Implementing Latent Dirichlet Allocation on NASA ADS

Internship report submitted to the Industrial Engineering and Operations Research Department of Columbia University in the City of New York

August 2016

Tarun Ruchandani
UNI: TR2456

Internship Institution: Harvard University - Center for Astrophysics
Advisor: Dr. Michael Kurtz
06/01/2016 – 08/31/2016

Table of Contents

1. Introduction

This summer I have had the most excellent opportunity to intern at Harvard Center for Astrophysics, under the able guidance of Dr. Michael Kurtz. I have worked on building a topic model – a statistical model for discovering the abstract "topics" – on NASA ADS.

1.1 About NASA ADS

The NASA Astrophysics Data System Abstract Service (hereafter ADS) is a central facility of bibliographic research in astronomy, invented by Dr. Michael Kurtz. Roughly 50,000 scientists use the ADS almost daily including essentially every working astronomer, and a few tens of thousands of physicists and geophysicists. About 250,000 scientists use it to download a couple of articles per week or month, and a few million individuals use it occasionally, often via Google or Wikipedia links. The ADS is also heavily used by dozens of data archivists and librarians who use it as the reference bibliographic system connecting publications with the digital resources which they curate.

2. Project description

Dr. Kurtz's PhD student, Kriste Krstovski, concluded his thesis in May 2016 on using LDA (explained in section 2.1) topic modeling on ADS XML text collection of every refereed Physics, Astronomy, and Geophysics written in the last 10 years, wherein he used his own software. Since he began, Apache Spark had been released with a state-of-the-art LDA capability, which made topic modeling investigations much easier.

This project aims to build an LDA space on ADS literature on Apache Spark, and understand the ontological graph of astrophysics.

2.1 About LDA

Latent Dirichlet Allocation (LDA) is a topic model which infers topics from a collection of text documents. LDA can be thought of as a clustering algorithm as follows:

- Topics correspond to cluster centers, and documents correspond to examples (rows) in a dataset.
- Topics and documents both exist in a feature space, where feature vectors are vectors of word counts (bag of words).
- Rather than estimating cluster using a traditional distance, LDA uses a function based on a statistical model of how text documents are generated.

The generative process for LDA corresponding to joint distribution of hidden and observed variables is detailed in Appendix.

2.2 Why LDA on Apache Spark?

Apache Spark provides data processing around map-reduce type architectures, in ways that are a lot faster than the traditional approach with tools such as Hadoop. Spark generalizes the map-reduce computational concept, and not require the step of writing data back to disk, but stores data in-memory via Resilient Distributed Dataset (RDD). This computational method is particularly well suited to machine learning. ADS servers provide cluster computing capabilities, and with the requirement of performing topic modeling on 12 million documents, Apache Spark best suits the needs.

2.3 Spark LDA parameters

LDA supports different inference algorithms via setOptimizer function. EMLDAOptimizer learns clustering using expectation-maximization on the likelihood function and yields comprehensive results, while OnlineLDAOptimizer uses iterative mini-batch sampling for online variational inference and is generally memory friendly.

The LDA function takes in a collection of documents as vectors of word counts and the following parameters:

- k: Number of topics (i.e., cluster centers). This is set to 20 in development.

- optimizer: Optimizer to use for learning the LDA model, either Expectation-Maximization (EM) LDAOptimizer or OnlineLDAOptimizer. We use EMLDAOptimizer.

- docConcentration: Dirichlet parameter for prior over documents' distributions over topics, also known as alpha. Set to default of (50 / k) + 1 for EMLDAOptimizer.

- topicConcentration: Dirichlet parameter for prior over topics' distributions over terms (words), also known as beta.. Set to default of 0.1 + 1 for EMLDAOptimizer.

- maxIterations: Limit on the number of iterations. Set to 500.

- checkpointInterval: If using checkpointing (set in the Spark configuration), this parameter specifies the frequency with which checkpoints will be created. Set to default of 10.

3. Duties and responsibilities

My overall project responsibility included building preliminary end-to-end data pipelines for a few hundred documents from more than 5 different journals. The pipeline is illustrated in following diagram.

