1. **Abstract**

Latent Dirichlet Analysis (LDA) is a method used to model the generative process of creating discrete data such as text corpora. LDA can take a large amount of data and create descriptions of that data. This method reduces the size of the data to these short descriptions while still maintaining the relationships necessary to carry out various inferences. For this project, we implemented the LDA algorithm presented in the paper, “Latent Dirichlet Allocation”, written by David M. Blei, Andrew Y. Ng, and Michael I. Jordan. The implementation of the algorithm is used to analyze text of State of the Union Addresses. In addition, it is used to create clusters of movies based on user rating data.

1. **Background**

The algorithm assumes a generative process for the creation of documents in a corpus D using N words. The topic for a particular word, $z\_n$, is modeled as Multinomial($\theta$), where $\theta \sim Dir(\alpha)$. Then, each word, $w\_n$, is chosen from a multinomial probability that is conditioned on the topic $z\_n$, P($w\_n | z\_n, \beta)$. (Blei et al., pg. 996)

LDA is most well-known for its application in the analysis of text data. It is used to create topics for documents, classify documents based on these topics and to determine which documents are similar to one another. The algorithm can also be used in other problems that have a similar structure to the document generating method described above. For example, in Blei et.al, 2003, the authors used a data set where web site users provide information about movies they enjoy. In this example, the users are analogous to the documents and their preferred movies are “words”. Topics can be found by determining similar movies.

There are multiple alternative algorithms that can be used for text analysis. These include the term-frequency inverse-document-frequency matrix, latent semantic indexing (LSI) and the probabilistic latent semantic indexing (pLSI) model. In Section 7 of the paper, the authors present the results of document modeling and document classification. In this section, the authors’ results show that their implementation of LDA performs better than competing methods in terms of perplexity measure, where better generalization performance is defined by a smaller perplexity measure.

Perplexity measure is defined as:

\begin{center}

Exp(-\frac{\sum\_{d=1}^M log p(\textbf{w}\_d}{\sum\_{d=1}^M N\_d})

\end{center}

In addition, this section demonstrates that other methods are prone to overfitting. As the number of topic increases, some of the alternative algorithms induce words that have small probabilities. This occurs because the documents in the corpus are divided into more collections. This can result in the perplexity measure becoming very large for these alternative algorithms. The problem comes from the requirement that the topic proportions in a future document must be seen in at least one of the training documents. On the other hand, LDA does not have this overfitting problem. For more detail, see Section 7 of Blei et al.

A disadvantage of LDA is that exact inference of the posterior is impossible as a result of intractability. Thus, it is necessary for the user to implement an approximating technique such as variational inference, a Gibbs sampler, or another technique. The approximation of the posterior can be computationally intensive and also time-consuming.

1. **Implementation**
2. **Code**

The LDA algorithm was implemented using the Python programming language in the Jupyter notebook environment. The LDA function requires four arguments. The number of topics, k, must be specified. In addition, the output from the make\_word\_matrix is necessary in the LDA function. This function takes the corpus as an input and returns three things: a matrix where each word in the document is a row and the columns are the unique words in the corpus, the list of words unique to the corpus, and the number of documents in the corpus. The third argument necessary for the LDA function is the tolerance for convergence. Finally, an indicator value of the form of the corpus documents is required for the LDA. This value is 0 if the documents are a list of strings, and the value is 1 if the documents are just one long string. In addition, we created a function to return a specified number of words for each topic.

1. **Testing and Base Cases**

In addition to the implementation described above, various checks have been included in the function to prevent incorrect arguments. One check is that the number of topics specified by the user must be greater than one. It is not possible to have zero or less than zero topics, and one topic would just be described by the entire document. In addition, there is a check in place to ensure that the corpus is not empty and that each element of the corpus is a string. This prevents the user from receiving an error that the input is not a string. Lastly, there are checks to ensure that the tolerance is greater than zero and that the entry for needToSplit is either zero or one. These checks print messages that inform the user of why their input was problematic.

1. **Tests for Correctness**

A generative model check was utilized in order to evaluate the functionality of the algorithm. A random number was generated in order to determine the number of documents from a Poisson(5) distribution, and the resulting number was six. Then, for each of the six documents, a random number of words for each document was drawn from a Poisson(5) distribution. These documents had between 2-6 words. Random values were used to create a $\beta$ vector, and the $\alpha$ vector was set to be 50/k, where k is the number of topics and k = 3. The 50/k value was chosen based on the results shown in Gregor Heinrich’s “Parameter estimation for text analysis”. (Heinrich, 2008, 24). The corpus was defined to have five unique words that are not stop words.

