Assignment 2: Twitter Topic Modeling with Latent Dirichlet Allocation

Background

In this assignment we are going to implement a parallel MapReduce version of a popular topic modeling algorithm called Latent Dirichlet Allocation (LDA). Because it allows for exploring vast document collection, we are going to use this algorithm to see if we can automatically identify topics from a series of Tweets. For the purpose of this assignment, we are going to treat every tweet as a document and only use the words and hashtags in these tweets to identify the topics.

What to expect? What to hand in?

For this assignment, you can collaborate in pairs. This assignment is 25% of your overall grade. This assignment will take about a couple of weeks if you are work diligently. The total points you can score in this assignment is 100.

You will hand in a brief write-up consisting of the answers to the questions below followed by your implementation of the assignment.

Deadlines

Deadlines are firm until and unless you have discussed with me.

Write up: Provide a brief report of how you went about implementing the LDA algorithm. Explain succinctly your choices for design and how the map and reduce functions were implemented. With your code, provide some timing information on your runs. Note that you will be evaluated for your attempt at designing the algorithm rather than getting the right answer.

Programming: Email me and Yang Song with the following attachment [lastname]-HW-2-submit.tgz on or before Apr 14, 2015. On the subject line, please include the following tag- [COSC-526: HW 2].
Latent Dirichlet Allocation (LDA) for Topic Modeling

1 LDA Introduction

LDA is an unsupervised learning algorithm particularly suited for topic modeling. Let’s say we have a couple of tweets that have the following sentences:

1. the cat drinks milk and #meowed happily. #cute
2. our labrador ran through the gates to get his #bonetreat
3. new #puppy at home

LDA models topics that can be viewed as “cat_related” or “dog_related”. While (1) will be clearly related to “cat”, (2) and (3) are “dog” related topics. Our task is to automatically extract these aspects from the set of tweets that we are going to examine.

We will follow the notation given in the paper by Blei, Ng and Jordan (http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf). We will use a corpus of $M$ documents, where each document is a tweet. A tweet $d$ will have $N_d$ words and we will indicate, $w$ is a specific word in the tweet. These $N_d$ words could come from $K$ topics. A generative model for a tweet is:

- For each topic index $k \in 1, 2, \ldots, K$, draw topic distributions $\beta_k \sim \text{Dirichlet}(\eta_k)$.
- For each word $n \in 1, 2, \ldots, N_d$:
  - we will choose the topic assignment $z_{d,n} \sim \text{Multinomial}(\theta_d)$ and
  - we will choose word $w_{d,n} \sim \text{Multinomial}(\beta_{z_{d,n}})$.

$\alpha$ and $\beta$ are parameters. Note that $\theta_i$ is the topic distribution of tweet $i$, $\phi_k$ is the word distribution for topic $k$ and finally $z_{i,j}$ is the topic for the $j^{th}$ word in document $i$. Note that $w_{i,j}$ is the only observable, i.e., that is the only thing that we observe from the tweets that we can count. The other variables are referred to as latent variables (or hidden) and have to be estimated from the tweet documents.

Learning the various distributions (listed above) is a problem of Bayesian inference. The key inferential problem that we need to solve for LDA is that of computing the posterior distribution of hidden variables, given a document:

$$p(\theta, z, w | \alpha, \beta) = \frac{p(\theta, z, w | \alpha, \beta)}{p(w | \alpha, \beta)}.$$  (1)
Unfortunately, we can quickly see that this distribution is intractable to compute. As we discussed in class, the term $p(w|\alpha, \beta)$ (for normalization) has to marginalize over the hidden variables:

$$p(w|\alpha, \beta) = \frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)} \int \left(\prod_i \theta_i^{\alpha_i-1}\right) \left(\prod_{n=1}^N \sum_{i=1}^k \prod_{j=1}^V (\theta_i \beta_{ij})^{w_{jn}}\right) d\theta,$$

with a coupling between $\theta$ and $\beta$. One way to estimate Eqn. 1 is to use Markov chain Monte Carlo (MCMC) techniques such as Gibbs sampling, which we went over in the class. However, this approach can have some limitations for high dimensional datasets and for distributed computing (using MapReduce).