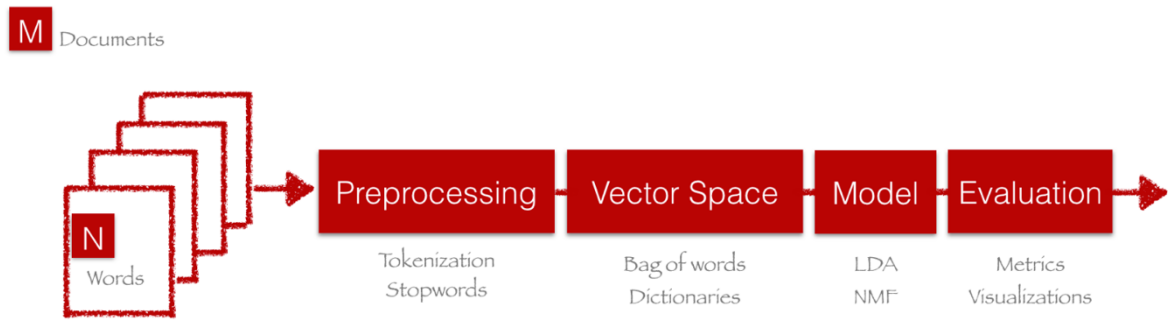

Fig 1. Data pipeline for topic modeling

3.1 Implementation

Preprocessing

The papers are extracted using Python tool for ADS.

The following data cleaning steps crucial for generating a useful topic model are performed:

- Tokenizing: converting a document to its atomic elements. In this case, we are interested in tokenizing to words.
- Stopping: removing meaningless words. Certain parts of English speech, like conjunctions (“for”, “or”) or the word “the” are meaningless to a topic model. These terms are called stop words and need to be removed from our token list. The definition of a stop word is flexible and the kind of documents may alter that definition. We use the stop_words package from Pypi.
- Stemming: merging words that are equivalent in meaning. We implement the Porter stemming algorithm, using the Porter Stemmer module from NLTK.

Vector Space

To generate an LDA model, we need to understand how frequently each term occurs within each document. To do that, we construct a document-term matrix with a package called textmining. For that, we create dictionary of tokens, assigning a unique integer id to each unique token while also collecting word counts, and a list of vectors equal to the number of documents wherein each document vector is a series of tuples.

Model

We initialize and fit the LDA model using PySpark extension of Apache Spark. To do this, we have to choose the number of topics, which we have chosen has 20 for a corpus of 577 papers we are working with for development. Rest of the parameters are set as discussed in section 2.3

Evaluation

The document-word matrix output from LDA is an NxM matrix, N being the number of documents, and M being the length of dictionary. We now aim to calculate 'distance' between each of these documents, wherein the 'distance' is calculated by Jensen-Shannon (JS) Divergence between each document vector of topic distribution, and every other such document vector, thus creating an NxN matrix of 'distances' between N documents.

JS Divergence is given by:
JSD(P||Q) = 0.5*D(P||M) + 0.5*D(Q||M)

where M = 0.5*(P+Q),
and D(P||Q) is Kullback-Leibler Divergence given by:
D(P||Q) = sum over i [(P(i))*log(P(i)/Q(i))]

A similarity matrix is constructed out of this divergence matrix by performing 1 – Dij (Dij being the JS Divergence measure between document i and j).

JS Divergence was chosen over KL Divergence because of the following reasons:

- Range of JS Divergence values is [0,1] as opposed to KL's [0,infinity)
- Computationally less expensive

Visualization

Before we visualize the results from JS Divergence between documents, we perform community detection on it, to better visualize the ontological graph. Communities are groups of nodes within a network that are more densely connected to one another than to other nodes.

This community-detected graph is generated using the Mapequation software, which generates a .tree file corresponding to the best hierarchical partitions of the graph nodes. Sample output is shown in Appendix.

Mapequation internally uses Rovall-Bergstrom algorithm for community detection. We also use the Community package in Python to compare outputs. We then feed the .tree file to Networkx module which generates graph together with Mathplotlib.

Creating Docker Container

In order to scale the entire above mentioned pipeline on ADS servers, Docker containers are being built around entire pipeline to be deployed onto production machines.