The set-up described in the previous paragraph was used to determine which of the words were in each of the documents. This process followed the generative model described in Blei et. al’s paper. (Blei et al, 2003, 996). Once the documents were generated, our LDA implementation was used on this corpus to estimate the $\alpha$ and $\beta$ values.

\*make a chart of the results\*

MSE for alpha = 0.06

MSE for beta = 0.63

Thus, the values returned by the algorithm are relatively close to the ones used to generate the corpus. It is not surprising that the mean squared errors for the $\beta$ vector is larger because there are more elements in the beta vector. The $\alpha$ vector converges faster than the $\beta$ vector, so this could be the reason for the larger MSE for $\beta$. (EXPAND HERE?)

1. **Profiling Code and Speed Up**

We profiled the initial code by implementing the algorithm on a corpus of four of President Clinton’s State of the Union Addresses. It was found that the main bottleneck in the algorithm was the maximization step of the algorithm. About 75% of the time of the LDA algorithm was spent on the maximization step. In the maximization step, the $\alpha$ update occurs in a while loop, and the $\beta$ update uses four *for* loops. The expectation step also slowed down the code, but only accounted for approximately 13% of the time of the LDA implementation. This mostly likely is a result of having fewer *for* loops than the maximization step.

We identified inefficiencies in the matrix construction function and implemented a storage system in order to avoid the unnecessary repetition of some calculations. In addition, during the initial coding of the algorithm, care was taken to avoid for loops and use vectorization where possible. Just in time compilation (Numba) was also added to the code for the algorithm implementation.

Finally, with the assistance of Professor Chan, the code was adapted so that Cython could be used. It was first necessary to adjust the code to prevent ragged arrays because Cython does not easily work with sparse or ragged arrays. The phi and the corpus matrix both initially used ragged arrays. To fix this, the largest dimension of the matrices was found, and the rest of the matrices were filled in with zeros such that the dimensions matched. In addition, the digamma function from the Cython GSL package replaced that from the scipy package. The final change was using *cdef* to define the inputs to the functions and the variables used within.

Timing for Toy Example

Method Time (Total)

Nothing 4.36 milliseconds

Numba 3.02 seconds

Cython 4.16 millliseconds

Timing for All Eight Clinton SOTU

Method Time (Total)

Nothing 4 min., 54 sec.

Numba 4 min., 51 sec.

Cython 1 min. 48 sec.

As the tables above show, the Cython provided the fastest implementation of the algorithm. For the smaller example, Numba performed the slowest and was only marginally faster in the State of the Union example. According to the Numba documentation, there is compilation time associated with Numba. In addition, it would be faster if Numba was used in No Python mode. However, this was not possible given the use of the numpy package. For the State of the Union example, the Cython code provides approximately a three time speed up over the other implementations.

1. **Results**
2. **Areas of Improvement/Further Research**

One area in which we would like to expand our research and improve our project is in the approximation of the posterior. In the implementation for this project, variational inference was utilized to estimate the intractable posterior distribution. However, there are other methods that can be used to estimate the posterior. These include Gibbs sampling and Metropolis-Hastings. It is possible that these approximations might increase the speed of the implementation.

Another area to expand on is the determination of the number of topics. This is an ongoing area of debate and research at the moment. The R implementation of LDA provides different measures to determine the number of topics, and these are something that we can look into adding to our code.

In addition, we want to continue to use the algorithms to situations beyond text corpora. The movie data example in this paper is one example of how LDA can be used for something beyond text data. The algorithm can be used for any situation that has the same structure as text corpora.

**References**

1. Colorado Reed, *Latent Dirichlet Allocation: Towards a Deeper Understanding*, January 2012, o

bphio.us/pdfs/**lda**\_tutorial.pdf.

1. David M. Blei, Andrew Y. Ng, and Michael I. Jordan, *Latent Dirichlet Allocation*, Journal of

Machine Learning Research 3, 2003, pg. 993-1022.

1. Max Sklar, *Fast MLE Computation for the Dirichlet Multinomial*, May 2014.
2. F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872
3. Thomas P. Minka, *Estimating a Dirichlet Distribution*, www.msr-waypoint.com.