An alternative approach to Gibbs sampling is the variational inference approach that is particularly suitable for MapReduce. Variational approaches use optimization techniques to find a distribution over the latent variables. Since they have a clear convergence criterion, variational methods can work with distributed computing environments as well as high dimensional datasets.

Note that our challenge of computing Eqn. 1 is the coupling between $\theta$ and $\beta$, essentially the probabilistic dependencies between $z$ and $w$ shown in Figure 1. To overcome this challenge, we will replace the actual distributions in Eqn. 1 with a family of distributions on the latent variables.

### 1.1 Variational Inference Approach

With the variational method, we begin by positing a family of distributions $q \in Q$, over the same latent variables $z$ with a simpler dependency pattern than $p$, parameterized by $\Theta$. This family is characterized by
the following variational distribution:

\[ q(\theta, z|\gamma, \phi) = q(\theta|\gamma) \prod_{n=1}^{N} q(z_n|\phi_n), \]  

(3)

where the Dirichlet parameter \( \gamma \) and the multinomial parameters \((\phi_1, \phi_2, \ldots, \phi_n)\) are the free variational parameters.

**Question 1: [15 points]** Having specified a simplified family of probability distributions, the next step would be to set up an optimization problem that determines the values of these variational parameters \( \gamma \) and \( \phi \). The optimization problem would bound the log likelihood of a document with:

\[ \log p(w|\alpha, \beta) = \log \int \sum_z p(\theta, z, w|\alpha, \beta) d\theta = \log \int \sum_z \frac{p(\theta, z, w|\alpha, \beta) q(\theta, z|\gamma, \phi)}{q(\theta, z|\gamma, \phi)} d\theta. \]  

(4)

Show that the log likelihood, \( \log p(w|\alpha, \beta) \) is

\[ E_{q}[\log p(\theta|\alpha)] + E_{q}[\log p(z|\theta)] + E_{q}[\log p(w|z, \beta)] - E_{q}[\log q(\theta)] - E_{q}[\log q(z)]. \]  

(5)

**Hint:** Use Jensen’s inequality, i.e., for a random variable \( X \) and a convex function, \( \phi \), \( E[\phi(X)] \leq \phi(E[X]) \).

The difference between the left-hand side and the right-hand side of Eqn. 4 is referred to as the Kullback Leibler (KL) divergence between the variational posterior \( q \) and the actual/true posterior \( p \). Thus, we can write Eqn. 4 as:

\[ \log p(w|\alpha, \beta) = L(\gamma, \phi; \alpha, \beta) + D(q(\theta, z|\gamma, \phi)||p(\theta, z|\gamma, \phi)), \]  

(5)

where \( L \) denotes the first term in Eqn. 4 and \( D \) is the KL-divergence (second term in Eqn. 4) respectively.

We can now expand the lower bound by using factorizations of \( p \) and \( q \):

\[ L(\gamma, \phi; \alpha, \beta) = E_q[\log p(\theta|\alpha)] + E_q[\log p(z|\theta)] + E_q[\log p(w|z, \beta)] - E_q[\log q(\theta)] - E_q[\log q(z)]. \]  

(6)

