# Latent Dirichlet Allocation (LDA) Topic Modeling as a Baseline to Detect Climate Anxiety Among Youth - Documentation

Karim El-Sharkawy

March 26, 2025

## 1 Introduction & Environment Setup

One of the key initiatives of Climate Resilient Communities (CRC) is identifying climate anxiety among youth to provide timely mental health support and interventions. In this project, we analyze climate-related data from Reddit and X (formerly Twitter) using an LDA Topic Model. The results of this model will serve as our baseline, which will then be compared to an alternative model run using BERTopic. Note that the data has already been cleaned prior to this analysis, so this notebook focuses only on loading the preprocessed data.

To run the notebook, you'll need the following dependencies:

- `NumPy` version 1.26.4 for basic operations
- `Pandas` version 2.2.2 for dataframe manipulation
- `NLTK` version 3.9.1 for stopword removal and lemmatization
- `sklearn` version 1.6.1
- `gensim` version 4.3.3
- `matplotlib`, `seaborn`, and `pyLDAvis` version 3.4.0, which can be installed via: `pip install pyLDAvis`
- `scipy` version 3.15.0 for hierarchical clustering

For topic modeling, we use both `scikit-learn` and `gensim`. While each can be used independently for topic modeling, combining them leverages the strengths of both. `scikit-learn` lacks Jaccard and Jensen-Shannon distance metrics, which are useful for measuring topic similarities, but `gensim` supports these. On the other hand, `scikit-learn` integrates well with `GridSearchCV` and can convert BoW and TF-IDF into matrices using `CountVectorizer`. Additionally, `scikit-learn` works seamlessly with `pyLDAvis`, a feature that `gensim` struggles with.

## 2 Preprocessing Functions

This section introduces preprocessing functions designed to clean and prepare text data for topic modeling. We start by handling non-English characters with the `isEnglish(s)` and `fix_text(txt)` functions. `isEnglish(s)` checks if a string contains only ASCII characters, while `fix_text(txt)` processes the text, replacing or removing non-English characters based on specific rules. Next, we present the `get_jensen_shannon(components, ntopics)` function, which calculates the Jensen-Shannon divergence between topic distributions to assess topic diversity. We also introduce `get_jaccard(components, ntopics)`, which measures the Jaccard similarity between topics' top terms, though it's not in active use.

The section continues with the `LDAwithCustomScore` class, which extends `LatentDirichletAllocation` to use Jensen-Shannon divergence for scoring topic similarity. We also describe the lemmatization and stopword removal process using the WordNet Lemmatizer and a customized stopword list. Finally, we explain the `preprocess(text, stopwords)` function, which integrates all these preprocessing steps into a unified pipeline for preparing the text for topic modeling.

## 2.1 isEnglish(s) & fix_text(txt)

These functions handle non-English characters by checking if the text contains only ASCII characters and replacing or removing non-English characters. Here's how they work:

```
def isEnglish(s):
    try:
        s.encode(encoding='utf-8').decode('ascii')
    except UnicodeDecodeError:
        return False
    else:
        return True

def fix_text(txt):
    if not isEnglish(txt):
        for i, s in enumerate(txt):
            if not isEnglish(s):
                if len(txt)>=i+2:
                    if txt[i+1] == 's':
                        txt = txt.replace(s,"'")
                    elif txt[i+1] == ' ' and "'" not in txt:
                        txt = txt.replace(s,'-')
                    else:
                        txt = txt.replace(s,'')
                else:
                    txt = txt.replace(s,'')
    return txt
```

`isEnglish(s)` returns `True` if the string is composed only of ASCII characters, and `False` otherwise. If the string contains non-English characters, the `fix_text(txt)` function iterates through each character and applies specific rules to clean the text. Non-English characters are either replaced (e.g., with an apostrophe or hyphen) or removed entirely.

## 2.2 get_jensen_shannon(components, ntopics)

We then have the `get_jensen_shannon(components, ntopics)` function which calculates the Jensen-Shannon (JS) divergence between topics. Before describing the code, it's important to understand why we use the JS distance here. Taking from Darling's paper, *A Theoretical and Practical Implementation Tutorial on Topic Modeling and Gibbs Sampling*, topics under an LDA model are defined as such:

$$\text{For } k = 1 \dots K:$$
$$\phi^{(k)} \sim Dirichlet(\beta)$$

with $K$ latent topics, where $\phi^{(k)}$ is a discrete **probability distribution** representing the distribution of the $k$-th topic. This suggests that each topic represents a probability distribution, and comparing topics involves comparing these distributions. This is precisely what `gensim`'s Jensen-Shannon measure does: it quantifies the divergence between probability distributions. In contrast, `sklearn.metrics` only provides Kullback-Leibler Divergence, which is asymmetric and more difficult to interpret. Hence, it is less suitable for topic models, especially when context is crucial. Finally, on to the function:

```
def get_jensen_shannon(components, ntopics):
    topic_dists = components
    js_dists = []
    for i in range(ntopics):
        for j in range(ntopics):
            if i>j:
                js_dists.append(jensen_shannon(topic_dists[i,:], topic_dists[j,:]))

    return np.min(js_dists), np.mean(js_dists)
```

The two inputs are:

1. `components`: a matrix where each row represents the distribution of words for a specific topic.
2. `ntopics`: the number of topics in the model.

The function compares each pair of topics by calculating the JS divergence between their word distributions. Specifically, for each pair of topics $(i, j)$ where $i \neq j$, the divergence is computed using the distributions of the $i^{th}$ and $j^{th}$ topics. It then returns the minimum and mean divergence values to assess the diversity of topics.

## 2.3 `get_jaccard(components, ntopics)`

The `get_jaccard(components, ntopics)` takes the same inputs, but calculates the Jaccard similarity between the top 10% terms of different topics (although it's not currently being used).

```
def get_jaccard(components, ntopics): # not being used
    topn = int(np.ceil(len(dictionary)*(10/100)))
    topic_word_probs = components
    top_terms = np.argsort(-1*topic_word_probs,axis=1)
    top_terms = 1*top_terms[:,0:topn]
    jdists = []
    for i in range(ntopics):
        for j in range(ntopics):
            if i > j:
                jdists.append(jaccard(top_terms[i,:], top_terms[j,:]))
    return np.min(jdists), np.mean(jdists)
```

The Jaccard similarity is given but unused. It's ideal when we're uncertain about the relevance of all words in a topic, such as with noise or irrelevant terms, and want to focus on shared term presence rather than distribution. However, for a more nuanced comparison that accounts for term importance within each topic, JS divergence is the better option.

## 2.4 `LDAwithCustomScore` Class

This class extends `LatentDirichletAllocation` from `sklearn.decomposition` to customize the scoring mechanism based on the Jensen-Shannon divergence:

```
class LDAwithCustomScore(LatentDirichletAllocation):
    def score(self, X, y=None):
        components = self.components_
        ntopics = self.n_components
        score = get_jensen_shannon(components, ntopics)[0]
        return score
```

The `score(self, X, y=None)` function overrides the default LDA scoring by calculating a score based on the similarity between topics and takes in two parameters:

- `X`: Input data (document-term matrix or feature matrix).
- `y`: Ignored, included for consistency with scikit-learn's API.

