Pacific Northwest
NATIONAL LABORATORY

PNNL-XXXXX

Developing Topic Modeling Visualizations for Large-Scale Dataset Analysis

August 2024

Maxwell A. Steele

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from
the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062
www.osti.gov
ph: (865) 576-8401
fox: (865) 576-5728
email: reports@osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
or (703) 605-6000
email: info@ntis.gov
Online ordering: http://www.ntis.gov

PNNL-XXXXX

Developing Topic Modeling Visualizations for Large-Scale Dataset Analysis

August 2024

Maxwell A. Steele

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Abstract

Topic modeling is an unsupervised machine learning technique used to understand and categorize large corpora of data. In the context of machine learning, topic modeling provides a methodical way to process and analyze vast amounts of unstructured text data into a format suitable for further tasks such as classification and clustering. We describe two dynamic topic modeling visualizations to support a Latent Dirichlet Allocation (LDA) algorithm, integrated within an interactive user interface (UI) for a React application. The UI includes two key visual tools, both supported by a custom Application Programming Interface (API) built within a Flask backend using Python. The first tool is an implementation of the pyLDAvis Python library to create a drilldown visualization which can process and query a scalable number of LDA models through the API, which retrieves these LDA models through AWS Simple Storage Service (S3). The second tool presents a bar chart which illustrates the distribution of articles across an adaptable number of abstract topics. The visualization supports advanced filtering by topic, article, and keyword, and utilizes the API to process and display Wiki Data for each article. These tools transform unstructured text into valuable insights. This scalable integration of natural language processing (NLP) and ML techniques enable users to handle and interpret large datasets (500 GB+) effectively, improving their workflows and decision-making processes.

1.0 Introduction

A fundamental goal for Pacific Northwest National Laboratory (PNNL) is to support and develop innovative data science and artificial intelligence methods to support national security and science missions[1]. The Foundational Data Science group within the National Security Directorate supports these goals by developing human language technologies that enable understanding of complex data.

The cornerstone of developing human language technology is Natural Language Processing (NLP), which serves as a bridge between human communication and digital data. Powered by Machine Learning (ML), NLP allows computers to understand semantic structures, as well as grammar and usage exceptions, to produce accurate and usable results[2]. The objective of this project is to develop ways to visualize the outputs of said algorithms, such that users can interact with these findings and incorporate them into their workflows.

1.1 Topic Modeling

Topic Modeling, an NLP technique, is a type of statistical modeling used to identify topics or themes within collections of documents. Topic Modeling forms groups of words that represent distinct topics within the documents[3].

Topic Modeling is an unsupervised machine learning algorithm, meaning it is compatible with unlabeled data sets. Thus, an advantage of a topic modeling procedure is that it can discover hidden patterns in data without the need for human intervention. The most popular methods for topic modeling include Latent Dirichlet Allocation (LDA), Non-negative Matrix Factorization (NMF), and BERTopic (Bidirectional Encoder Representations from Transformers-based Topic Modeling)[4]. This project will utilize LDA.

1.2 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a generative probabilistic topic modeling technique used to estimate two parameters of the model: topic-term distribution and document-topic distribution. Topic-term distribution analyzes the distribution of a topic over all possible words in the vocabulary (collection of unique words). Similarly, document-topic distribution considers the distribution of some document over all possible topics. It is important to note that though the topics in the collection of documents are unknown, we assume that the text in the model are generated based on these unknown topics.

Once the LDA model is trained and evaluated, the results must be interpreted to gain insight from the topics. We present two use cases of visualization tools that aim to assist users with this complicated task within the context of a web application.

---

[1] “Foundational Data Science.” PNNL. Accessed August 5, 2024. https://www.pnnl.gov/foundational-data-science.
[2] What is NLP? - natural language processing explained - AWS. Accessed August 5, 2024. https://aws.amazon.com/what-is/nlp/.
[3] What is Topic Modeling? A guide to text mining tools and methods. Penn
[4] “Supervised vs Unsupervised Learning.” IBM, April 24, 2024. https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning.