Each of the five terms here, can now be expanded in terms of its model parameters \((\alpha, \beta)\) and variational
parameters \((\gamma, \phi)\):

\[
L(\gamma, \phi; \alpha, \beta) = \log \Gamma\left(\sum_{j=1}^{k} \alpha_j\right) - \sum_{i=1}^{k} \log \Gamma(\alpha_i) + \sum_{i=1}^{k} (\alpha_i - 1))\left(\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j\right)\right)
\]

\[
+ \sum_{n=1}^{N} \sum_{i=1}^{k} \sum_{j=1}^{V} \phi_{ni}\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j\right)
\]

\[
+ \sum_{n=1}^{N} \sum_{i=1}^{k} \sum_{j=1}^{V} \phi_{ni} w_{nj} \log \beta_{ij}
\]

\[
- \log \Gamma\left(\sum_{j=1}^{k} \gamma_j\right) + \sum_{i=1}^{k} \log \Gamma(\gamma_i) - \sum_{i=1}^{k} (\gamma_i - 1))\left(\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j\right)\right)
\]

\[
- \sum_{n=1}^{N} \sum_{i=1}^{k} \phi_{ni} \log \phi_{ni}
\]

1.2 Deriving the Variational Inference Updates

We first maximize in Eqn. 6 with respect to \(\phi_{ni}\), the probability that the \(n^{th}\) word was generated by the latent topic \(i\). Note that this is a constrained maximization since \(\sum_{i=1}^{k} \phi_{ni} = 1\). To maximize this, we form a Lagrangian, by isolating the terms which contain \(\phi_{ni}\) and adding the appropriate multipliers. Let \(\beta_{iv}\) be \(p(w^v_n = 1 | z^i = 1)\) for the appropriate \(v\). (Each \(w_n\) is a vector of length \(V\) with exactly one component equal to 1).

\[
L_{[\phi_{ni}]} = \phi_{ni}(\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j\right)) + \phi_{ni} \log \beta_{iv} - \phi_{ni} \log \phi_{ni} + \lambda_n\left(\sum_{j=1}^{k} \phi_{ni} - 1\right)
\]

**Question 2: [15 points]** Show that the update rule for the variational multinomial by taking the partial derivative with respect to \(\phi_{ni}\) and setting it to zero will be \(\phi_{ni} \propto \beta_{iv} \exp(\Psi(\gamma_i) - \Psi\left(\sum_{j=1}^{k} \gamma_j\right))\).

The next part of our optimization requires us to derive with respect to \(\gamma_i\), the \(i^{th}\) component of the
posterior Dirichlet parameter. The terms containing $\gamma_i$ are:

$$L_{[\gamma]} = \sum_{i=1}^{k} (\alpha_i - 1) (\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k})) + \sum_{i=1}^{k} \phi_{ni} (\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k}))$$

$$- \log \Gamma(\sum_{j=1}^{k} \gamma_j) + \log \Gamma(\gamma_i) - \sum_{i=1}^{k} (\gamma_i - 1) (\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k}))$$

(9)

**Question 3: [10 points]** Show that the update rule for the variational multinomial by taking the partial derivative with respect to $\gamma_i$ and setting it to zero will be $\gamma_i = \alpha_i + \sum_{n=1}^{N} \phi_{ni}$.

Now, each document has variational parameters $\gamma$, $\phi$ and $\lambda$ and we have to figure out how to update these as part of our MapReduce implementation. For the variational inference to be successful, we need an approach that is fast (and can be distributed). For this purpose, we will use the Newton-Raphson technique.

Recall the $\alpha$ parameter controls the sparsity of topics in the document distribution. We can think of updating $\alpha$ as follows:

$$\alpha_{\text{new}} = \alpha_{\text{old}} - \mathcal{H}^{-1}(\alpha_{\text{old}}).g(\alpha_{\text{old}}),$$

(10)

where the Hessian matrix, $\mathcal{H}$ and $\alpha$ gradient are, respectively, as:

$$\mathcal{H}(k,l) = \delta(k,l) C \Psi'(\alpha_k) - C \Psi'(\sum_{l=1}^{k} \alpha_l),$$

(11)

and

$$g(k) = C \left( \Psi\left( \sum_{l=1}^{K} \alpha_l \right) - \Psi(\alpha_k) \right)$$

$$+ \sum_{d=1}^{C} \Psi(\gamma_{d,k}) - \Psi\left( \sum_{l=1}^{K} \gamma_{d,l} \right).$$

(12)

Here $\delta(k,l)$ is the delta function, $\delta(k,l) = 1$ when $k = l$, and 0 otherwise. $\Psi'$ refers to the first derivative of the digamma function.