## 2.5 `lemmatize(text)` & Stopword Removal

```
stopwords = list(gensim.parsing.preprocessing.STOPWORDS)
stopwords.append([
...
])

def lemmatize(text):
    lemmatizer = WordNetLemmatizer()
    return lemmatizer.lemmatize(text, pos='v')
```

We use the WordNet Lemmatizer from the `nltk` library to perform lemmatization on the input `text`, treating each word as a verb (`pos='v'`). Additionally, we define a list of stopwords using `gensim`'s predefined stopwords set. Moreover, we remove additional words such as

1. redundant, vague, or filler words (e.g. 'basically', 'it is', 'idk').
2. platform-specific words (e.g. 'hashtag', 'following', 'upvote').
3. expected climate terms ('climate', 'climate change', 'global warming') that might dominate topics.
4. Profanities, which carry emotional meaning but might introduce noise.

Note that removing emotional stopwords may be harmful if sentiment analysis is not conducted on the dataset beforehand (which was already done). We remove the expected climate terms because we already know they're going to show up in our topic model when the data is climate specific, so the word 'climate' for example is to be expected.

### 2.6 `preprocess(text, stopwords)`

```
def preprocess(text, stopwords):
    result = []
    stem_dict = []

    text = fix_text(text)  # Fix text first
    tokens = gensim.utils.simple_preprocess(text)

    for token in tokens:
        if token not in stopwords:
            lemmatized_token = lemmatize(token)
            if lemmatized_token not in stopwords and len(lemmatized_token) > 3:
                result.append(lemmatized_token)
                stem_dict.append((lemmatized_token, token))

    return result, stem_dict
```

The process begins with `fix_text(text)` defined before, and returns a cleaned version of the input text. Then, `gensim.utils.simple_process(text)` converts the input string into lowercase, removes punctuation, and returns a list of tokens. During the processing loop, each token is checked against the stopwords list. If the token is not a stopword, it is lemmatized using the `lemmatize()` function. After lemmatization, the word is checked again to ensure it is not a stopword and that its length is greater than three characters. Valid tokens are appended to the `result` list, while a tuple of the lemmatized token and its original form is added to the `stem_dict` list.

## 3 LDA Topic Modeling

This section outlines the process of selecting, evaluating, and visualizing the best model for both datasets. We begin with the `best_model` function, which uses a grid search approach to find the optimal number of topics based on cross-validation performance. The resulting model is evaluated using two important metrics: coherence and perplexity, which assess the interpretability and generalization ability of the model, respectively. Additionally, we explore clustering techniques with the `Cluster` function, which groups documents based on their topic distributions using KMeans. To further aid in model interpretation, the `visualize_topics` function provides an interactive visualization of the topic-term relationships, and the `hierarchy` function performs hierarchical clustering to reveal document clusters based on their topic distributions. Together, these methods provide a comprehensive approach for building, evaluating, and interpreting LDA models in text analysis.

### 3.1 Finding the best model: `best_model`

```
def best_model(documents, ntopics_list):
    processed_info = []
    for allinfo in documents['text'].values:
        preprocessed, stemdict = preprocess(allinfo, stopwords)
        processed_info.append(preprocessed)

    countvec = CountVectorizer(ngram_range=(1,1), stop_words=stopwords, max_df=.25, min_df=10)
    clean_text = [' '.join(text) for text in processed_info]
    X = countvec.fit_transform(clean_text).toarray()
    wft = np.sum(X, axis=0).T

    allterms = countvec.get_feature_names_out()

    # Using grid search CV with pipeline to find the best model
    search_params = {'n_components': ntopics_list}
    lda = LDAwithCustomScore(random_state=0)
    model = GridSearchCV(lda, param_grid=search_params, cv=5)
    model.fit(X)

    return model.best_estimator_, processed_info, X, allterms, countvec # return the best model
```

The purpose of the text preprocessing function is to clean and prepare Reddit posts and comments for modeling as described earlier. We end with `processed_info` updated. Using `sklearn`'s `CountVectorizer`, the list of preprocessed tokens are converted into a document-term matrix (DTM) where each row represents a document (post/tweet) and each column corresponds to a term in the entire corpus.

- `ngram_range=(1,1)` ensures that only unigrams (single words) are considered. Using bigrams might introduce more noise and will be harder to analyze if they are not meaningful in the specific context of the data.
- `max_df=.25` excludes words that appear in more than 25% of the documents. By excluding such frequent terms, we focus on words that are more specific to the individual topics.
- `min_df=10` excludes words that appear in fewer than 10 documents. Words that are too rare may not contribute meaningfully to topic discovery and might increase the model's susceptibility to overfitting on specific instances of the dataset. By using `min_df=10`, we ensure that only words that appear in a sufficient number of documents (and thus, are likely to be relevant to more than one topic) are included in the DTM.

The outputs of the function include two key components. First, `X` is a document-term matrix represented as a NumPy array, where each row corresponds to a Reddit post and each column represents the frequency of a specific term within the post. Second, `wft` provides the sum of word frequencies across all documents, giving the overall frequency of each word in the entire dataset.

The `GridSearchCV` is utilized to determine the optimal number of topics (`n_components`) for the Latent Dirichlet Allocation (LDA) model by testing various values from the `ntopics_list` and evaluating the model's performance using 5-fold cross-validation.

When selecting the best number of topics (`n_components`), cross-validation is a more reliable method than testing multiple values without validation because it leads to a model that generalizes well to all data, not just a specific dataset. Cross-validation helps avoid this by evaluating the model on different subsets of the data. In k-fold cross-validation (5 in this case), the model is trained on k-1 folds and validated on the remaining fold. This helps:

1. Evaluate the model on unseen data for a more realistic performance estimate.
2. Prevent overfitting to any specific subset.
3. Ensure the number of topics generalizes well across different subsets.

### 3.2 Model Evaluation: `Evaluate()`

```
def Evaluate(model, processed_info, X, allterms):
  topic_word_distributions = model.components_
  top_n_words = 10
  topics = [[allterms[i] for i in topic.argsort()[:-top_n_words - 1:-1]] for topic in topic_word_distributi

  dictionary = gensim.corpora.Dictionary(processed_info)
  coherence_model = CoherenceModel(topics=topics, texts=processed_info,
                                   dictionary=dictionary, coherence='c_v')
  coherence_score = coherence_model.get_coherence()

  print(f'Coherence Score: {coherence_score}')

  perplexity = model.perplexity(X)
  print(f'Perplexity: {perplexity}')

  topic_distribution = model.transform(X)

  return coherence_score, perplexity, topic_distribution
```

The `topic_word_distributions` contain the topic-word distribution matrix, where each row represents a topic and each column corresponds to a word's weight in that topic. The code extracts the top 10 words for each topic using `argsort()` to find the indices of the words in each topic, sorted by their weights in descending order.

