Topic Modeling

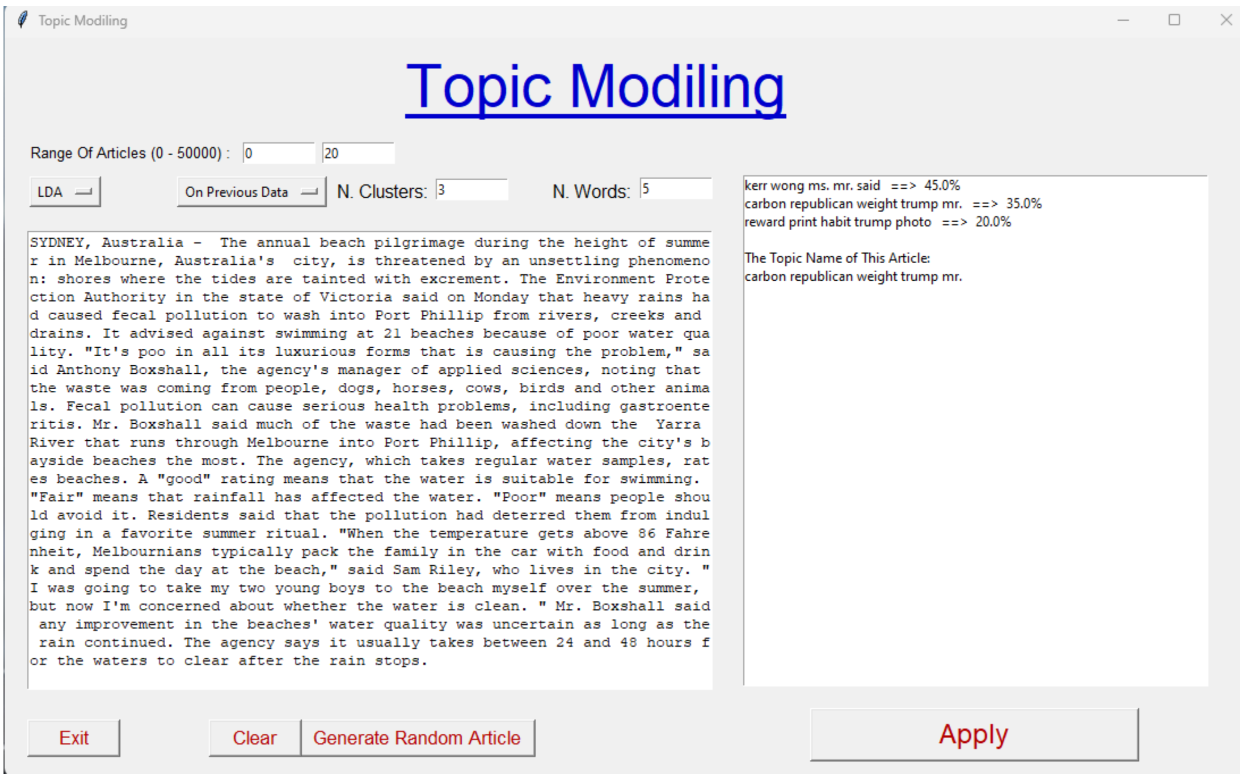
NLP Project Team (T007)