**Question 4: [10 points]** The Hessian matrix $\mathcal{H}$ depends entirely on a vector $\alpha$, which changes upon updating $\alpha$. The gradient $g$ is dependent on two terms, namely the $\alpha$ and $\gamma$ tokens. Now your decision for implementing the MapReduce version of LDA will have to do with what parts of Eqn. [12] Identify which parts of Eqn. [12] can be implemented as part of map and reduce operations. Note that $\alpha$ is the last parameter that you want to update. We will talk further about updating $\alpha$ in the next section.
2 Implementing LDA using MapReduce

2.1 Implementing the Mapper: Update $\phi$ and $\gamma$

The mapper will compute updates for these variational parameters and uses them to create sufficient statistics needed to update the global parameters. Given a document, the update for $\phi$ and $\gamma$ are:

$$
\phi_{v,k} \propto E[q[\beta_{v,k}].e^{\Psi(\gamma_k)}], \gamma_k = \alpha_k + \sum_{v=1}^{V} \phi_{v,k},
$$

where $v \in [1,V]$ is the term index and $k \in [1,K]$ is the topic index. In this case, $V$ is the size of the vocabulary and $K$ is the total number of topics. For this assignment, we will limit $K$ to be 10.

The algorithm for the mapper is shown in Algorithm 1. In the first iteration, the mappers initializes variables where $\lambda$ are initialized with the counts of a single document. The respective $\phi_{d,k}$ are initialized with $1/k$ for all $d$ and $k$. The $\gamma_i$ are initialized with $\alpha_i + N/k$ for all $i$.

Note that we represent a document as a term frequency vector: $w = ||w_1, w_2, \ldots, w_V||$, where $w_i$ is the corresponding term frequency in document $d$. We assume that the input term frequency vector is associated with all the terms in the vocabulary, i.e., if term $t_i$ does not appear in document $d$, $w_i = 0$.

function Mapper
Input: KEY: document ID $d \in [1, C]$ where $C = |C|
Input: VALUE: document content
  $\phi \leftarrow$ a zero $V \times K$-dimensional matrix
  $\sigma \leftarrow$ a zero $K$-dimensional row vector
  Read in the document content $||w_1, w_2, \ldots, w_V||$
while until convergence do
  for $v \in [1, V]$ do
    for $k \in [1, K]$ do
      Update $\phi_{v,k} = \frac{\lambda_{v,k}}{\sigma_k}. \exp \Psi(\gamma_{d,k})$
      Normalize $\phi_v$, set $\sigma = \sigma + w_v \phi_v$
      Update row vector $\gamma_{d,*} = \alpha + \sigma$
    for $k \in [1, K]$ do
      for $v \in [1, V]$ do
        Emit $\langle k, v \rangle$: $w_v \phi_{v,k}$
        Emit $\langle \Delta, k \rangle$: $\Psi(\gamma_{d,k}) - \Psi(\sum_{i=1}^{K} \gamma_{d,i})$.
        Emit $\langle k, d \rangle$: $\gamma_{d,k}$ to file
Note that $\sigma$ is just an accumulator for $\phi_v$. Note that in addition to passing the document, where you are updating the different parameters, you also have to pass the hash map of (term/word $\rightarrow$ count) of the
entire corpus. To create this hash map, make a pass over the entire set of tweets and count the number of unique words. Words can include both normal words (standard English words) and hashtags. You are also welcome to create the hash map which uses only hashtags. This will create a much smaller hash map that can be loaded into the Mapper function.