There are multiple types of topic model evaluations. We are using:

1. The coherence score measures the interpretability and quality of the topics, based on how often words that appear together in documents are also ranked highly in the topic-word distributions. The closer to 1, the better the interpretability (on a 0 to 1 scale).

2. Perplexity is a measure of how well the model predicts unseen data. Lower perplexity values indicate better generalization to new, unseen data (on a scale of 0 to $\infty$).

Using both metrics is important because they evaluate different aspects of the model. Combining both gives us a more comprehensive picture of model performance, ensuring that we are not only extracting meaningful topics but also ensuring that the model can generalize well.

Before we discuss them separately, it's important to mention that the topic distribution (`topic_distribution = model.transform(X)`) is calculated and returned in this function. This will be useful for the visualization and clustering portions.

### 3.2.1 Coherence Score

```
dictionary = gensim.corpora.Dictionary(processed_info)
coherence_model = CoherenceModel(topics=topics, texts=processed_info,
    dictionary=dictionary, coherence='c_v')
coherence_score = coherence_model.get_coherence()

print(f'Coherence Score: {coherence_score}')
```

- `gensim.corpora.Dictionary(processed_info)` creates a dictionary of unique words from the preprocessed data.

- `CoherenceModel` from `gensim` calculates the coherence score using the `c_v` metric.

The function takes three inputs: `topics`, which is a list of the top 10 words for each topic; `processed_info`, the preprocessed Reddit text data; and `dictionary`, the Gensim dictionary. The output is `coherence_score`, a numerical value that represents the coherence of the topics, where higher values indicate better coherence.

### 3.2.2 Perplexity

```
perplexity = best_lda_model.perplexity(X)
print(f'Perplexity: {perplexity}')
```

The function takes `X` as input, which is the document-term matrix used to train the LDA model. The output is `perplexity`, a numerical value representing the perplexity of the model, where lower values indicate better performance.

## 3.3 Topic-Term Distributions: `visualize_topics()`

```
def visualize_topics(model, topic_distribution, X, countvec):
    topic_term_dists = model.components_ / model.components_.sum(axis=1)[:, np.newaxis]
    doc_lengths = X.sum(axis=1).tolist()
    term_frequency = np.asarray(X.sum(axis=0)).flatten().tolist()
    vocab = countvec.get_feature_names_out()

    data = pyLDAvis.prepare(
        topic_term_dists,  # topic-word distributions
        topic_distribution,   # document-topic distributions
        doc_lengths,       # total word count per document
        vocab,             # list of terms/features
        term_frequency     # sum of word counts for each term
    )

    return data
```

This function calculates key distributions related to topics and terms in the LDA model. It computes:

- `doc_topic_dists`: A matrix where each row represents a document, and each column shows the proportion of each topic in that document.
- `topic_term_dists`: A normalized topic-term distribution, ensuring each topic's word distribution sums to 1.
- `doc_lengths`: The total word count for each document.
- `term_frequency`: The frequency of each term across all documents.
- `vocab`: The list of terms (vocabulary) used in the document-term matrix.

The function outputs these five components and visualizes the topics using PyLDAvis. First, it prepares the data with `pyLDAvis.prepare` by inputting the necessary distributions. Then, `pyLDAvis.enable_notebook()` enables visualization in a Jupyter notebook, and `pyLDAvis.display(data)` shows the interactive topic model 2D visualization with topic-term relationships and frequent terms for each topic.

## 3.4 Agglomerative Clustering: `hierarchy()`

```
sys.setrecursionlimit(10000)
names = ['Reddit', 'Twitter']
def hierarchy(model, topic_distribution, dataset_name, n_clusters=None, distance_threshold=0, linkage_metho
    agg_clustering = AgglomerativeClustering(n_clusters=n_clusters, distance_threshold=distance_threshold)
    agg_clustering.fit(topic_distribution)

    children = agg_clustering.children_
    n_samples = len(topic_distribution)
    linkage_matrix = np.zeros((n_samples - 1, 4)) # empty linkage matrix

    for i, (left, right) in enumerate(children):
        linkage_matrix[i, 0] = left
        linkage_matrix[i, 1] = right
        linkage_matrix[i, 2] = agg_clustering.distances_[i]
        linkage_matrix[i, 3] = 2

    plt.figure(figsize=figsize)
    dendrogram(linkage_matrix, labels=[f"Doc {i+1}" for i in range(n_samples)])  # creating a dendrogram
    plt.title(f'{dataset_name} Hierarchical Clustering')
    plt.xlabel('Documents')
    plt.ylabel('Distance')
    plt.show()
    return agg_clustering
```

This code performs hierarchical clustering using agglomerative clustering on the topic distributions. Agglomerative clustering is applied to the topic distributions to create a hierarchy of clusters. The parameters `n_clusters=None, distance_threshold=0` ensure that no predefined number of clusters is specified, and the algorithm will produce a full dendrogram. The `agg_clustering.fit(topic_distribution)` method performs the clustering and assigns hierarchical cluster relationships based on the topic distributions.

The output is `agg_clustering.labels_`, which contains the labels assigned to each document, representing the cluster that each document belongs to in the hierarchical clustering scheme.

It then visualizes the hierarchical clustering of the topic distributions as a dendrogram. The `linkage` function from `scipy` computes the hierarchical clustering of the topic distributions using the Ward method. The `dendrogram(linked)` function then visualizes the hierarchical clustering as a tree-like structure, where the documents are grouped based on their topic distribution similarity. Finally, the dendrogram plot is displayed and shows how documents are clustered hierarchically based on their topic distributions.

### 3.5 Putting it all Together

Once the LDA model has been evaluated and the optimal number of topics has been selected, several functions work in concert to refine the model and provide deeper insights into the data. The Cross-Validation Grid Search is crucial in this process, guiding the selection of the ideal number of topics by ensuring that they are both meaningful and interpretable, thus preventing overfitting. Additionally, the coherence and perplexity scores assess the model's performance to interpret existing data and generalize to unseen data. These metrics help in fine-tuning the model for a more accurate representation of the underlying text.

Moreover, visual and clustering techniques enhance the interpretability of the model's results. The topic-term visualizations, generated by the `visualize_topics()` function, offer an interactive and intuitive way to explore the relationships between topics and their most frequent terms, making it easier to understand the structure of the topics. The hierarchical clustering function further contributes by grouping similar documents based on their topic distributions, providing an additional layer of insight into the data. This combination of statistical metrics and visual tools not only refines the model but also ensures that the extracted topics are meaningful, generalizable, and easily interpretable, offering a comprehensive understanding of the text corpus.

## 4 Visualization

This code visualizes various performance metrics and topic distributions of the two models (`best_reddit_model` and `best_twitter_model`). It generates multiple plots to compare the models in terms of training time, coherence score, perplexity score, word cloud representations, and topic-word distributions.