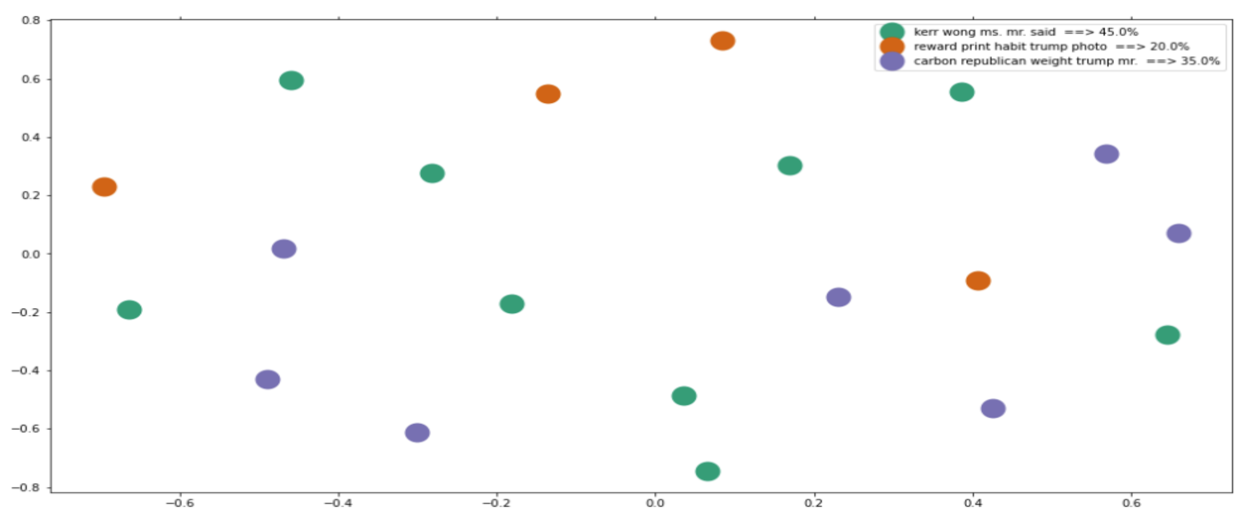
Student name	IDs	Section
بسنت حسام محمد عبدالفتاح	20201701323	2
حسام حاتم عبدالباقي محمد	20201701235	3
كريم جمال ابراهيم مرزوق	20201700603	6
علي وليد علي عبدالمنعم	20201700512	5
على محمد عبد المطلب احمد	20201701107	5

Table of Contents

Project Description

Topic modeling is a part of NLP that is used to determine the topic title for each similar group of documents based on the content. To achieve this in our project we used Clustering and Topic Modeling algorithms. Five algorithms have been used which are LDA, K-means, Mini-batch K-means, NMF, and LSA. In the following sections we will clarify each one of them in detail.





Preprocessing

Data reading and cleaning

Here we aimed to handle minor issues in the database including reformatting the data and removing the nulls.

```
def read_data(path = "articles1.csv", selected_column = 'content'):
    #Read Data set
    Data = pd.read_csv(path, encoding='latin-1')

    #Selecting required columns and rows
    Data = Data[[selected_column]]

    #Cleaning Data
    Data = Data[pd.notnull(Data[selected_column])]
    Data[selected_column] = Data[selected_column].str.replace('â\x80\x99',"'")
    Data[selected_column] = Data[selected_column].str.replace('â\x80\x98',"'")
    Data[selected_column] = Data[selected_column].str.replace('â\x80\x9c','"')
    Data[selected_column] = Data[selected_column].str.replace('â\x80\x9d','"')
    Data[selected_column] = Data[selected_column].str.replace('â\x80\x94','-')
    Data[selected_column] = Data[selected_column].str.replace('â\x80¦','...')
    Data[selected_column] = Data[selected_column].str.replace('â\x80¢','•')
    Data[selected_column] = Data[selected_column].str.replace('Ã©','é')
    Data[selected_column] = Data[selected_column].str.replace('Ã³','ó')
    Data[selected_column] = Data[selected_column].str.replace('Ã¼','ü')
    Data[selected_column] = Data[selected_column].str.replace('Ã¡','á')
    Data[selected_column] = Data[selected_column].str.replace('_____','')
    Data[selected_column] = Data[selected_column].str.replace("'  '",'')
    Data[selected_column] = Data[selected_column].str.replace('   ',' ')
    Data[selected_column] = Data[selected_column].str.replace('    ',' ')

    # Convert dataframe to list
    data = Data[selected_column].tolist()

    return data
```

```
# read data from files and clean it.
Data = read_data("articles1.csv", 'content')
```