**Question 4: [20 points]** Explain your choice of the hash map in your report. Implement the Mapper function as outlined above. Note that Java implementation comes with a Digamma function in the Java Math library (http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/special/Gamma.html). You can use this implementation for the \( \Psi \) notation used throughout.

### 2.2 Implementing the Reducer: Update \( \lambda \)

The reduce function updates the variational parameter parameter \( \lambda \) associated with each topic. Thus, it requires aggregation over all intermediate \( \phi \) vectors, where:

\[
\lambda_{v,k} = \eta_{v,k} + \sum_{d=1}^{C} \left( w_{v}^{(d)} \phi_{v,k} \right),
\]

where \( d \in [1, C] \) is the document index and \( w_{v}^{(d)} \) denotes the number of appearances of the term \( v \) in document \( d \). \( C \) is the total number of documents.

The reducer algorithm will look as implemented in Algorithm 2.

**function** REDUCER  
**Input:** KEY: \( \langle \Delta, k \rangle \) and \( \langle k, d \rangle \)  
Compute the sum \( \sigma \) over all the sequences.  
Emit \( \langle v, k \rangle : \sigma \).

**Question 5: [20 points]** Implement the Reducer function as outlined above.

### 2.3 Putting it all together: Updating \( \alpha \)

**Question 6: [10 points]** Now we come back to updating \( \alpha \), which refers to the main hyper parameter that we were estimating as part of our Newton-Raphson technique. In the main function of your code, which we will call as the driver, we will just update our sum once more for the \( \alpha \) parameter. As shown in Eqn. [12], implement the driver function so that the correct updates to \( \alpha \) are made.
2.4 Testing your implementation

In order to test your algorithm, we have provided a small dataset from Twitter (about 20 GB). You can look at the dataset on the cluster using the HDFS location: hdfs:///data/tweets. You can see it via command line by using the command: hdfs dfs -ls /data/tweets. The data consists of tweets from the time of the Boston bombing event. We have about 6 days of tweets collected every minute. These correspond to the six folders in the tweets folder. Note that within each folder (named by its date), you will find a compressed json file recording a set of tweets every minute. A json file, if you recall, consists of a dictionary of <key, value> pairs. In order to process these files, you will have to write a pre-processor that will process the individual compressed json file and extract the key “tweet”. This will give you access to the actual document that you need to run the LDA algorithm. A template for processing your tweet file is given below:

```java
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import org.apache.commons.io.FileUtils;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.*;
import org.apache.lucene.index.*;
import org.apache.lucene.queryparser.classic.QueryParser;
import org.apache.lucene.search.*;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.store.LockObtainFailedException;
import org.apache.lucene.util.BytesRef;
import org.apache.lucene.util.Version;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import org.joda.time.format.DateTimeFormatter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.*;
import java.util.*;
import java.util.Map.Entry;
import java.util.zip.GZIPInputStream;
public static int processJSONFile (File jsonFile, IndexSearcher indexSearcher, Set<DateTime> processedDateTimes, DateTime startDT, DateTime endDT) throws FileNotFoundException, IOException {
    JsonParser parser = new JsonParser();
    BufferedReader fileReader = new BufferedReader(new InputStreamReader(new GZIPInputStream(jsonFile.getInputStream())));
    // Processing code here
}
```
GZIPInputStream(new FileInputStream(jsonFile)));

JsonElement jsonElement = null;
try {
    jsonElement = parser.parse(fileReader);
} catch (Exception ex) {
    ex.printStackTrace();
    log.debug("Exception caught while parsing JSON file: "+
        ex.getMessage());
    return 0;
}

JsonObject jsonObject = null;
try {
    jsonObject = jsonElement.getAsJsonObject();
} catch (Exception ex) {
    ex.printStackTrace();
    log.debug("Exception caught while parsing JSON file: "+
        ex.getMessage());
    return 0;
}