### 4.1 Comparing Coherence & Perplexity Scores

```
coherence = [reddit_coherence_score, twitter_coherence_score]
perplexity = [reddit_perplexity, twitter_perplexity]
names = ['Reddit', 'Twitter']

colors = ['r', 'b']

for i in range(len(names)):
    plt.scatter(coherence[i], perplexity[i], color=colors[i], s=100)

    plt.text(coherence[i], perplexity[i], f'{names[i]}',
             fontsize=10, ha='right', va='bottom') # names next to points

plt.xlim(0, 1)
plt.xlabel('Coherence')
plt.ylabel('Perplexity')
plt.title('Coherence & Perplexity of Reddit and Twitter LDA Models')
```

```
### Code is cut here due to length, check notebook for full code ###
plt.scatter(... label=f'Reddit: Coherence={coherence[0]:.2f}, Perplexity={int(perplexity[0])}')
plt.scatter([], [], color='b', label=f'Twitter: Coherence={coherence[1]:.2f}...')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.1), ncol=1)

plt.grid(True)
plt.show()
```

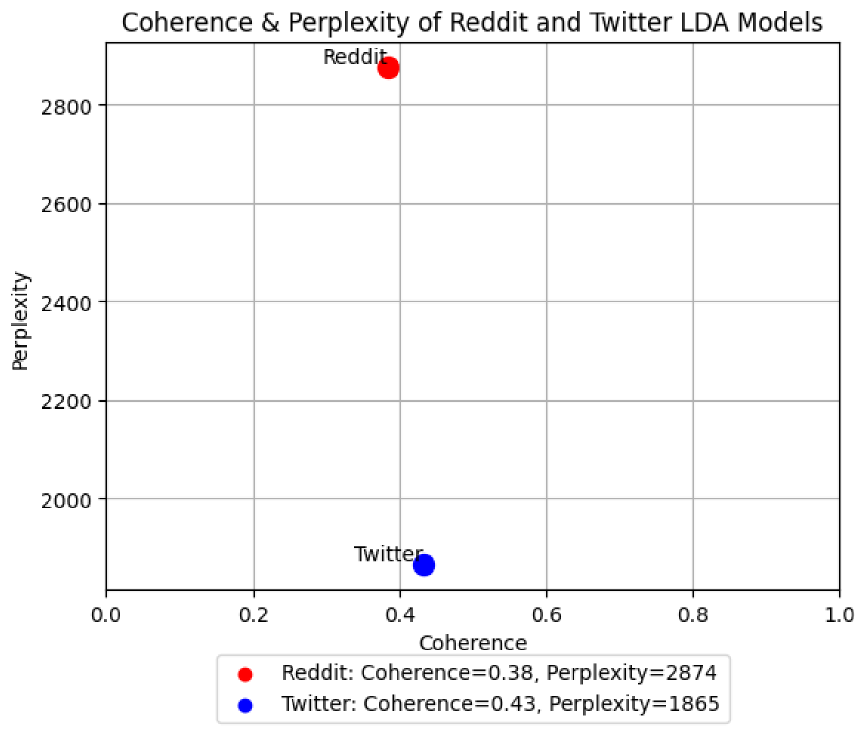


Figure 1: Reddit VS Twitter Comparison

## 4.2 Word Clouds for Top Words in Topics

Word clouds are a useful way to visualize the most significant words in each topic, allowing a quick overview of the key themes that each topic captures.

```
from wordcloud import WordCloud

def generate_word_cloud(model, terms, top_n=10):
    top_words = [] # select the top top_n words for the word cloud
    for topic in model.components_:
        top_words += [terms[i] for i in topic.argsort()[:-top_n - 1:-1]]

    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(top_words))
    return wordcloud

reddit_wordcloud = generate_word_cloud(best_reddit_model, reddit_terms, top_n=10)
plt.figure(figsize=(10, 6))
plt.imshow(reddit_wordcloud, interpolation='bilinear')
plt.title('Reddit LDA Model Word Cloud')
plt.axis('off')
plt.show()
```

```
twitter_wordcloud = generate_word_cloud(best_twitter_model, twitter_terms, top_n=10)
plt.figure(figsize=(10, 6))
plt.imshow(twitter_wordcloud, interpolation='bilinear')
plt.title('Twitter LDA Model Word Cloud')
plt.axis('off')
plt.show()
```

The code includes a function `generate_word_cloud`, which generates word clouds (shown below) for the top words in the topics learned by the LDA models for Reddit and Twitter. This function accepts a model, a list of terms, and the number of top words to display. It first identifies the top words for each topic in the model and then generates a word cloud. The function is used to create and display word clouds for both Reddit and Twitter models using `WordCloud` from the `wordcloud` library.

