Latent Dirichlet Allocation implementations on Apache Flink

Dmytro Arbuzin
375083
dmytro.arbuzin@campus.tu-berlin.de

Diana Matar
376692
diana.matar@campus.tu-berlin.de

Alejandro Arauco Dextre
377013
alejandro.arauco@campus.tu-berlin.de

Abstract

In this paper we examine two different implementations of the Latent Dirichlet Allocation algorithm for topic modelling: Gibbs sampling and online Variational Bayes. Both implementations are done over a parallel data engine known as Apache Flink and tested over a pre-processed dataset. Ultimately, these implementations are compared to each other in both Apache Flink and Apache Spark, where both were originally found.

KEYWORDS: Machine learning, topic modelling, LDA, Flink, Spark

1 Introduction

Both the sciences and industry are currently undergoing a profound transformation: large-scale, diverse data sets - derived from sensors, the web, or via crowd sourcing - present a huge opportunity for data-driven decision making. This data poses new challenges in a variety of dimensions: in its unprecedented volume, in the speed at which it is generated (its velocity) and in the variety of data sources that need to be integrated. A whole new breed of systems and paradigms is currently developed to be able to cope with these challenges.

One of the good examples of such new systems are parallel data processing engines like Apache Spark [5] and Apache Flink [4], which are currently becoming more and more popular replacing and utilizing MapReduce programming model [8]. Flink and Spark are fast and effective for different kinds of data manipulations and aggregations due to in-memory computations. However, the ultimate goal of the data science is to extract knowledge from the raw data, so in order to be consistent with this objective parallel data processing engines required not only to process data efficiently, but also be able to apply different kinds of graph analysis and machine learning algorithms. The last one can be challenging due to the fact, that some machine algorithms are not suited for distributed systems and simply can not be parallelized, another ones require some clever techniques to be able to run on top of distributed systems.

To support this goal Apache Spark has introduces machine learning library called Mlib [11]. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs. Apache Flink also has introduced machine learning library FlinkML [6], however currently it provides users with only some simple algorithms.

This paper discusses the implementation of Latent Dirichlet Allocation (LDA) [7], one of the most common topic modelling algorithms, for Apache Flink system.

The remainder of the paper is structured as follows: Section 2 explains the core concept of the LDA and discusses different implementation techniques. Section 3 introduces the main design challenges of the LDA implementation for Apache Flink. Section 4 evaluates Flink LDA performance on big datasets. It also compares Online LDA vs Gibbs Sampling as well as compares our Online LDA implementation for Flink with Online LDA for Spark. 5 summarises the paper.

2 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) [7] is a wide-spread topic modelling machine learning algorithm, which was initially introduced by David Blei, Andrew Ng, and Michael I. Jordan in 2003. The authors described LDA as generative probabilistic model for collections of discrete data such as text corpora. In this paper we follow original paper notation.

LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. LDA model is represented with the following primary parameters:

- *Word*. Basic unit of discrete data. It defined to be an item from the *vocabulary* indexed by {1,....,V}.
- *Document*. Sequence of N *words* indexed by {w1,w2,...,wN}.
- *Corpus*. The collection of M *documents* indexed by {d1, d2,...., dM}.

Therefore, the LDA model itself utilises many secondary parameters in order to calculate distributions. The only one input parameter for the LDA model is the number of topics K to be discovered, all other parameters are set by default or calculated during the training. Here are common parameteres for LDA model:

- *Alpha*. The hyper-parameter for the Dirichlet prior weight of topic K in a document. It can be a vector meaning that each topic has different priority, but usually it is just one number, meaning that all topics are equal. The default value of alpha is 1 divided by number f topics.
- *Beta*. The hyper-parameter for the Dirichlet prior weight of word a *w* in a topic. Usually the same for all words, however can be a vector.
- *Theta*. It represents topics distribution over document.
- *Gamma*. It represents words distribution over topic.

Figure 1 presents LDA model graphically using plate notation. In addition to already described parameters, *z* represents the topic of word *w* in each document.