2.0 Methodology

Interpreting the results of Topic Modeling is a fundamentally challenging task, especially when dealing with a large set of documents. Latent Dirichlet Allocation (LDA) is typically used with thousands of documents, of which LDA can produce dozens or hundreds of topics, which are modeled as distributions of thousands of terms[1]. This project utilizes two key tools for two separate visualizations to interpret the results of Topic Modeling: pyLDAvis and Vega. These visualizations are incorporated into a React JS application. React is a JavaScript library that powers user interfaces[2].

2.1 pyLDAvis

The first visualization uses pyLDAvis, a web-based interactive visualization of topics calculated using LDA (Figure 1). It is available as a Python library for open-source use, which is how it was utilized in this case. Because the visualization will be utilized in a web application with a separate data source, we must use an application programming interface (API) within a full-stack context. This means we have a frontend (user interface) and a backend (API). We will build our API with Flask, a micro web framework for Python.

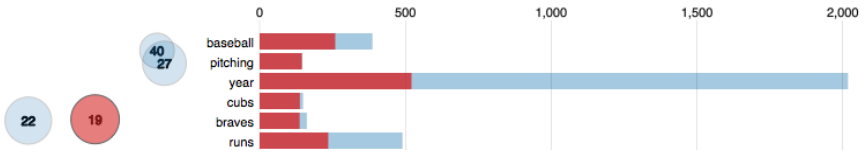
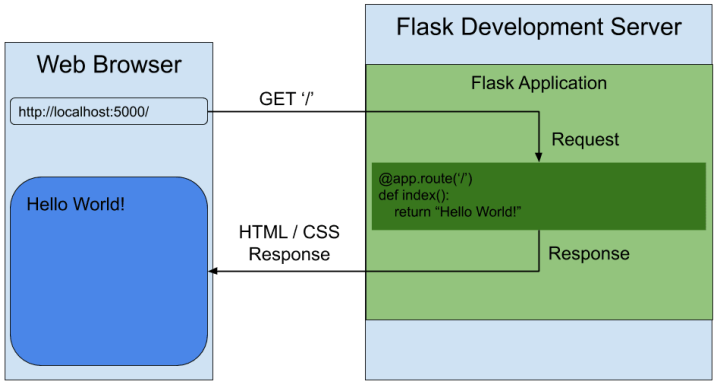


Figure 1: pyLDAvis visualization. Each circle represents a topic. If a topic is closer to another, those topics are similar. Relevant terms are shown on the right[3].

2.1.1 Application Programming Interface Implementation

To add visualization functionality for our app, we will wrap the following processes in a route (a designated path or endpoint in a web application that corresponds to a specific function in the API).



[1] Visualization of the inferred Dirichlet distributions modeling the... | download scientific diagram. Accessed August 5, 2024. https://www.researchgate.net/figure/Visualization-of-the-inferred-Dirichlet-distributions-modeling-the-three-free-parameters_fig3_335796636.
[2] LDAvis: A method for visualizing and interpreting topics. Accessed August 5, 2024. https://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf.
[3] "Pyldavis." PyPI. Accessed August 5, 2024. https://pypi.org/project/pyLDAvis/.

Figure 2: Anatomy of a route in Flask. In this case, it returns “Hello, World!” to the browser.[1]

To create a pyLDAvis visualization, we must have access to three components: a trained LDA model, corpus (set of documents the LDA model was trained on), and the vectorizer associated with the model. The vectorizer is used to convert input text data (the data in the corpus) to numerical data that the model is compatible to work with.

To access these components, the project utilizes Amazon Web Services (AWS) resources, such as Simple Storage Service (S3) to host these models on the cloud. To access them in our Flask API, we use the boto3 library, which is the AWS Software Development Kit (SDK) for Python. This provides an API to query S3 and fetch the necessary components[2].

```
s3_client = boto3.client(
    's3',
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY
)

def fetch_s3_object(self, key):
        return s3_client.get_object(Bucket=BUCKET_NAME, Key=key)['Body'].read()
```

Figure 2: We use a boto3 client to access our S3 bucket—where our LDA models, corpus, and vectorizers are stored on the cloud. A function uses the client to fetch the object from the bucket.

There are many advantages of using cloud computing to power this visualization. For instance, we can scale the number of LDA models hosted, or the number of documents used in our models. If future work requires new models to be stored, or different documents to be utilized, we can easily accomplish this without changing any major components in the app.