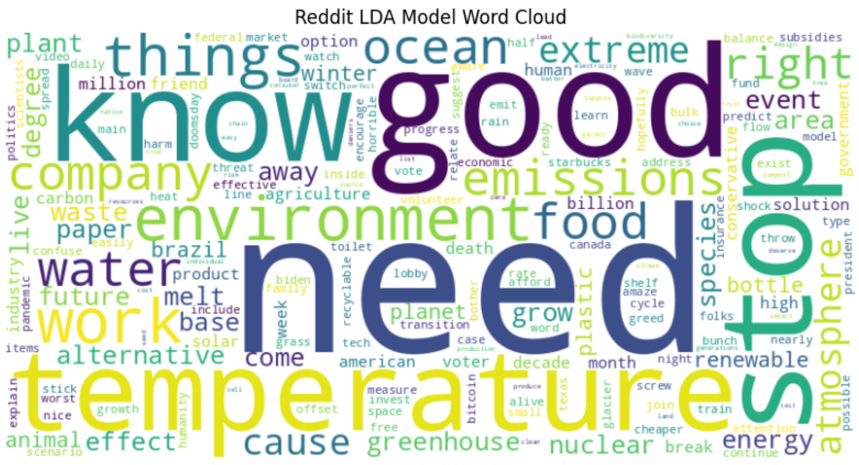


Figure 2: Reddit Word Cloud

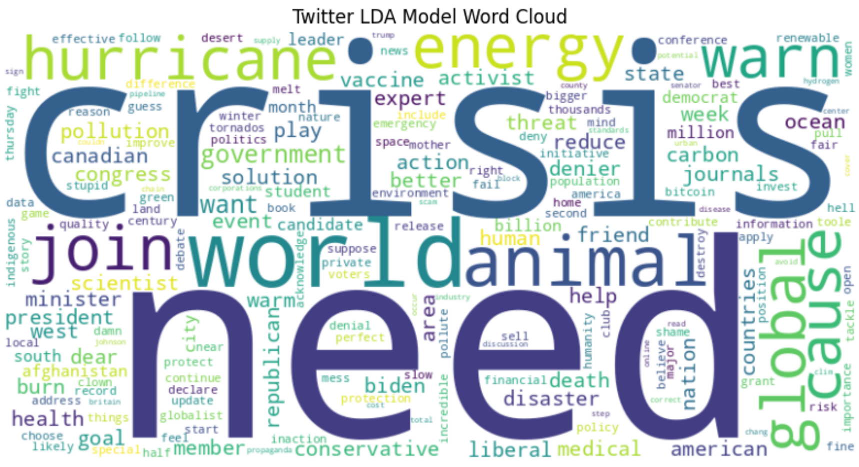


Figure 3: Twitter Word Cloud

## 4.3 Topic Word Distribution Comparison

We also compare the distribution of words across topics for both models. By visualizing the top words associated with each topic, we can better understand the composition of each topic and how the model has grouped related terms.

```
def plot_topic_word_distributions(model, terms, dataset_name, top_n=1):
    topic_word_distributions = model.components_ / model.components_.sum(axis=1)[:, np.newaxis]
    topics = np.argsort(-topic_word_distributions, axis=1)[:, :top_n]

    data = np.zeros((len(topics), top_n))
    for i, topic in enumerate(topics):
        data[i] = topic_word_distributions[i, topic]

    fig, ax = plt.subplots(figsize=(15, 30))
    ax.barh(range(len(topics)), data.sum(axis=1), color='gray', alpha=0.5)

    colors = plt.cm.get_cmap('tab20', len(topics))
    for i, topic in enumerate(topics):
        ax.barh(i, data[i], label=[terms[j] for j in topic], color=colors(i))

    ax.set_title(f'{dataset_name} Topic Word Distribution')
    ax.set_yticks(range(len(topics)))
    ax.set_yticklabels([f'Topic {i+1}' for i in range(len(topics))])
    ax.legend(title='Top Words', bbox_to_anchor=(1.05, 1), loc='upper left')

    plt.xlabel('Word Distribution')
    plt.tight_layout()
    plt.show()

plot_topic_word_distributions(best_reddit_model, reddit_terms, dataset_name=names[0], top_n=1)
plot_topic_word_distributions(best_twitter_model, twitter_terms, dataset_name=names[1], top_n=1)
```

The `plot_topic_word_distributions` function compares the distribution of words across topics in the Reddit and Twitter LDA models. It normalizes the word distribution for each topic, then plots a stacked bar chart to visualize the distribution of the top words for each topic. The bar chart displays topics on the y-axis and the word distribution on the x-axis.

By examining the chart, we can observe how the LDA model assigns different words to each topic. A higher bar for a word indicates that it is a dominant feature of the corresponding topic. This visualization allows for a clearer understanding of the topic's characteristics. For example, in some topics, a small number of words may dominate, while in others, the distribution may be more evenly spread across many words.

The color-coding of the bars in the plot helps distinguish between different topics. Each topic is assigned a unique color, making it easy to identify the contributions of different words to each topic. This visualization allows us to compare the distributions of word usage across topics in the Reddit and Twitter models, providing insight into how each model has learned and represented the topics.

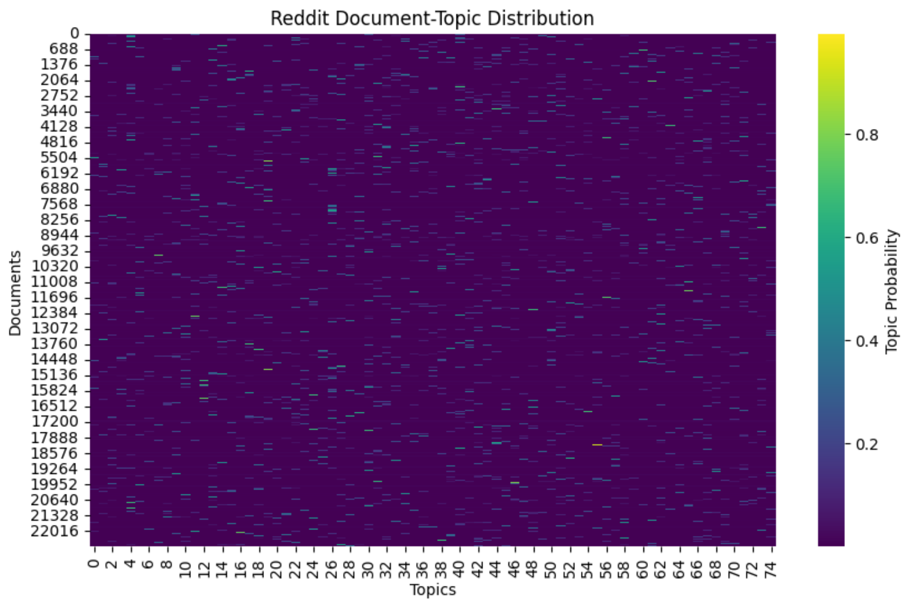
## 4.4 Heatmap of Document-Topic Distributions

Heatmaps allow us to see which topics are more prominent in particular documents, giving a sense of how specific topics relate to different portions of the text corpus.

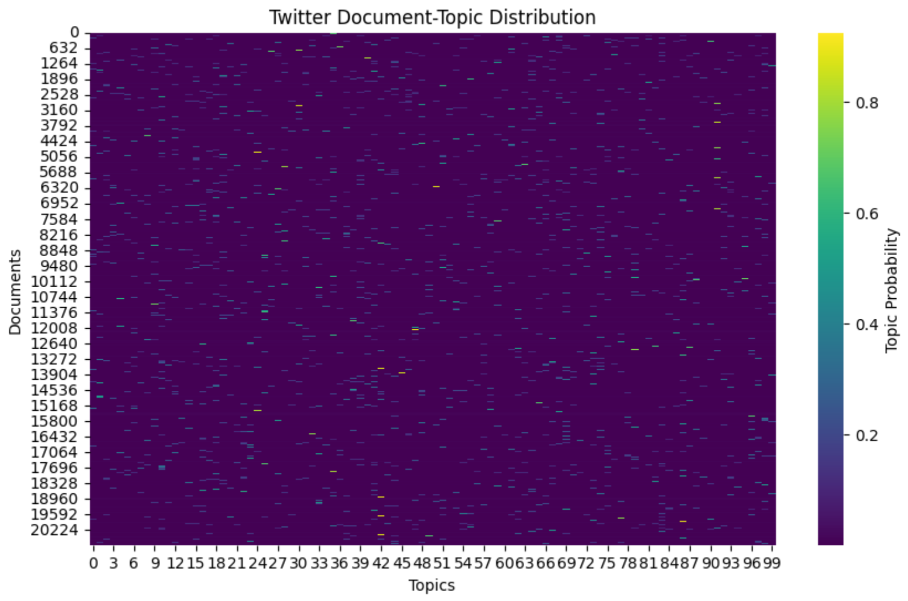
```
plt.figure(figsize=(10, 6))
sns.heatmap(reddit_topic_distribution, cmap='viridis', cbar_kws={'label': 'Topic Probability'})
plt.title('Reddit Document-Topic Distribution')
plt.xlabel('Topics')
plt.ylabel('Documents')
plt.show()

plt.figure(figsize=(10, 6))
sns.heatmap(twitter_topic_distribution, cmap='viridis', cbar_kws={'label': 'Topic Probability'})
plt.title('Twitter Document-Topic Distribution')
plt.xlabel('Topics')
plt.ylabel('Documents')
plt.show()
```

We used the `seaborn` library to visualize the document-topic distributions for both Reddit and Twitter models using heatmaps shown below. They show the probability of each topic for each document, with the topics on the x-axis and documents on the y-axis. A darker color indicates a lower probability for a topic in a document, helping us to identify non-dominant themes within individual posts or tweets.



