Topic Modelling on Wikipedia data using LDA

Ankit Gohil, Christopher Parnin, Tim Menzies
Department of Computer Science
North Carolina State University
Raleigh, USA
{agohil, cjparnin, tjmenzie}@ncsu.edu

Abstract—*Topic Modeling is a text mining technique which can be be used to identify patterns in large collection of texts. A topic model takes a corpus of texts as input and discovers a set of “topics” - a cluster of co-occurring words - as well as the distribution of these words in the corpus. A good topic model will group the words which make sense when they occur together in a document. One of the most popular models for topic modeling is Latent Dirichlet Allocation, or LDA proposed by Blei et al. For LDA to work effectively, some preprocessing on the corpus is required. This report will talk about preparing the Wikipedia data and some other corpus experimented upon for LDA. This will also talk about the results obtained by running LDA on complete Wikipedia data.*

Keywords*- Topic Modeling; Latent Dirichlet Allocation; Apache Spark, Wikipedia;*

I. INTRODUCTION

Cluster computing frameworks are becoming increasingly important for large-scale text mining. Processing text corpus in a distributed environment can significantly improve the compute time of applications. Apache Spark is a cluster computing framework with capabilities for fault tolerance and parallel data processing. Additionally, Spark also provides out of the box machine learning algorithms in its MLLib package, which also includes LDA. Spark’s capabilities are fully utilized when jobs are run in a cluster of nodes. While it’s easier to configure a Spark cluster on commercial cloud providers like AWS, IBM Bluemix etc., it turns out to be costly from a research perspective where you are required to run multiple big data experiments on a daily basis. A part of this project was to automate spawning the Spark cluster on university’s VCL nodes. The instructions for setting up a Spark cluster on VCL can be found here[1]. Once the cluster is set up, we can run the preprocessing and LDA jobs in a distributed fashion.

II. BACKGROUND

The developers of Spark claim it to be 100 times faster than Hadoop MapReduce for large scale data processing. While MapReduce programs rely on two-stage computation storing the results of reduce stage on disk, Spark runs in-memory on the cluster. Spark uses Resilient Distributed Datasets, or RDD, an immutable distributed collection of objects, to perform in-memory computations on large clusters. RDDs facilitates the programmers to do a fast computation for iterative algorithms as well as interactive data analysis. Nevertheless, Spark needs a lot of memory. One of the challenges while writing a Spark program is to tune the runtime parameters such that Spark program does not cause Out Of Memory exception. For example, an attempt to collect all the results on a driver may result in such an exception.

LDA is a way of automatically discovering topics in a collection of documents. The traditional implementations of LDA run like a batch process where you require the entire corpus to be loaded into main memory. Online LDA, a variation of the original LDA, allows the model to be trained incrementally in small batches.

III. RELATED WORK

LDA modeling on whole Wikipedia has been done previously by Apache Spark committers before they integrated the LDA algorithm in Spark[2]. Distributed LDA has also been implemented in Python’s gensim package which also includes scripts for pre-processing the data[3]. While gensim provides batch LDA, Spark has options to choose the type of LDA model including the online LDA.

1 https://github.com/amritbhanu/Spark VCL
2 https://databricks.com/blog/2015/03/25/topic-modeling-with-lda-mllib-meets-graphx.html
3 https://radimrehurek.com/gensim/dist_lda.html

IV. EXPERIMENT

The experiments performed as part of this project can be broadly divided into below 3 categories:
1) Spin a Spark cluster on VCL nodes
2) Run LDA on a small corpus of scientific papers
3) Run LDA on Wikipedia data dump.

1) *Spinning a Spark cluster on VCL nodes*

Much of the work for automating the Spark cluster was done by my colleague and the instructions to set up the cluster can be found at his Github repository[1]. It seamlessly creates a Spark cluster integrated with HDFS, which can be used to store large amount of data with replication to provide fault tolerance. The virtual machine image used for each node has 8 GB memory and around 39 GB of disk space. When a Spark cluster is launched, some of this memory is consumed by the JVM and other system threads. Hence, around 6GB of memory is available for the executors to run Spark jobs.

