DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF GEORGIA

CSCI6900 Assignment 4: LDA on Spark

DUE: Friday, April 10 by 11:59:59pm

Out March 20, 2015

1 OVERVIEW

In this assignment, we will make use of Spark's distributed implementation of Latent Dirichlet Allocation (LDA), a topic modeling algorithm. Because it allows for exploring vast collections of documents, 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 will treat every tweet as a document and only use the words and hashtags in these tweets to identify the topics.
LDA is an unsupervised learning algorithm for topic modeling. Take, for example, the following tweets:

- The cat drinks milk and #meowed happily. #cute
- Our labrador ran through the gates to get his #treat.
- New #puppy at home!

LDA models topics that can be interpreted, in the above example, as "cat related" (first) or "dog related" (last two). While LDA will not given specific *names* to the topics, it can identify words that tend to cluster together as it learns some concept of a topic; see the student paper written by Etsy employees from earlier this semester [1].

The Spark implementation, introduced in the latest v1.3 release, follows the original LDA paper Blei *et al* 2003 [2]. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words. Specifically,

[1] `http://cobweb.cs.uga.edu/~squinn/mmd_s15/papers/p1640-hu.pdf`
[2] `http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf`

we will use a corpus of M documents, where document $\vec{d}$ is a tweet. A tweet will have N_d words. These N_d words come from K latent topics. The formal model for generating a tweet is as follows:

- For each topic index $k \in 1, 2, ..., K$, draw topic distribution $\theta_k \sim \texttt{Dir}(\alpha_k)$, where `Dir` is the Dirichlet distribution [3].
- For each word $n \in 1, 2, ..., N_d$ in tweet $\vec{d}$:
  - We choose the topic assignment $z_n \sim \texttt{Multinomial}(\theta_d)$.
  - We will choose word $w_n \sim \texttt{Multinomial}(z_n, \beta_{z_n})$.

Blei *et al* note that the resulting posterior for this model is intractable to solve directly, due to the coupling between β (an estimate of word importance under a certain topic) and θ (distribution of topics within a single document). One way to estimate the posterior is to use Markov Chain Monte Carlo (MCMC) techniques such as Gibbs sampling (often referred to as "collapsed" Gibbs sampling). While this approach can converge quickly, it is not guaranteed to converge and can run into problems with high-dimensional data.

An alternative approach to Gibbs sampling is *variational inference*. This approach uses convex optimization techniques to find a distribution over the latent variables. Since convex problems have a clear convergence criterion, and because variational methods decouple the dependence between β and θ in the posterior, these methods are particularly well-suited to distributed environments. Spark's implementation uses the variational methods laid out in Blei *et al*.

2 PROGRAM

For this assignment, you will need to upgrade to the latest version of Spark (1.3). The latest Spark documentation provides an overview of the implementation, its associated parameters, and an example for how to invoke the method [4]. It is only available for the Scala and Java APIs, so you will be restricted to one of those languages for this assignment (sorry Python users; see the bonus if you want to change this!).
The code you will write for this assignment will be minimal, and will largely be for the purpose of addressing the questions in the Deliverables section. However, it should be noted: Spark's LDA implementation **does not work directly with text**; rather, you will have to preprocess the text into the format Spark expects. Specifically, each document is represented as a vector of integers, where the integers are unique IDs that identify vocabulary terms. One of your first tasks will be to convert the raw text of each tweet into a list of integer IDs, one for each token in the tweet.

[3] `http://en.wikipedia.org/wiki/Dirichlet_distribution`
[4] `http://spark.apache.org/docs/latest/mllib-clustering.html#latent-dirichlet-allocation-lda`

3 DATA

For this assignment, we are using a collection of tweets that have been processed into the format:

```
tweet_id tweet_text
```

where `tweet_id` is the unique 64-bit integer identifying the tweet, and `tweet_text` is the content of the tweet itself. The two fields are separated by a tab character, and tweets (which we consider documents) are separated by newlines.
The data are available in a 2GB gzip archive here:
`http://ridcully.cs.uga.edu/assignment4/twitter.tar.gz`

WARNING: The tweets have not been sanitized, so you may find some "colorful" language. These are not intended to offend anyone; they constitute a random sampling of the Twitter timeline over a few days in November 2014.

4 DELIVERABLES

When grading your assignments, I will use the **most recent commit in your repository *prior* to the submission deadline**. I should be able to clone your repository, change to the directory containing this assignment, and run your code. If my BitBucket username is `magsol`, my assignment should be in `magsol/assignment4`.

Please include a `README` file with 1) the command you issued to run your code, and 2) any known bugs you were unable to fix before the deadline. In addition, please provide answers to the following questions.