After these components have been fetched, we can use the pyLDAvis library to prepare the visualization. It must be processed as HTML and returned to the frontend component.

```
visualization = pyLDAvis.lda_model.prepare(lda, corpus, vectorizer)
html_str = pyLDAvis.prepared_data_to_html(visualization)
return html_str, 200, {'Content-Type': 'text/html'}
```

Figure 3: pyLDAvis provides a method to calculate the visualization data. We must process it as HTML for the browser and return it with a status of 200 (success) from our API.

[1] Kennedy, Patrick. “Developing Web Applications with Python and Flask.” testdriven.io. Accessed August 5, 2024. https://testdriven.io/courses/learn-flask/.
[2] “Boto3 Documentation.” Boto3 1.34.153 documentation. Accessed August 5, 2024. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.

Since we have a scalable number of models, users must be able to choose between different models. Names for the models are both unique and based on the results of the LDA modeling. Additionally, they can be stored in S3 and accessed through the API. These features make model name/label the optimal choice to display to an end user.

```
@app.route('/api/get_model_options', methods=['GET'])
def get_model_options():
   return jsonify({"options": NAMES})
```

Figure 4: a Flask endpoint to return the names of the models. It is returned as JSON (JavaScript Object Notation) to preserve a key-value style.

2.1.2 User Interface Implementation

The visualization is designed as a feature for a React web application. React is a popular JavaScript library used for building user interfaces. React is component-based, meaning the UI can be encapsulated into separate components that manage different behavior for the app, enabling well-organized UI. This visualization is built into a separate component for these reasons.

To access the data from the backend (Flask API), we use a fetch() function to request data from the endpoint in our API. As previously mentioned, we have a scalable number of LDA models to display with pyLDAvis. This means we need the user to clarify which model they are requesting. We can do this with a query parameter which will pass along this information in the fetch request to the API.

```
const encodedModel = encodeURIComponent(model);
        const response = await fetch(`/api/pyldavis_s3?model=${encodedModel}`);
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        const result = await response.text();
        setVisualization(result);
```

Figure 5: the encoded model is passed as a query parameter to the API. The result is stored in a piece of state, which saves the information.

After the model has been returned from the API endpoint to the UI, the user can see pyLDAvis and interact with its information. This includes switching between different topics and viewing relevant, topic-specific terms.