(a) Reddit Heatmap



(b) Twitter Heatmap

Figure 4: Reddit & Twitter Heatmaps

## 5 Results & Discussion

Overall, the results for the two LDA models were as follows: For Reddit, the optimal number of distinct topics was 75, after optimization using GridSearchCV. The model achieved a coherence score of 0.3839 and a perplexity value of 2874.72. In contrast, the Twitter model identified 100 distinct topics, with a coherence score of 0.4317 and a perplexity of 1865.36.

## 5.1 Evaluation & Comparison

### 5.1.1 Coherence and Perplexity Scores Analysis

Coherence scores measure the semantic similarity between words within each topic. A higher coherence score (which goes from 0 to 1) suggests that the topics are more meaningful and interpretable.

- The Reddit model's coherence score of **0.38** indicates moderate interpretability, but there may be some overlap between topics or less distinct themes.
- The Twitter model's coherence score of **0.43** suggests that the topics are more internally consistent, indicating that the model successfully captures clear distinctions between topics, making them easier to interpret.

Perplexity measures how well the model predicts unseen words in the corpus. Lower perplexity generally indicates a better generalization of topics.

- The Reddit model's perplexity score of **2874.72** suggests that the model struggles with generalization and is most likely overfitting. The relatively small dataset size may contribute to this overfitting, as there might be insufficient data to define strong topic boundaries.
- The Twitter model's perplexity score of **1865.36** is significantly lower, indicating a better fit to the data. This means the model captures patterns in the corpus more effectively and overfits less (but not completely).

Overall, the Reddit model identifies 75 topics, which suggests a more consolidated set of topics compared to Twitter's 100 topics. With fewer topics, the Reddit model may generalize the content more, potentially sacrificing granularity and distinctness.

In contrast, the Twitter model, with 100 topics, offers a finer granularity, potentially capturing more subtle distinctions in the data.

### 5.1.2 Why the Twitter Model Performs Better

Several factors could contribute to Twitter's more interpretable and generalized model:

1. Twitter character limit makes posts shorter and forces focus and direct language, meaning that tweets within the same topic are more linguistically similar. Hence, there's a clear topic separation. The direct language naturally clusters together, helping the model form clear topics.
2. Reddit discussions (including comments sometimes) tend to be longer and cover multiple angles of a discussion. Hence, topics seem to blend together and increase ambiguity

Hence, the Twitter model performs better due to shorter, more focused content and clearer linguistic patterns, leading to a higher number of topics and better coherence and perplexity scores. The Reddit model suffers from topic overlap due to longer posts and broader discussions, making it harder to generate distinct topics. Reddit's discourse is more complex and multifaceted, while Twitter's content is more streamlined and topic-specific.

## 5.2 Hierarchies

Because of the large number of topics in each model, the clustering is very complex and messy. For the reddit hierarchy, the top levels of the show distinct groupings, meaning the model has successfully identified broad topic categories (aligning with the previous subsection results). The Twitter model has meaningful relationships between different topic distributions. The large blue and orange branches at the top indicate some major topic divisions, which might correspond to distinct overarching themes in the dataset. However, the lower-level green branches are densely packed, making it hard to visually interpret individual clusters.