1. How many unique words are there in this corpus?
2. Feed the documents into the Spark LDA, and run it for $K = 10$ topics. Take a look at the top-weighted words associated with each topic. Provide your best inference of the topics Spark has identified. Provide the top-weighted single word for each of the 10 topics Spark identified.
3. You'll notice in the previous question that your answers are likely dominated by useless articles (a, an, the), prepositions (in, with, about), and pronouns (my, me, your). Put together a list of stop words, and when you build your vocabulary prior to feeding it to Spark, filter out the stop words. Provide the list of stop words you chose.
4. Once you implement a stop words filter, run the LDA again with $K = 10$ topics. Now provide your best inference of the topics Spark has identified, and provide the top-weighted single word for each of the 10 topics.

5. The Spark documentation indicates that its LDA is experimental, and currently cannot classify unobserved documents according to a pre-learned LDA model. Using what you know about LDA and natural language processing from this course, provide a high-level sketch of how classification could be performed using a learned LDA model on a new, unobserved tweet. Make sure to identify specific variables that you would need to use (though you can assume they are already populated, since the model has already been trained).

If you run your code on AWS: Include your Spark logs in your submission. Also provide the details of your cluster architecture and discuss its performance as compared to running your program on your local machine in standalone mode.

5 BONUS 1

Read the paper titled *Mr. LDA: a flexible large scale topic modeling package using variational inference in MapReduce* [5], in which the authors implement LDA using variational inference on Hadoop (their code is posted on github [6]). Implement this algorithm on the Spark framework.

Yes. **Implement this entire algorithm;** it is sufficiently different from the MLlib version. In particular, pay attention to the three main phases of variational inference:

1. The mapper is responsible for computing the partial updates for γ, λ, and α.
2. The reducer is responsible for merging the partial updates for λ and α.
3. The driver is responsible for reconstituting γ, and performing the convex optimization over α.

Some important points to keep in mind, should you choose to accept this bonus assignment:

- Each mapper processes a single document. Thus, the number of unique words in a document relative to the entire vocabulary will be *very small*.
- λ is your term-frequency dictionary, normalized to be probabilities over topics. **You may assume this will fit in the memory of each worker; put another way, you can make this a broadcast variable.**
- Subsequent to the previous point, while Algorithm 1 states that ϕ is a $V \times K$ matrix, it is actually a sparse hashmap: only the terms that appear in the document will be represented by a K-length vector.
- γ has dimensions $M \times K$, where M is the number of documents. You will have to be clever will this one; it's not sparse, so you can't broadcast it to every worker.

[5] `http://dl.acm.org/citation.cfm?id=2187955`
[6] `https://github.com/lintool/Mr.LDA`

- Wherever convergence is involved, make sure you specify a maximum number of allowed iterations.
- The Ψ function is a special function that you can find in Apache Commons; specifically, it is implemented using `digamma`. Its derivative, Ψ' (which you'll use in the driver), is the `trigamma` function.
- Sections 3.1, 3.2, 3.3, and 3.4 detail out what you need to know to implement Mr. LDA on Spark.
- **Your program should output top-ranked words for each topic and their weights**. No other output is needed.
- **I will award partial credit.** If you attempt part of this question, I will award points based both on what works, and how much work you put into it (show me those commit messages!).

6 BONUS 2

Implement Bonus 1, but distribute the term frequencies λ. Put another way, implement a message-broadcasting protocol similar to the one you used in the Naive Bayes assignment, whereby you no longer broadcast the entire term-frequency dictionary λ to all the workers. Instead, you send only the counts that are relevant to the document the worker is processing (hint: try to take advantage of the fact that the documents themselves never change, even though you have to iterate over them multiple times).

7 MARKING BREAKDOWN

- Code correctness (commit messages, program output, and the code itself) **[50 points]**
- Questions 1 and 3 **[5 + 5 points]**
- Questions 2 and 4 **[15 + 15 points]**
- Question 5 **[10 points]**
- Bonus 1 **[100 points]** (no that's not a typo; it's a full homework assignment)
- Bonus 2 **[50 points]** (still not a typo; there are 250 points possible on this homework)

8 OTHER STUFF

Spark's performance, while generally much better than Hadoop, can vary wildly depending on how effectively you are caching your intermediate results (and what your memory strategy is [7]). In particular, I highly recommend this chapter from *Learning Spark* [8], as it provides

[7] `http://spark.apache.org/docs/latest/tuning.html`
[8] `https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch04.html`

invaluable advice for partitioning your data in the most effective way possible so as to minimize network shuffling.

You are not required to run your code on AWS, but if you do, Spark includes scripts for easily setting up an AWS cluster [9]. The only requirement is to run Spark in standalone mode on your local machine; we will vary the number of cores used to "simulate" a distributed environment.

[9] `http://spark.apache.org/docs/latest/ec2-scripts.html`