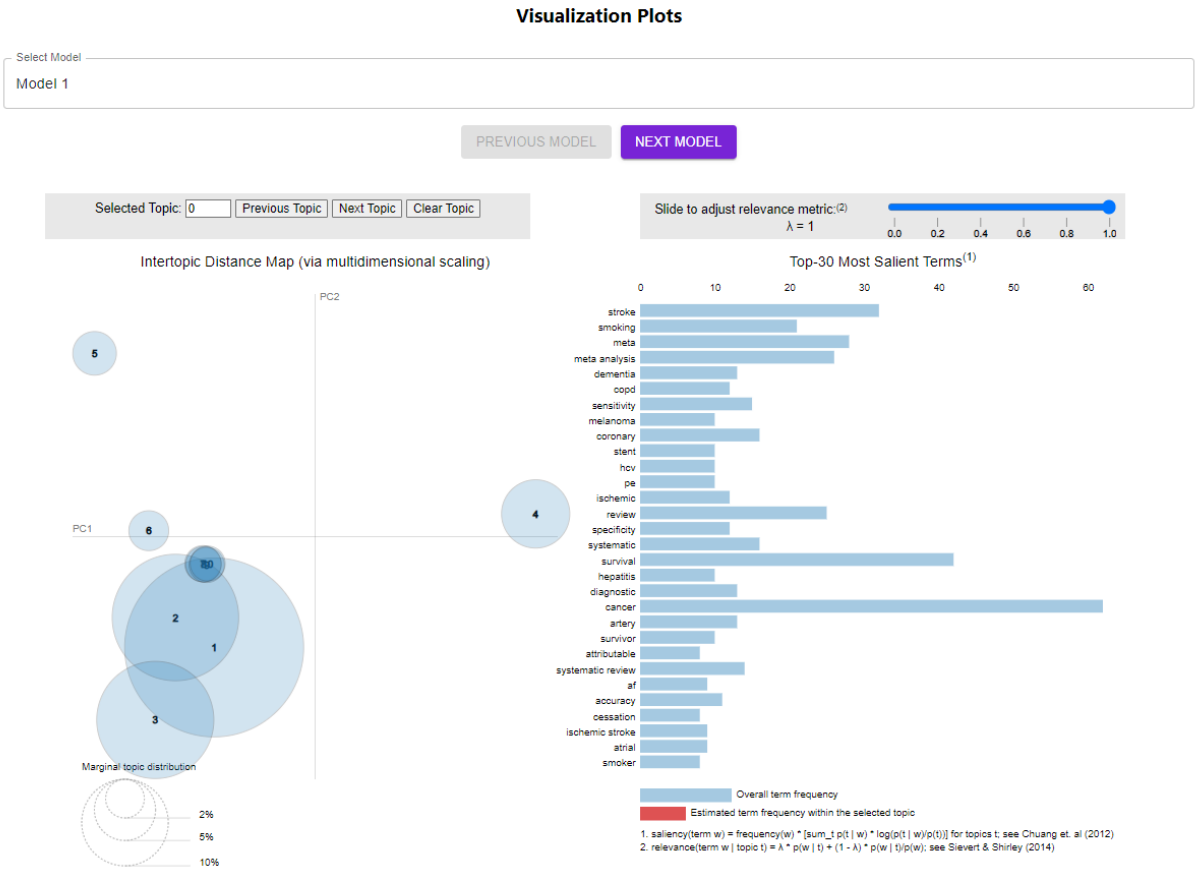


Figure 6: pyLDAvis in the user interface (UI) for the web application. Users can use buttons or dropdown menu to switch between different models.

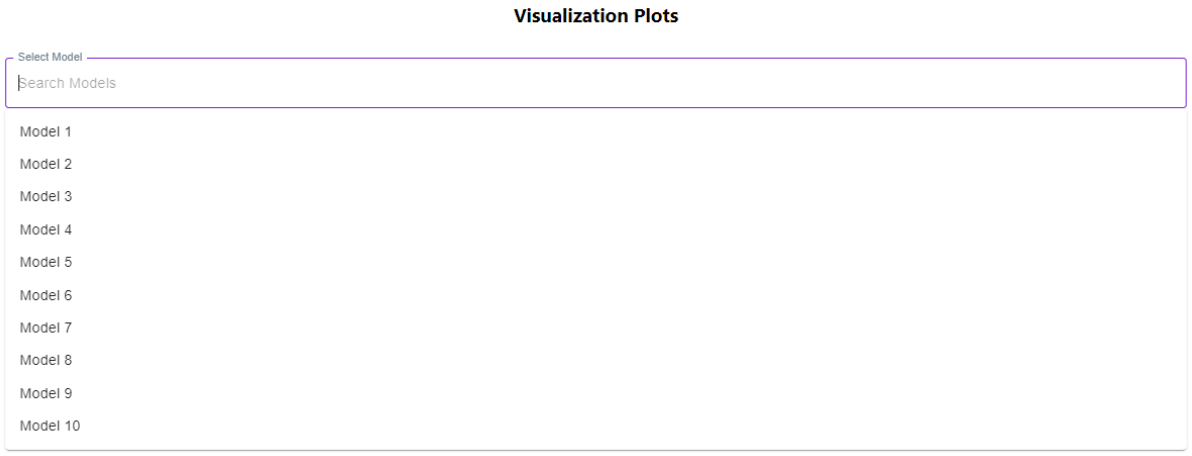


Figure 7: Within the dropdown menu, users can choose between 10 different LDA models (meaning 10 unique pyLDAvis visualizations).

2.2 Vega

Vega is a visualization grammar, which is a language used for creating, saving, and sharing interactive visualization designs. For this project, Vega is used to represent alternate findings from LDA modeling.

2.2.1 Implementation

As mentioned in the introductory section, LDA assigns each document exactly one topic. In other words, we have a set number of topics, and each article is assigned one of these topics. A calculation that follows is the number of articles per topic. This allows users to gain an understanding of the relevance of topics, where more articles under a given topic may point towards higher relevance. Shown below is an implementation of this calculation using Vega.