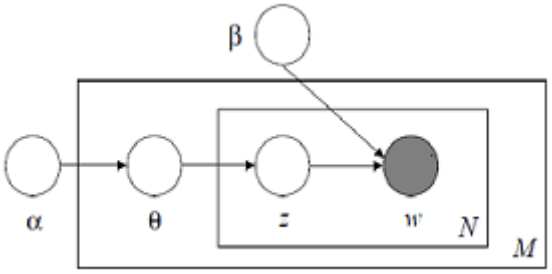

Figure 1: Graphical model representation of LDA. The boxes are "plates" representing replicates. The outer plate represents documents, while the inner plate represents the repeated choice of topics and words within a document. [7]

As it seen from the figure 1 LDA consists of 3 levels, corpus level, document level and finally word level. The output of the LDA model is always the matrix (K x V), with the probabilities of each word *V* from the vocabulary being related to certain topic *K*. However, in order to calculate this matrix the model can use different algorithm to count and update the parameters *Theta* and *Gamma*. Below we present few of them.

2.1 Gibbs sampling

In this version of the LDA algorithm, the parameters *theta* and *gamma* are not estimated but instead calculated based on the posterior distribution over the assignment of words to topics. The calculations of these probabilities are done using a Markov chain Monte Carlo procedure in order to avoid the complexity of computing conditional probabilities over a large state space.[9]

2.2 Online LDA

In order to calculate posterior LDA parameters methods like Gibbs Sampling or Variational Bayesian inference (VB) require full corpus to be available in the beginning, which is not always possible in modern applications. Imagine discovering topics for twitter feed or news RSS feeds. In order to overcome this problem Matthew D. Hoffman introduced online variational Bayes algorithm in 2010 [10]. The algorithm is shown at the Figure 2.

Define $\rho_t \triangleq (\tau_0 + t)^{-\kappa}$
Initialize $\boldsymbol{\lambda}$ randomly.
for $t = 0$ to ∞ **do**
  E step:
  Initialize $\gamma_{tk} = 1$. (The constant 1 is arbitrary.)
  repeat
    Set $\phi_{twk} \propto \exp\{\mathbb{E}_q[\log \theta_{tk}] + \mathbb{E}_q[\log \beta_{kw}]\}$
    Set $\gamma_{tk} = \alpha + \sum_w \phi_{twk} n_{tw}$
  until $\frac{1}{K}\sum_k |\text{change in} \gamma_{tk}| < 0.00001$
  M step:
  Compute $\tilde{\lambda}_{kw} = \eta + D n_{tw} \phi_{twk}$
  Set $\boldsymbol{\lambda} = (1 - \rho_t)\boldsymbol{\lambda} + \rho_t \tilde{\boldsymbol{\lambda}}$.
end for

Figure 2: The algorithm of online LDA. [10]

It follows the original Variational Bayesian, however it splits the the corpus into mini-batches,which theoretically allows corpus size *M* to be infinite. The algorithm can be showed very good performance results in the original paper, therefore we decided to implement it for Apache Flink, fitting nicely the original streaming architecture of Flink.

3 Implementation details

Here we explain both implementations done for Apache Flink.

3.1 Gibbs sampling

The implementation of Gibbs sampling LDA on big data platforms like Apache Spark or Apache Flink can be summarized into 6 steps:

1. Start platform and initialize parameters
2. Read documents into HDFS
3. Construct a dictionary based on the documents' content
4. Initialize a topic assignment array for each document
5. Infer documents' topic distribution and word assignment per topic using Gibbs sampling
6. Save results into HDFS

3.1.1 Start platform and initialize parameters

Refers to basic set up of execution environment, encoding and parallelization. Input parameters: *alpha*, *beta*, number of topics and numbers of iterations to be executed on Gibbs sampling step.

3.1.2 Read documents into HDFS

In the implementation used, the files are loaded into a dataset where each tuple represents a document composed of a document ID and an array of its words with duplicates.

3.1.3 Construct a dictionary based on documents' content

An alphabetically sorted deduplicated dictionary is created based on the words of all the documents loaded. Each word in this dictionary is sequentially assigned with a word ID.