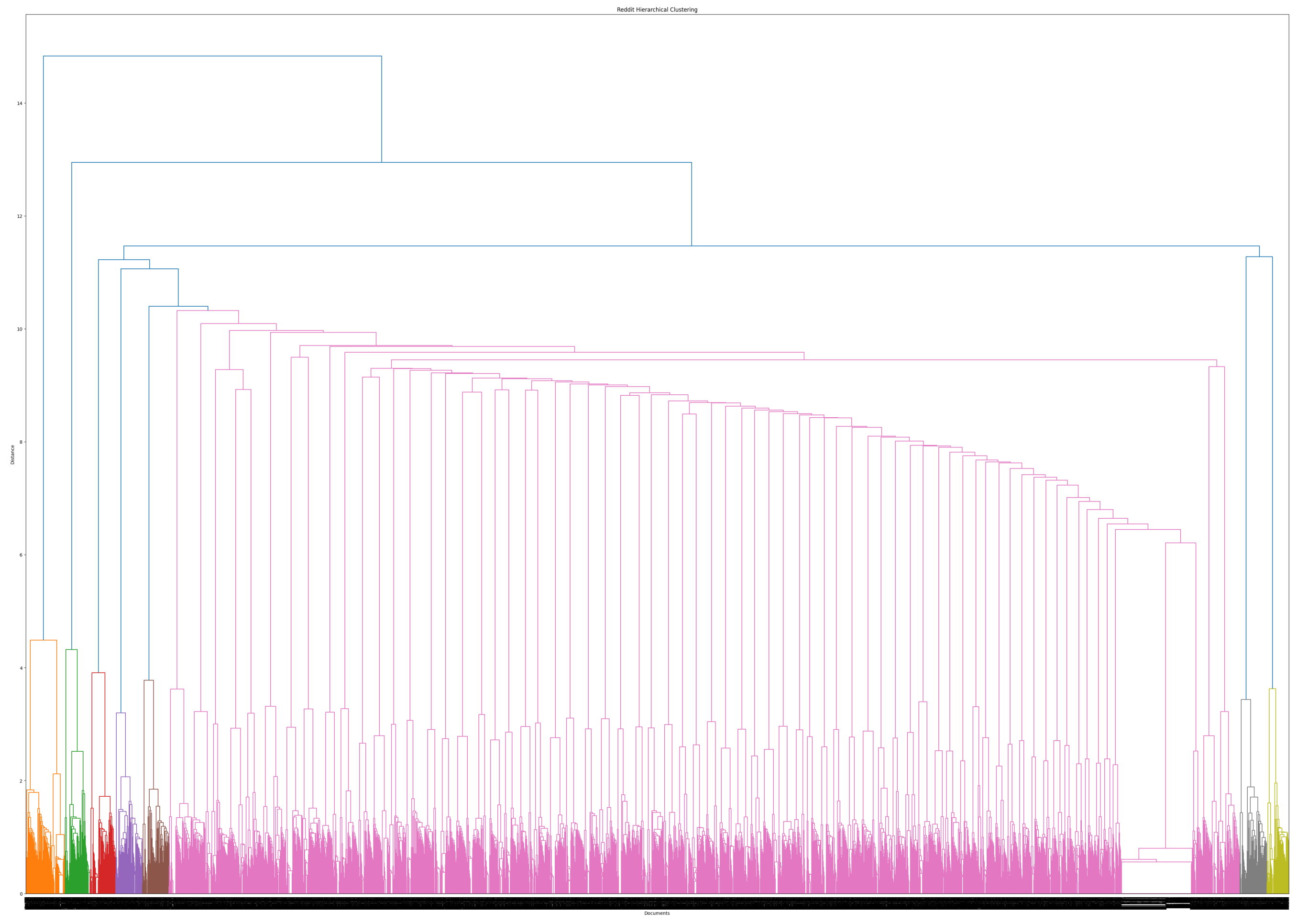


Figure 5: Reddit Hierarchy

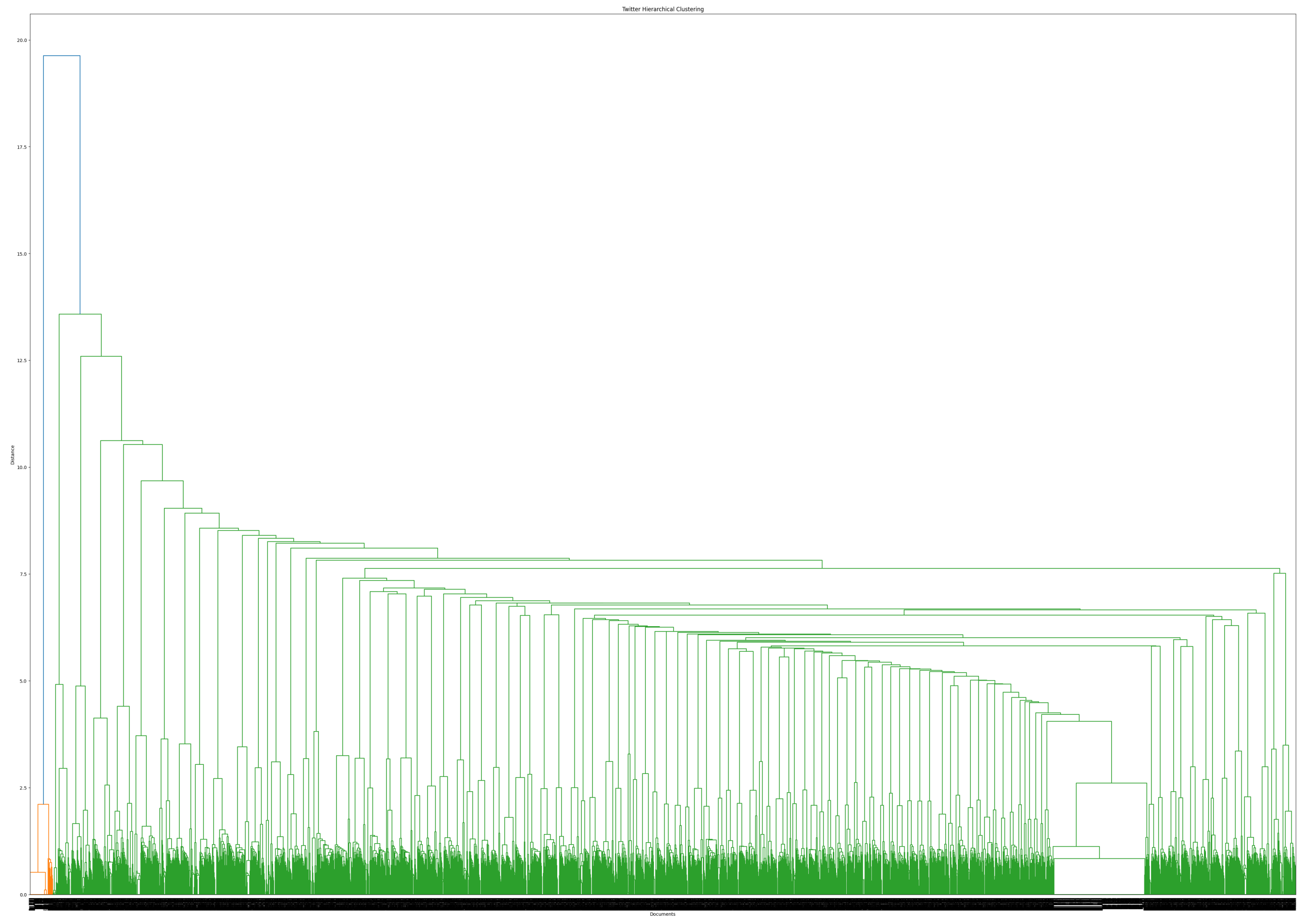


Figure 6: Twitter Hierarchy

## 5.3 Topic-Word Distributions

We now examine climate-related discussions on Reddit and X (Twitter) based on the topic word distributions shown below. By analyzing the key terms and themes present in both platforms, we can identify the specific aspects of climate discourse that each prioritizes.

### 5.3.1 Reddit's Focus

Reddit's discussions tend to be analytical, scientific, and policy-driven, exhibiting the following key trends:

- Climate Science and Long-Term Impact: Discussions often include references to climate models, temperature trends, and global warming projections.
- Policy and Environmental Action: A strong emphasis on governmental policies, sustainability efforts, and legislative debates about climate regulations.
- Renewable Energy and Technological Solutions: Frequent discourse around advancements in solar, wind, and alternative energy sources.
- Climate Activism and Social Movements: Topics include grassroots environmental movements, activism strategies, and climate advocacy.
- Scientific Skepticism and Debate: While largely pro-science, some discussions engage with climate skepticism and misinformation, often in a debunking context.

### 5.3.2 X (Twitter)'s Focus

X tends to be more event-driven, focusing on immediate climate-related occurrences and their social impact:

- Extreme Weather Events: Heavy discussion around hurricanes, floods, wildfires, and other disasters, often linked to climate change.
- Viral Climate Awareness: Climate-related content spreads through trending hashtags, calls to action, and viral posts.
- Political and Public Figures: Conversations frequently center around political leaders, corporations, and public figures' stances on climate issues.
- Environmental Justice and Social Impact: Greater emphasis on how climate change disproportionately affects marginalized communities.
- Short-Term Reactions and Public Discourse: Discussions tend to focus on immediate impacts rather than long-term scientific projections.

### 5.3.3 Main Differences

- Reddit's climate discussions are more research-oriented, while X's focus is on real-time events and reactions.
- Reddit discusses legislative solutions, while Twitter highlights political leaders and corporate responsibility.
- Reddit debates climate change projections and mitigation strategies, whereas Twitter reacts to ongoing disasters and their human impact.
- Reddit discussions engage with data-driven climate science, while Twitter emphasizes climate justice, activism, and media coverage.

Reddit and X (Twitter) provide complementary perspectives on climate discourse. Reddit serves as a hub for in-depth discussions on policy, science, and technology, whereas Twitter amplifies awareness through real-time updates, social activism, and public discourse.