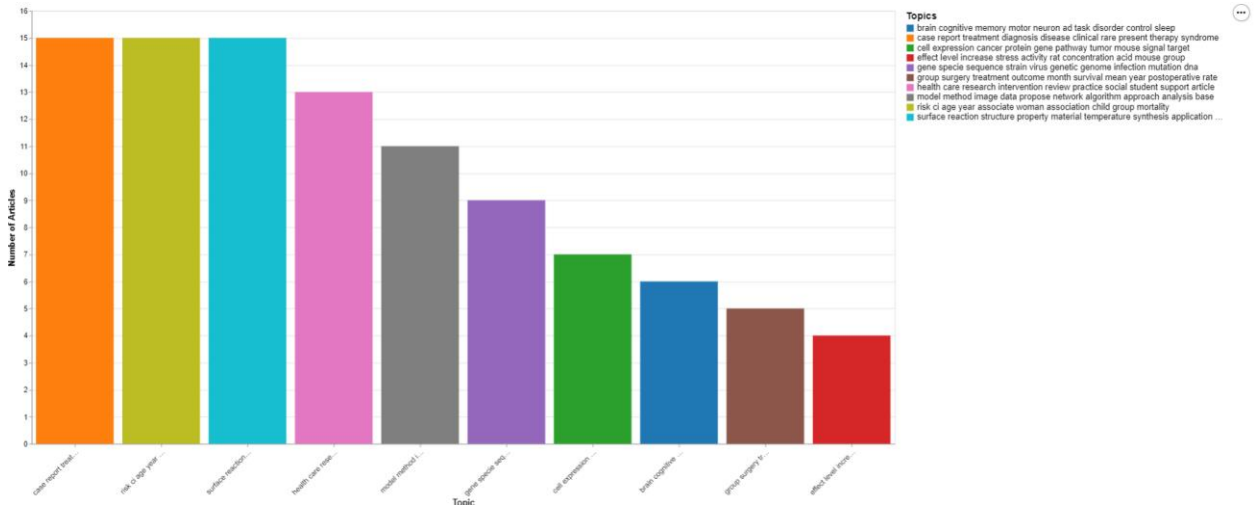


Chart 1: A Vega visualization pairing abstract topics with number of articles under that topic name.

This pairs well with metadata the articles may provide, which, for the user's reference, can be stored in a table underneath this visualization. For this project, we pair advanced filtering tools with the Vega visualization, allowing the ability to sort by topic, article, and/or by keyword. These features are intended to give users a better understanding on individual articles as well as the topics that represent them.

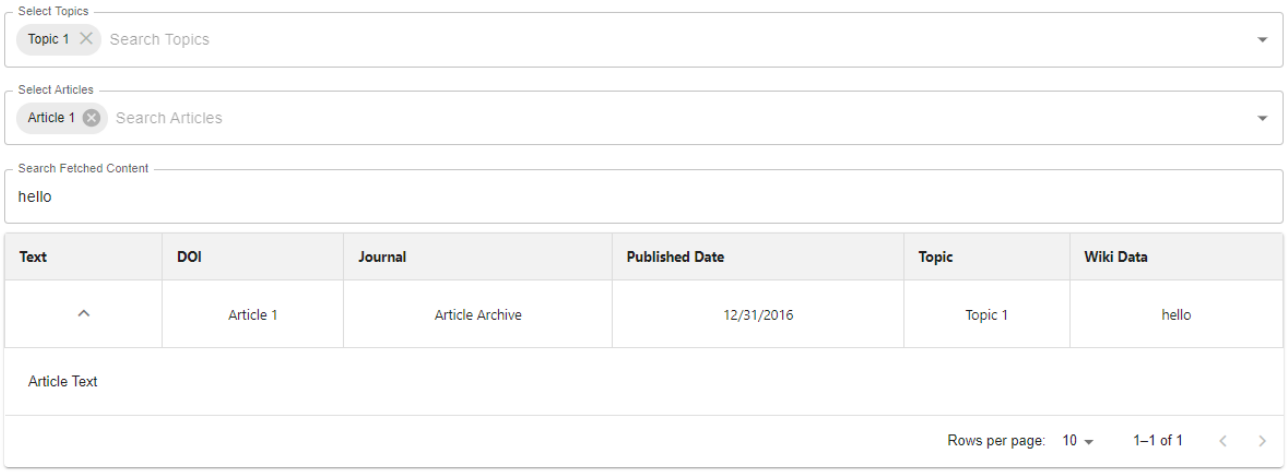


Figure 9: Three dropdown menus allow users to filter a table of articles, which includes identifying metadata. In this case, the user filtered down the sample results to produce one match.