2) Run LDA on a corpus of scientific papers

To get started with, the LDA was run on a small corpus of around 130 PDF scientific papers. LAPDF is an effective tool for extraction of full text from scientific articles[4]. The corpus was created after extracting plain text from the above described scientific articles and undergoing the usual NLP preprocessing steps like stop word removal and stemming (described in section 5). Multiple iterations of LDA job were run on a 3-node Spark cluster with executor memory of 2GB each to benchmark the runtime performance of the same job on NCSU's High Performance Computing cluster. Each job on VCL cluster took less than 5 minutes on the preprocessed corpus. The results of this experiment helped us decide to use the Spark cluster on VCL nodes against going for the HPC nodes.

3) *Run LDA on Wikipedia data dump*

The Wikimedia foundation releases a monthly dump of English Wikipedia data as a single, large XML file compressed as bzip2 file. The compressed file is about 12 GB compressed and 53 GB uncompressed. This XML contains some metadata and a page element for each Wikipedia page. This element contains information like page title and its text along with other information. The text is in Wikimedia markup syntax[5] which is used by Mediawiki software to format the page as HTML. Our first task is to extract plain text from this mark up and preprocess it to input it to the LDA model. The next section discusses the necessary preprocessing steps.

4 http://scfbm.biomedcentral.com/articles/10.1186/1751-0473-7-7
5 https://en.wikipedia.org/wiki/Help:Wiki_markup

V. PREPROCESSING

Preprocessing is an essential step in data mining tasks. It involves cleansing the data, removing noise and often times dimensionality reduction by reducing the volume of data to process. In our experiment, we preprocess the Wikipedia dump to generate suitable input for LDA. The first step is to extract plain text from Wikimedia markup syntax. There are several open source parsers available to do this job[6]. We first ran a serial implementation of preprocessing job on HPC cluster which took 4-5 hours to complete. This step was later converted to a parallel implementation in Spark using the APIs provided by Cloud9 project which are designed for use with Hadoop MapReduce[7].

After extracting plain text, each article was converted into a bag of words. This involves removing punctuations, lowercasing the terms and converting them to their root forms. Converting the different inflectional forms to a single term is called stemming or lemmatization. For example, *democracy* and *democratic* don't deserve to be separate terms. This step is necessary before counting term frequencies. We have used the Java API provided by Stanford Core NLP project for lemmatization. This returns a list of tokens for each document which is then subjected to TF-IDF computation. TF-IDF, which stands for "term frequency-inverse document frequency" is a common weighing scheme in text mining and helps in dimensionality reduction of the corpus. We compute TF-IDF score of each term and take top 10% or 20% terms in the corpus to be input to LDA.

An alternative to computing TF-IDF is to use Spark's CountVectorizer pipeline. CountVectorizer converts a collection of text documents to vectors of token counts. We can specify the size of the vocabulary while creating a CountVectorizer object, and it will create vectors of size equal to the mentioned vocabulary size. The model produces sparse representations for the documents over the vocabulary, which can then be passed to LDA[8].

VI. RESULTS

We integrated the preprocessing steps and LDA into a single Spark job and run it on a 30-node VCL cluster. Each executor was assigned 6 GB memory for the Spark job. Transferring the uncompressed 53GB XML file to

6 https://www.mediawiki.org/wiki/Alternative_parsers
7 https://lintool.github.io/Cloud9/docs/content/wikipedia.html
8 https://spark.apache.org/docs/latest/ml-features.html#countvectorizer

HDFS took more than 9 hours. Pre-processing and LDA took a little less than 3 hours to complete.

VII. CONCLUSION

In this project, we explored two approaches to heavy text mining applications – scaling up vs. scaling out. Scaling up was performed on university's HPC cluster and it was taking 4-5 hours for preprocessing step alone which was run sequentially. On the other hand, distributed LDA job on Spark VCL nodes took around 3 hours to complete on a 30-node cluster, including preprocessing. The compute capacity can be increased on demand by adding more nodes to the cluster. We run online variation of LDA and the parameters passed to it can be experimented with. This will help in benchmarking the LDA performance with different parameters as well as the topic stability with online LDA.