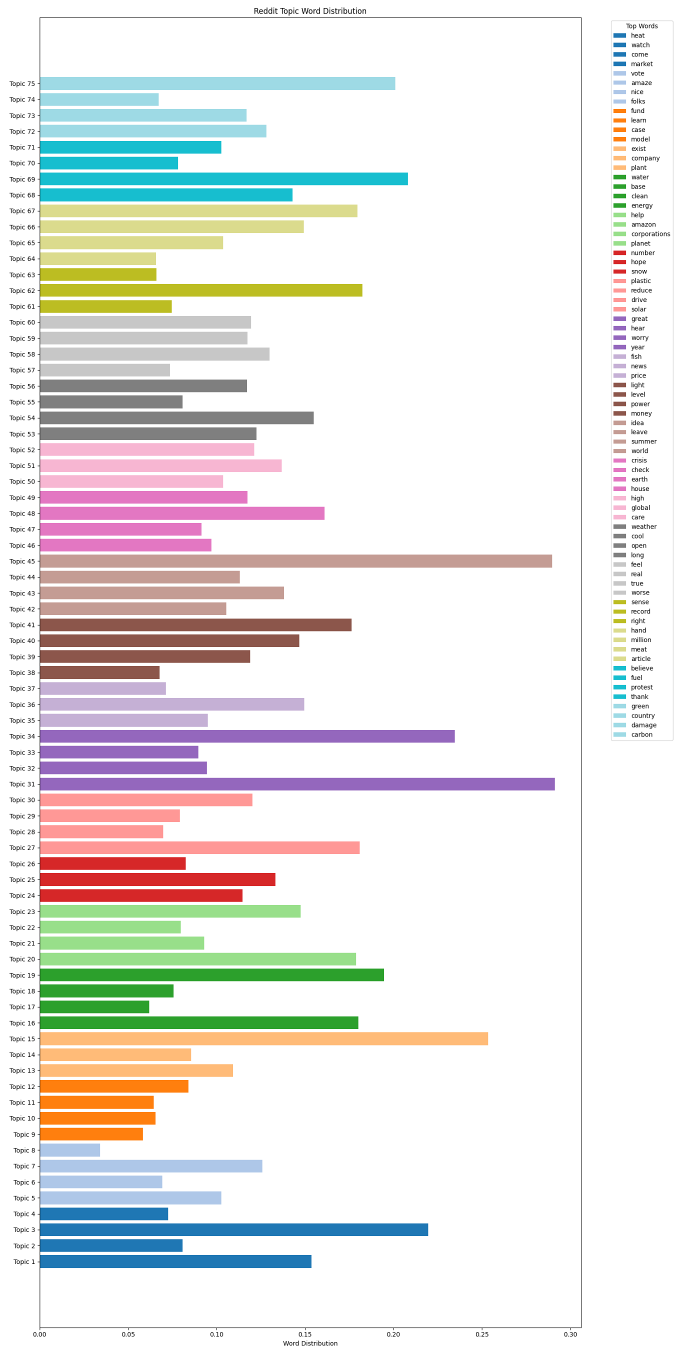


Figure 7: Reddit Topic-Word Distribution

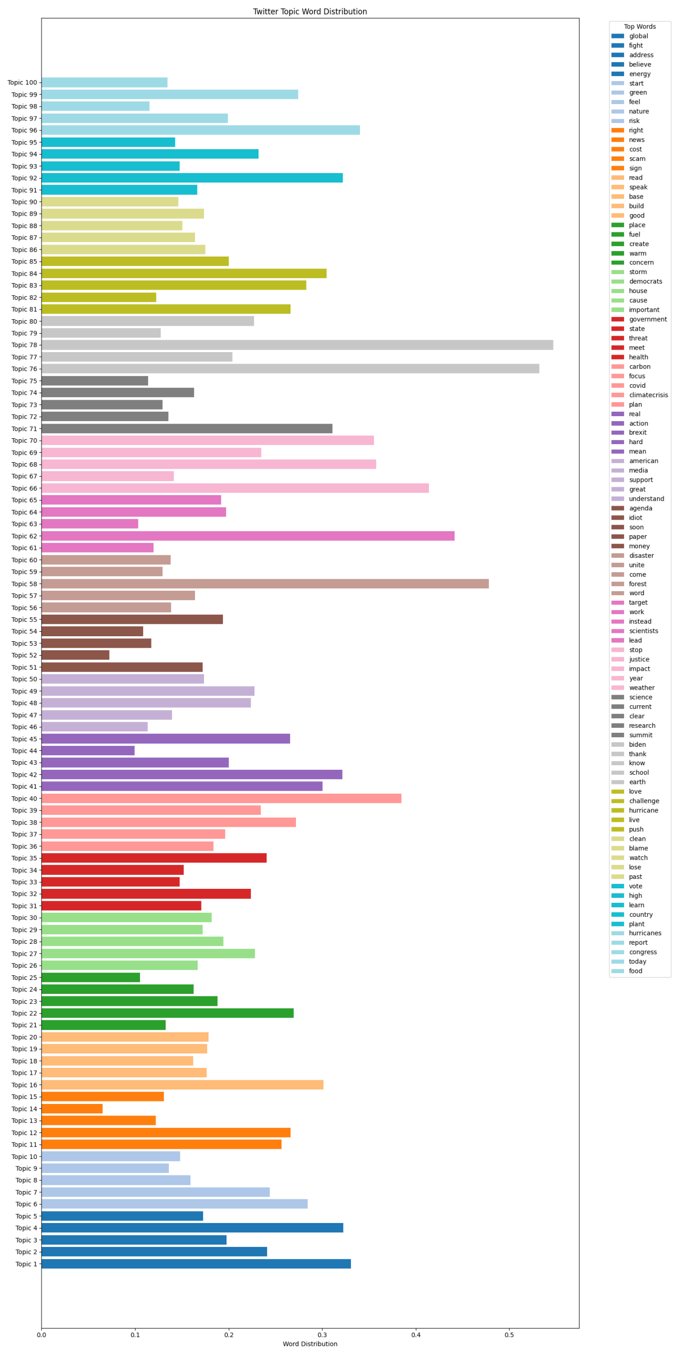


Figure 8: Twitter Topic-Word Distribution

## 6 Conclusion

This study presents an application of Latent Dirichlet Allocation (LDA) to detect and analyze climate anxiety among youth using data collected from Reddit and X. By leveraging both `scikit-learn` and `gensim`, we capitalize on their respective strengths—efficient matrix conversion, hyperparameter tuning, and advanced similarity metrics like Jensen-Shannon divergence.

Our preprocessing pipeline ensures high-quality input data by incorporating language filtering, lemmatization, and stopword removal, improving the robustness of topic modeling. Through LDA, we identify key thematic clusters in climate-related discussions, providing a structured view of the discourse surrounding climate anxiety. The use of Jensen-Shannon divergence and Jaccard similarity allows for quantitative evaluation of topic cohesion and separation.

The results of this baseline study set the stage for further refinement and comparison using BERTopic, which may offer improved coherence and interpretability. Future work includes fine-tuning hyperparameters, expanding the dataset to ensure broader representational diversity, and exploring alternative topic modeling techniques for more nuanced insights. Ultimately, this research contributes to a better understanding of climate anxiety in online discussions, guiding mental health interventions and policy recommendations.