Feature extraction

Feature Extraction Makes Machine Learning More Efficient. It cuts through the noise, removing redundant and unnecessary data. This frees machine learning programs to focus on the most relevant data.

```
def tokenize_and_stem_single(text):
    tokens = [word for word in nltk.word_tokenize(text)]
    filtered_tokens = []
    for token in tokens:
        if (re.search('[a-zA-Z]', token) and token.casefold() not in stop_words and not re.search('\W', token)):
            filtered_tokens.append(token)
    stems = [stemmer.stem(t) for t in filtered_tokens if t]
    return stems
```

```
def tf_idf(topics = []):
    tfidf_vectorizer = TfidfVectorizer(
        max_df=0.95,
        #max_features=200000,
        min_df=0.05,
        stop_words='english',
        #use_idf=True,
        tokenizer=tokenize_and_stem_single,
    )

    #fit the vectorizer to data
    matrix = tfidf_vectorizer.fit_transform(topics)
    terms = tfidf_vectorizer.get_feature_names()

    return matrix, terms
```

Clustering and Topic Modeling algorithms

I. Mini-batch K-means

The Mini Batch K-means algorithm is a variation of the traditional K-means algorithm that uses smaller random subsets or batches of data points to update cluster centers at each iteration. This approach makes the algorithm computationally efficient, particularly for large datasets. The Mini Batch K-means algorithm approximates the results of K-means while reducing the time and memory requirements, making it suitable for real-time and online clustering tasks.

```python
def mini_batch_kmeans_model(topics, matrix, terms, num_clusters=3, num_words_of_name=5):
    #Running clustering algorithm
    model = MiniBatchKMeans(n_clusters=num_clusters)
    model.fit(matrix)

    #final clusters
    clusters = model.labels_.tolist()
    topic_data = {'topic': topics, 'cluster': clusters }
    frame = pd.DataFrame(topic_data, columns = ['cluster'])

    # Sorted Clusters (bigger to smaller) by number of docs per cluster
    categories = frame['cluster'].value_counts().keys()

    #sort cluster centers by proximity to centroid
    order_centroids = model.cluster_centers_.argsort()[:, ::-1]

    # printing top names for topic
    names=[]
    for i in range(len(categories)):
        name=""
        num_words_of_name = min(len(order_centroids[categories[i]]), num_words_of_name)
        for index in order_centroids[categories[i], :num_words_of_name]:
            name+= (terms[index] + " ")
        names.append(name)

    return names, frame, categories
```

II. LSA Model

LSA (Latent Semantic Analysis) is a technique used for analyzing relationships between documents and terms within a large corpus. It represents documents and terms as vectors in a high-dimensional space and reduces the dimensionality to capture latent semantic meaning. LSA algorithm identifies patterns of word co-occurrence and similarity to uncover underlying semantic relationships in textual data.

```python
def LSA_model(topics, matrix, terms, num_clusters=3, num_words_of_name=5):

    model = TruncatedSVD(n_components = num_clusters)
    model.fit(matrix)

    topic_results = model.transform(matrix)

    #final clusters
    clusters = topic_results.argmax(axis=1)
    topic_data = {'topic': topics, 'cluster': clusters }
    frame = pd.DataFrame(topic_data, columns = ['cluster'])

    # Sorted Clusters (bigger to smaller) by number of docs per cluster
    categories = frame['cluster'].value_counts().keys()

    # printing top names for topic
    names=[]
    for index in categories:
        name=""
        topic = model.components_[index]
        num_words_of_name = min(len(topic), num_words_of_name)
        for i in topic.argsort()[-num_words_of_name:]:
            name += (terms[i]+" ")
        names.append(name)

    return names, frame, categories
```

