSystem fs = FileSystem.get(new Configuration());
    for (FileStatus status : fs.listStatus(inputpath)) {
      BufferedReader rdr =
        new BufferedReader(new InputStreamReader(new GZIPInputStream(fs.open(status.getPath()))));
      System.out.println("Reading " + status.getPath());
      while ((line = rdr.readLine()) != null) {
        filter.add(new Key(line.getBytes()));
      }
      rdr.close();
    }
    System.out.println("Trained Bloom filter with ", numElements + " entries.");
    System.out.println("Serializing Bloom filter to HDFS at " + bfpath);
    FSDataOutputStream strm = fs.create(bfpath);
    filter.write(strm);
    strm.flush();
    strm.close();
    fs.close();
    System.exit(0);
  }
}

Writing a Bloom Filtering mapper

public class BloomFilteringMapper extends Mapper<Object, Text, Text, NullWritable> {
  private BloomFilter filter = new BloomFilter();
  protected void setup(Context context) throws IOException, InterruptedException {
    // Get file from the DistributedCache
    URI[] files = DistributedCache.getCacheFiles(context.getConfiguration());
    System.out.println("Reading Bloom filter from: ", files[0].getPath());
    // Open local file for read.
    DataInputStream strm =
      new DataInputStream(new FileInputStream(files[0].getPath()));
    // Read into our Bloom filter.
    filter.readFields(strm);
    strm.close();
    System.exit(0);
  }

  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map<String, String> parsed = transformXmlToMap(value.toString());
    // Get the value for the comment
    String comment = parsed.get("Text");
    StringTokenizer tokenizer = new StringTokenizer(comment);
    // For each word in the comment
    while (tokenizer.hasMoreTokens()) {
      // If the word is in the filter,
      // output the record and break
      String word = tokenizer.nextToken();
      if (filter.membershipTest(new Key(word.getBytes()))) {
        context.write(value, NullWritable.get());
        break;
      }
    }
  }
}

Writing a Bloom Filtering mapper

public static class BloomFilteringMapper extends Mapper<Object, Text, Text, NullWritable> {
  private BloomFilter filter = new BloomFilter();
  protected void setup(Context context) throws IOException, InterruptedException {
    // Get file from the DistributedCache
    URI[] files = DistributedCache.getCacheFiles(context.getConfiguration());
    System.out.println("Reading Bloom filter from: ", files[0].getPath());
    // Open local file for read.
    DataInputStream strm =
      new DataInputStream(new FileInputStream(files[0].getPath()));
    // Read into our Bloom filter.
    filter.readFields(strm);
    strm.close();
  }

  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map<String, String> parsed = transformXmlToMap(value.toString());
    // Get the value for the comment
    String comment = parsed.get("Text");
    StringTokenizer tokenizer = new StringTokenizer(comment);
    // For each word in the comment
    while (tokenizer.hasMoreTokens()) {
      // If the word is in the filter,
      // output the record and break
      String word = tokenizer.nextToken();
      if (filter.membershipTest(new Key(word.getBytes()))) {
        context.write(value, NullWritable.get());
        break;
      }
    }
  }
}