3.1.4 Initialize a topic assignment array for each document

A random topic ID is assigned for each word on each document. Based on those assignments, two global variables, *nkv* and *nk* are initialized. Variable *nkv* consists of an integer matrix of dimensions *number of topics x dictionary size* where the random assignments per topic per word are summarized. Variable *nk* consists of an integer vector that contains the subtotal of each row of *nkv* on each of its elements.

3.1.5 Infer documents' topic distribution and word assignment per topic using Gibbs sampling

For each word on each document, a topic re-assignment is done based on the outcome of building a probability topic distribution. This new topic distribution is based on the current distribution stored in *nkv* and *nk* variables and the *alpha* and *beta* parameters. Once the re-assignment is done, global variables *nkv* and *nk* are updated. This process is repeated as many times as the corresponding input parameter is set.

3.1.6 Save results into HDFS

Finally, two results are obtained: first, the distribution of words per topic and their corresponding probabilities (namely *gamma*), which are calculated based on the final value of *nkv* global variable and, second, the probability topic distribution per document calculated from the topic assignment array found on each document from step 4, which is known as *theta*.

3.2 Online LDA

In order to run Online LDA on Apache FLink next steps are required to be implemented:

1. Initialize the model with prior parameters
2. Submit miniBatch of documents size *M * miniBatchFraction* to the expectation step of VB.
3. Expectation Step
4. Maximization step
5. Create the LDA model

3.2.1 Initialization and default parameters

Initial prior parameters for Online LDA are next: *alpha* = 1/K; *beta* = 1/K; *kappa* = 0.51; *tau* = 1024; *miniBatchFraction* = 0.05; *gammaShape* = 100; *result matrix K x V* with random initial parameters.

Parameters *kappa* and *tau* are special values for early iterations. We use recommended numbers from the original paper.

3.2.2 Submitting mini-batch

After initializing the model we feed it with the documents, sampling the whole corpus with the size of 5% (recommended parameter). Here we use Flink Batch API, however, with a few modifications the algorithm can be adjust to Flink Streaming API.

3.2.3 Expectation step

At this step we submit a subset (like 5%, decide by the miniBatchFraction) of the corpus to the Online LDA model, and it will update the topic distribution adaptively for the terms appearing in the subset.

For each mini-batch we calculate common parameters *eLogTheta* and *expELogBeta* based on current result matrix and create a broadcast variable out of *expELogBeta* to pass it to map funtion later.

Then we filter the documents for empty ones and index them to have a local index inside the batch.

After all the preparations are done, we finally calculate *gamma* and *theta* distributions with Variational Topic Inference method. Here we utilize FLink distributed architecture and uses MapPartition function. Inside every mapper we calculate distributions for every document within for-each loop and then return the sum of distributions from each mapper together with a indexed list of *gamma* vectors.

The output of MapPartition function is a dataset of result matrices from each partition with *gamma* vectors for each document. At this point we use reduce function and calculate the sum of all matrices. We don't use *gamma* vectors in this version, however it can added for optimization in the future.

3.2.4 Maximalization step

After obtaining the result matrix for each miniBatch submission, we update the global result matrix with the new values.

3.3 LDA model

In the end, after the whole corpus is processed we create new LDA model with the prior parameters and topic matrix *lambda*, which is the result of the algorithm. We can use this model later to discover topics of new documents or any other statistical information.

3.4 Implementation details and future optimization

In this implementation we were following the example of the same algorithm for Apache Spark system, so the logic of classes and methods follows the Spark version, however we also were trying to utilize Flink's ML library as much as possible using Flink ML's Matrices and Vectors datatypes, in the end it caused a problem, as operations on Flink ML datatypes are quite limited, so we developed a helper class called *LDAUtils* with static functions to perform different actions on Flink datatypes (transpose, product, dot, etc). Performance can be optimized much better by using some common library for all operations. Currently in different parts we are using: Apache Commons Math Library, ScalaNLP

Breeze, Flink ML library and our own methods, which leaves a lot of space for improvement and optimization.