JsonElement tweetCountElement = jsonObject.get("tweet_count");
int fileTweetCount = tweetCountElement.getAsInt();

int tweetStoredCounter = 0;
int ignoredTweetCounter = 0;

JsonElement tweetsElement = jsonObject.get("tweets");
if (tweetsElement != null) {
    JsonArray tweetsArray = tweetsElement.getAsJsonArray();
    for (JsonElement tweetElement : tweetsArray) {
        Document tweetDocument = new Document();
        jsonObject = tweetElement.getAsJsonObject();
        DateTime tweetDT = null;
        for (Entry<String, JsonElement> entry : jsonObject.entrySet()) {
            String key = entry.getKey();
            if (key.equals("text")) {
                // convert to text field and write to lucene document with
tokenization
                String tweetText = value.getAsString();
entryField = new VecTextField(key, tweetText,
   Field.Store.YES);
tweetDocument.add(entryField);
}
}

// if start and end date times are set, check that the tweet date
time falls between time range
if (startDT != null && endDT != null) {
    if (tweetDT.isBefore(startDT) || tweetDT.isAfter(endDT)) {
        log.debug("Skipping tweet because it is outside defined date
        range: " + tweetDT.toString());
        ignoredTweetCounter++;
        continue;
    }
}

// Now that you got the tweet text, you can do what you want to do with
it...
fileReader.close();
return tweetStoredCounter;
}

You can have yet another driver to read the JSON files from a path separately as listed below:

public static int processJSONFolders (String jsonArchiveDirectoryPath, String
stopwordsFilePath, DateTime startDT, DateTime endDT) {
    try {

        File f = new File( stopwordsFilePath );
        if (!f.exists() || !f.canRead()) {
            log.error("Error reading stopwords file " + stopwordsFilePath);
            System.exit(0);
        }

        InputStream in = new FileInputStream(stopwordsFilePath);
        InputStreamReader reader = new InputStreamReader(in);
    } catch (IOException e) {
        e.printStackTrace();
    }

    // get the JSON files
    File archiveDirectory = new File(jsonArchiveDirectoryPath);
    if (!archiveDirectory.exists()) {

log.error("Archive directory "+jsonArchiveDirectoryPath+" does not
exist.");
System.exit(0);
}
if (!archiveDirectory.isDirectory()) {
  log.error("ArchiveDirectory "+jsonArchiveDirectoryPath+" is not a
directory.");
  System.exit(0);
}

Collection<File> jsonFiles = FileUtils.listFiles(archiveDirectory,
  fileExtensions, true);

int totalTweetsStored = 0;
// read and store the tweets
TreeSet<DateTime> processedDateTimes = new TreeSet<DateTime>();

int filesProcessed = 0;
for (File f : jsonFiles) {
  int tweetsProcessed;
  try {
    tweetsProcessed = processJSONFile(f, indexWriter, indexSearcher,
      processedDateTimes, startDT, endDT);
    if (tweetsProcessed > 0) {
      indexWriter.commit();
    }
    totalTweetsStored += tweetsProcessed;
    log.debug(f.getName() + " processed with "+ tweetsProcessed + "
tweets stored");
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }

  ++filesProcessed;
}
return totalTweetsStored;

As we discussed in the class, the command for accessing the hadoop cluster is here:

ssh -p 2222 username@cosc526fa15.eecs.utk.edu -L 50070:localhost:50070
The -L option is for tunneling those ports. After connecting via SSH using the command above, you should be able to open a web browser and navigate to the below sites to see the respective cluster pages.

- http://localhost:50070/: web UI of the NameNode daemon
- http://localhost:50030/: web UI of the JobTracker daemon
- http://localhost:50060/: web UI of the TaskTracker daemon

As part of your report, also include the log from the web interface to see what jobs were running and how they are running.

You also have an option to implement your map reduce functions as part of a python program and run it on the Amazon cluster using the credits that were distributed in class. The data location will be available on request. The data will be stored in the same way as you have access to the local cluster.