For an example use case, a user can select a topic out of the ten available topics in the "Select Topics" dropdown menu. This will filter the table such that only articles with that associated topic are shown. Then, choosing the "Select Articles" dropdown menu will populate with all articles that fall under the topic the user has selected. If there are multiple topics selected, any article that falls under any of the selected topics will be shown. Selecting the articles within this dropdown will also update the table. Finally, users can type keywords into the "Search Fetched Content" box and the table will immediately update with highlightedmatches.

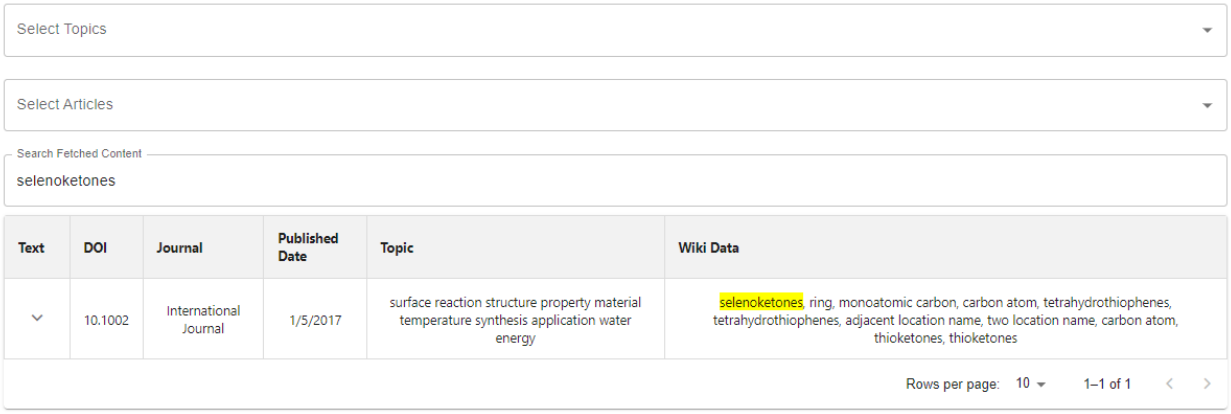


Figure 10: A user searches for "selenoketones" and fetches one matching result from all articles (no other filters present).

3.0 Conclusion and Future Work

Overall, dynamic topic modeling is a computationally expensive yet beneficial endeavor. It can be used to enhance data-driven decision making and strategic planning within various fields to uncover patterns and trends.

In the future, improving the quality and quantity of data for the LDA topic modeling will improve both visualizations. In turn, the application programming interface (API) used by the pyLDAvis topic modeling visualization will be updated to support optimized queries for large amounts of data from AWS services.

The next step of development for the Vega visualization is to create a “drill down” functionality. Visualization drill downs allow users to create hierarchies in visualizations so that users can click on a component of a chart and "drill in" to explore the data within that piece of the visualization. This could support advanced labeling for topics (sub-labels) to provide the user interface with a more nuanced approach to topic modeling at a large scale. Similarly to the pyLDAvis visualization, upgraded infrastructure in the frontend and backend must be provided to support a larger document set.

For both visualizations, improved styling will be applied to improve the overall aesthetic within the app, improving user experience and harmony with the other features included in the application.

Acknowledgments

I am grateful for the opportunity I had to work under several knowledgeable individuals over the course of this internship. A special thanks to my mentor, Marjolein Oostrom for her guidance on this project, and to Abby Reynolds for going above and beyond to assist my work.

This work was supported by Department of Homeland Security Workforce opportunities to Increase Research Engagement and Diversity (WIRED).