4. A typical day during this internship

My day started at 8:30 AM, with reading and learning theoretical concepts. This included, but not limited to, academic papers suggested by Dr. Kurtz, watching Professor Dan Jurafsky's course on Natural Language Processing, learning new concepts such as Louvain method (what the Rovall-Bergstrom algorithm follows at its core), and JS Divergence.

I then attended the daily 10 AM scrum calls with the ADS team, wherein each team member discussed their what they were going to achieve that day. Post that, I worked on implementing my learnings, and building data pipelines discussed in section 3.

I met with relevant teammate(s) individually in evening between 4 – 6 PM for help on coding and architecture problem(s) I encountered. I then spent the final half hour everyday documenting all of the learnings, and updating code on Github.

I also attended the weekly status update meeting on Thursdays with Dr. Kurtz, and also with ADS group. I attended the weekly Summer Symposiums which featured scientific talks by resident and visiting researchers, also on Thursdays. These symposiums served as good networking opportunity for meeting with fellow summer interns.

5. Future project work

Following are the potential next steps for the project:

- Extend to a larger corpus by building Docker Container and deploying code to ADS production server.
- Persistent storage for output of LDA, JSD matrix, .tree file, and graph.
- Optimization of:
    - JS Similarity matrix generation
    - .tree file generation
- Build a recommendation engine to use the LDA space to classify the readers.

6. Skills and qualifications acquired

Through this internship experience, I have acquired the following skills and qualifications:

- Operating knowledge of topic modeling,
- Advanced Python programming skills,
- Understanding of process of scientific inquiry,
- Working in development and production regimes in parallel,
- Understanding and working knowledge of large scale information systems,
- Communicating technical concepts to non-technical audience

7. Contribution of this internship to my career plans

I began this internship with the motivation and goal of pursuing a doctoral program at the end of my Masters program at Columbia University, and it has indeed prepared me for a potential pursuit of doctoral studies.

This internship experience has trained me in finer concepts of Machine Learning, and in Information Science and Engineering in particular, and by applying my knowledge in an academic setting, provided me with an experience of the best of both the academic and non-academic worlds. Hence, it has also prepared me for a career in Data Science.

8. Relationship to my coursework at Columbia

My coursework in STATW4400 Statistical Machine Learning at Columbia University has been immensely helpful in understanding the details of LDA. In particular, the topics under Unsupervised Learning viz. EM algorithm, Gibbs Sampling, KL Divergence directly helped me in my project understanding and implementation. Additionally, IEORE4101 Probability Models

helped me understanding the underlying probability theory for LDA; IEOR E4004: Intro to OR: Deterministic Models together with implementing community detection technique algorithm discussed in section 3.1 has helped me deepen in knowledge in graph theory. Also, inspired by this internship, I intend on studying Bayesian Data Analysis course this Fall semester.

9. Acknowledgements

I am profoundly grateful to my advisor Dr. Kurtz for pointing me to right papers to read, right people to reach out to for help, encouraging me to implement all of the concepts that I learn, and for inculcating in me the enthusiasm towards research. I am grateful to him for all this, and much more.

I am pleased to have had very supportive and wonderful teammates to work with on the ADS group: Alberto Accomazzi, Alexandra Holachek, Carolyn Grant, Donna Thompson, Edwin Henneken, Jonathan N. Elliott, Roman Chyla, and Zachary Kiihne. They made me look forward to coming to work every morning.

Particular thanks goes to Donna Wyatt for facilitating all of the administrative tasks, helping me get acquainted with CfA, and informing me of various social events.

I would also like to thank Kriste Krstovski, for helping me kick start the project in right direction.

And last, but not the least, I am grateful to Prof. John Cunningham at Columbia University, whose course STATW4400 Statistical Machine Learning during Spring 2016 semester sparked my interest in Machine Learning.