4 Experimental evaluation

4.1 Datasets

We use two datasets for experiments. The first is 20 Newsgroups[1] for local evaluation. It consists of 20,000 newsgroup documents with a vocabulary of 6000 words. The second dataset is an extract of Dbpedia entities via a Wikipedia dump with size of 5 GB[3]. The final version of the second dataset consists of 400,000 documents with two different vocabularies of 6000 and 2000 words. this dataset is used for scalability evaluation.

4.2 Data preparation

Data preparation phase is challenging and time consuming. Providing big volume, free of cost collection of documents is only available via Wikipedia dumps. However, it is a complex process to extract readable content from Wikipedia dumps. The extracted content is cleaned, stemmed and tokenized. The dictionary size is limited by a defined threshold based on the word frequency across the whole dataset. For Online LDA implementation, the model does not work directly with text, therefore, the corpus is preprocessed into collection of vectors in which each document is represented as a vector of numbers and those numbers are unique IDs that identify dictionary terms.

4.3 Online LDA Evaluation In Spark

preprocessed The machine learning library of spark, called MLlib, implements an algorithm for performing online variational inference originally described by Hoffman et al[2].In the provided algorithm, LDA model processes the data incrementally in small batches where it can easily scale to very large datasets. We prepared test case to evaluate the model. several experiments were conducted with the change of the following settings:

1. Corpus size: 3.5 GB for vocabulary of 2000 word and 20 GB for vocabulary of 6000 words.

2. Number of topics.

In terms of volume, figure3 shows that LDA model linearly speedup when scaling the data.

While it is totally understandable that the increase of topics number would affect the computation cost and increase the processing time, figure4 reflects that the iterative computation of LDA model in spark implementation archives linear stability over the increase of number of topics.

Considering the computational complexity of LDA, the experiment results were really impressive. MLlib is powerful and very user-friendly as well its integration with Spark components gives it computational power.

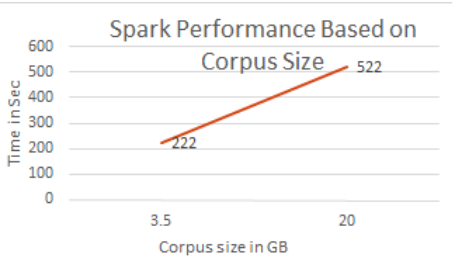

Figure 3: 80% growth rate in the volume of the corpus increases the processing time by 50%.

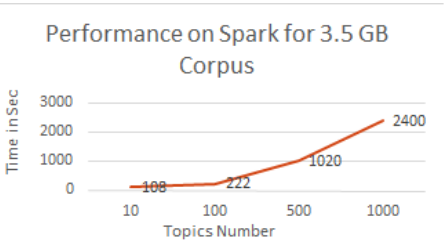

Figure 4: Linear stability is achieved by the increase of number of topics.

4.4 Online LDA Evaluation In Flink

To evaluate our implementation of online LDA on FLink, we prepared test case and conducted several experiments with the change of corpus size and the number of the topics as follows:

1. Corpus size: 0.3 , 1.5 and 3.5 GB of data.

2. Number of topics: 10, 100 and 500.

Figure5 shows that the increase of corpus size in Flink LDA model gradually affects the performance, as expected. However the model does not maintain linear increase of the processing time as the case in Spark .

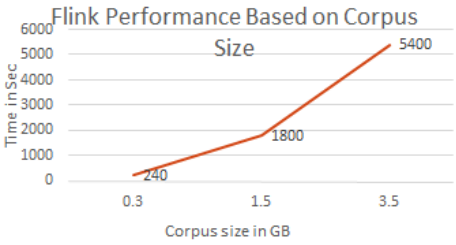

Figure 5: 60% growth rate in the volume of the corpus increases the processing time by 80%.

LDA is a memory intensive process that requires large amount of memory and computational power. During our experiment on shared cluster we have run out resources several times also due to heavy iterative computation the system was slowing down and de-allocating resources during the execution time special with very large corpus.