III. NMF Model

NMF (Non-Negative Matrix Factorization) is a matrix factorization technique used for dimensionality reduction and feature extraction. It assumes that the input matrix consists of non-negative values and decomposes it into two non-negative matrices representing a low-rank approximation of the original data. NMF algorithm extracts interpretable features by enforcing non-negativity constraints, making it useful for tasks such as text mining and image processing.

```
def NMF_model(topics, matrix, terms, num_clusters=3, num_words_of_name=5):

    model = NMF(n_components = num_clusters, random_state=42)
    model.fit(matrix)

    topic_results = model.transform(matrix)

    #final clusters
    clusters = topic_results.argmax(axis=1)
    topic_data = {'topic': topics, 'cluster': clusters }
    frame = pd.DataFrame(topic_data, columns = ['cluster'])

    # Sorted Clusters (bigger to smaller) by number of docs per cluster
    categories = frame['cluster'].value_counts().keys()

    # printing top names for topic
    names=[]
    for index in categories:
        name=""
        topic = model.components_[index]
        num_words_of_name = min(len(topic), num_words_of_name)
        for i in topic.argsort()[-num_words_of_name:]:
            name += (terms[i]+" ")
        names.append(name)

    return names, frame, categories
```

IV. LDA Model

LDA (Latent Dirichlet Allocation) is a probabilistic generative model used for topic modeling. It assumes that documents are composed of multiple topics, and each word in a document is generated from one of these topics. LDA algorithm infers the latent topic structure in each set of documents and assigns the most probable topics to each word.

```
def LDA_model(topics, matrix, terms, num_clusters=3, num_words_of_name=5):

    model = LatentDirichletAllocation(n_components=num_clusters, random_state=42)
    model.fit(matrix)

    topic_results = model.transform(matrix)

    #final clusters
    clusters = topic_results.argmax(axis=1)
    topic_data = {'topic': topics, 'cluster': clusters }
    frame = pd.DataFrame(topic_data, columns = ['cluster'])

    # Sorted Clusters (bigger to smaller) by number of docs per cluster
    categories = frame['cluster'].value_counts().keys()

    # printing top names for topic
    names=[]
    for index in categories:
        name=""
        topic = model.components_[index]
        num_words_of_name = min(len(topic), num_words_of_name)
        for i in topic.argsort()[-num_words_of_name:]:
            name += (terms[i]+" ")
        names.append(name)

    return names, frame, categories
```

V. K-means Model

The K-means algorithm is an iterative clustering algorithm used to partition a dataset into k distinct clusters. It assigns each data point to the cluster with the nearest mean, iteratively updating the cluster centers until convergence. The algorithm aims to minimize the within-cluster sum of squared distances, resulting in compact and well-separated clusters.

```python
def kmean_model(topics, matrix, terms, num_clusters=3, num_words_of_name=5):
    #Running clustering algorithm
    model = KMeans(n_clusters=num_clusters)
    model.fit(matrix)

    #final clusters
    clusters = model.labels_.tolist()
    topic_data = {'topic': topics, 'cluster': clusters }
    frame = pd.DataFrame(topic_data, columns = ['cluster'])

    # Sorted Clusters (bigger to smaller) by number of docs per cluster
    categories = frame['cluster'].value_counts().keys()

    #sort cluster centers by proximity to centroid
    order_centroids = model.cluster_centers_.argsort()[:, ::-1]

    # printing top names for topic
    names=[]
    for i in range(len(categories)):
        name=""
        num_words_of_name = min(len(order_centroids[categories[i]]), num_words_of_name)
        for index in order_centroids[categories[i], :num_words_of_name]:
            name+= (terms[index] + " ")
        names.append(name)

    return names, frame, categories
```

Conclusion

We had started this project with the aim of determining the title for each cluster of articles and then assign the article used as test data in its suitable cluster. As we managed to implement the requirements successfully, we now are able to say that we proved that the Topic modeling algorithms, the LSA, LDA, and NMF, has approximately similar accuracy, which is higher than the general clustering algorithms accuracy, K-means, Mini-batch K-means.