10. References

1. Probabilistic Topic Models
https://www.cs.princeton.edu/~blei/papers/Blei2012.pdf

2. Introduction to Topic Modeling in Python
http://chdoig.github.io/pygotham-topic-modeling/#/

3. Natural Language Processing course by Dan Jurafsky and Christopher Manning
http://online.stanford.edu/course/natural-language-processing

4. Fast unfolding of communities in large networks
http://arxiv.org/pdf/0803.0476.pdf

5. The NASA Astrophysics Data System: Overview
http://ads.harvard.edu/pubs/A+AS/2000A+AS..143...41K/ds1780.pdf

6. Apache Spark Latent Dirichlet allocation (LDA)
http://spark.apache.org/docs/latest/mllib-clustering.html#latent-dirichlet-allocation-lda

7. MapEquation
http://www.mapequation.org/code.html

8. LDA with Python
https://rstudio-pubs-static.s3.amazonaws.com/79360_850b2a69980c4488b1db95987a24867a.html

Appendix

Generative process for LDA

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d) \left(\prod_{n=1}^{N} p(z_{d,n} | \theta_d) p(w_{d,n} | \beta_{1:K}, z_{d,n}) \right)$$

Fig 2. Generative Process for LDA

$ß_{1:K}$ are topics,
$ß_k$ is a distribution over the vocabulary,

θ_d, are topic proportions for the d^{th} document,
$\theta_{d,k}$ is the topic proportion for topic k in document d,

z_d are topic assignments for the dth document,
$z_{d,n}$ is the topic assignment for the nth word in document d,

w_d, are observed words for document d,
$w_{d,n}$ is the nth word in document d, which is an element from the fixed vocabulary.

Sample output of LDA

Topic-word distribution:

```
[[  2.72436509e-06   2.72436509e-06   2.72708945e-03 ...,   2.72436509e-06
    2.72436509e-06   1.36490691e-03]
 [  2.29518860e-02   1.08771556e-06   7.83263973e-03 ...,   1.08771556e-06
    1.08771556e-06   1.08771556e-06]
 [  3.97404221e-03   4.96135108e-06   2.98177200e-03 ...,   4.96135108e-06
    4.96135108e-06   4.96135108e-06]
 ...,
 [  3.90980357e-02   1.70316633e-03   4.42279319e-03 ...,   3.39953358e-06
    3.39953358e-06   3.39953358e-06]
 [  2.39373034e-06   2.39373034e-06   2.39373034e-06 ...,   2.39373034e-06
    2.39373034e-06   2.39373034e-06]
 [  3.32493234e-06   3.32493234e-06   3.32493234e-06 ...,   3.32493234e-06
    3.32493234e-06   3.32493234e-06]]
```

Document topic distribution:

```
[[ 0.01347826  0.07869565  0.00043478 ...,  0.01347826  0.00043478
   0.02217391]
 [ 0.06594203  0.29057971  0.00072464 ...,  0.02971014  0.00072464
   0.18188406]
 [ 0.02552301  0.07991632  0.00041841 ...,  0.00041841  0.00041841
   0.00041841]
 ...,
 [ 0.00063694  0.12165605  0.28726115 ...,  0.00063694  0.00063694
   0.03248408]
 [ 0.13946188  0.04977578  0.01838565 ...,  0.00044843  0.00044843
   0.08116592]
 [ 0.00263158  0.29210526  0.00263158 ...,  0.02894737  0.00263158
   0.00263158]]
```

Connecting PySpark with iPython notebooks:

```
export SPARK_HOME="/spark/location"
cd $SPARK_HOME
IPYTHON_OPTS="notebook" ./bin/pyspark
```

Sample Output of Mapequation .tree file

```
# 'JS_output2.txt mymaps/ -N 10 --tree --bftree -z' -> 23 nodes and 253 links partitioned in 0s from codelength 4.521169199 in one level to codelength 4.521169199 in 2 levels.
# path flow name node:
1:1 0.0480789 "21" 21
1:2 0.047523 "3" 3
1:3 0.046965 "2" 2
1:4 0.0462474 "17" 17
```

Each row (except the first one, which summarizes the result) begins with the multilevel module assignments of a node. The module assignments are colon separated from coarse to fine level, and all modules within each level are sorted by the total flow (PageRank) of the nodes they contain. Further, the integer after the last colon is the rank within the finest-level module, the decimal number is the amount of flow in that node, i.e. the steady state population of random walkers, the content within quotation marks is the node name, and finally, the last integer is the index of the node in the original network file.