Figure6 shows that the increase of topics number dramatically affect the processing time.

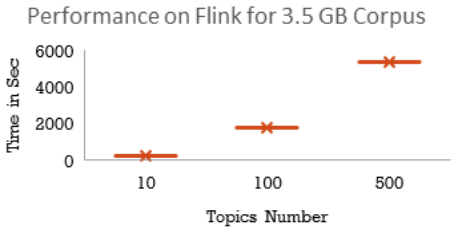

Figure 6: The increase of number of topics has a very noticeable impact on the processing time.

As shown in figure7 Flink showed good results on small data-sets, however it performed poorly on larger scale.On the other hand spark has maintained its high performance and scalability through our experiments.

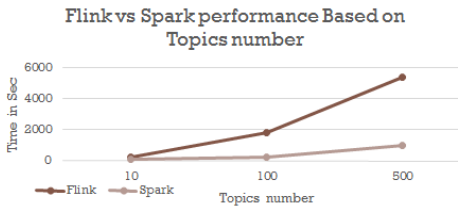

Figure 7: Implementation of Online LDA on Flink cannot achieve linear stability over performance.

4.5 Gibbs sampling Evaluation In Flink

Conducting the experiment of Gibbs sampling on small scale was reasonable. For corpus of 500 MB, processing time is 750 second. However, on larger scale the algorithm was slower than what we expected. Execution for corpus of 1.5 GB took longer than two hours and due to server load we needed to terminate the process.

The current implantation of Gibbs sampling canâĂŹt be fully distributed because each iteration need the values of parameters theta and gamma from previous iteration. Where both parameters are re-calculated at the end of each iteration. Giving the fact that Gibbs sampling can provide more accurate result, further work on heuristics on Gibbs sampling can be done to converge faster and consume less memory.

5 Conclusion

During this project we familiarised ourselves with topic modelling field of machine learning and studied one of the the most popular algorithms: Latent Dirichlet Allocation. We have implemented two versions of LDA: Gibbs sampling and Online Variational Bayes for Apache Flink data processing engine. LDA is considered to be one of the most easiest algorithms in topic modelling, however it still requires strong knowledge and understanding on probability theory and linear algebra. In our implementation of Online Variational Bayes we were following the example of the same algorithm implemented for Apache Spark system. The implementation is done in Java programming language and uses different helping libraries for calculations: The Apache Commons Mathematics, ScalaNLP and FLink ML library.

We have evaluated our implementations on different datasets and with different input parameters. The test results showed that Online Variational Bayes is faster than Gibbs Sampling. As for the evaluation of Spark vs Flink, in our test case Spark outperformed Flink. The primarily reason for this that Spark's version of LDA is well optimized and it uses one common ScalaNLP library for calculations.Flink showed good results on small datasets, however it performed poorly on big ones. We also count this as optimization problem and see it as the main point of improvement and future work.

References

[1] 20 newsgroups. `http://qwone.com/~jason/20Newsgroups/`, 2008. [Online; accessed: 2008-01-01].

[2] Large scale topic modeling: Improvements to lda on spark. `https://databricks.com/blog/22/large-scale-topic-modeling-improvements.html`, 2015. [Online; accessed: 2015-09-22].

[3] Wikipedia:database download. `https://en.wikipedia.org/wiki/Wikipedia:Database_download`, 2015. [Online; accessed: 2015-11-11].

[4] Apache flink home page. `https://flink.apache.org/`, 2016. [Online; accessed: 2016-01-27].

[5] Apache spark home page. `http://spark.apache.org/`, 2016. [Online; accessed: 2016-01-27].

[6] Flinkml - machine learning for flink. `https://ci.apache.org/projects/flink/flink-docs-master/apis/batch/libs/ml/index.html`, 2016. [Online; accessed: 2016-02-02].

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(rge 1):5228–5235, 2004.

[10] M. Hoffman, F. R. Bach, and D. M. Blei. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pages 856–864, 2010.

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10:10–10, 2010.