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 [2] and Apache Flink [1], which are currently becoming more and more popular replacing and utilizing MapReduce programming model [5]. 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 [8]. 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 [3], however currently it provides users with only some simple algorithms.

This paper discusses the implementation of Latent Dirichlet Allocation (LDA) [4], 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) [4] 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. In other words, 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 for document.
- *Gamma*. It represents words distribution for 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.

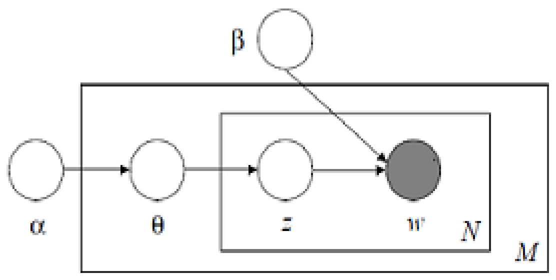

Figure 1: Graphical model representation of LDA. The boxes are "plates" representing replicates. The outer plate represents documents, while the inner plate represents the repeated choice of topics and words within a document. [4]

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.[6]

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 [7]. The algorithm is shown at the Figure 2.

```
Define $\rho_t \triangleq (\tau_0 + t)^{-\kappa}$
Initialize $\boldsymbol{\lambda}$ randomly.
for $t = 0$ to $\infty$ 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. [7]

It follows the original Variational Bayesian, however it splits the the corpus into mini-batches,which theoretically allows corpus size *M* 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 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 implement Online LDA on Apache FLink next steps are required:

1. Initialize the model with prior parameters. *alpha* = 1/K; *beta* = 1/K; *kappa* = 0.51; *tau* = 1024; *miniBatchFraction* = 0.05; *gammaShape* = 100; *gamma matrix K x V* with random parameters;

2. Submit miniBatch of documents size *M* * *miniBatchFraction* to the expectation step of VB.

3. For the size of miniBatch calculate *Theta* matrix using Dirichlet expectation.

4. For each document calculate *gamma* vector and *theta* matrix. Here we utilize Flink architecture and use flatMap function.

5. Reduce the *gamma* and *theta* results from each document to achieve miniBatch results.

6. Update the global *gamma* matrix of probabilities with the batch result.

With the utilizing Flink distributed architecture we avoid iterating over all documents, which the main benefit of using distributed data processing system such as Flink.

4 Experimental evaluation

4.1 Data preparation

4.2 Online LDA vs Gibbs sampling

4.3 Flink vs Spark

5 Conclusion

References

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

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

[3] 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].

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

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

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

[7] 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.

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