COMP4432 Machine learning
Group: A PLUS

PAN Yalu 20080027D
XU Hulu 20075015D
YANG Yifan 20075243D

1 Introduction

A Million News Headlines Dataset is the news headlines published over a period of 19 years. The dataset gives both publish date and headline text in the format csv file, which is popular for text analytics, natural language processing, and topic modelling tasks.

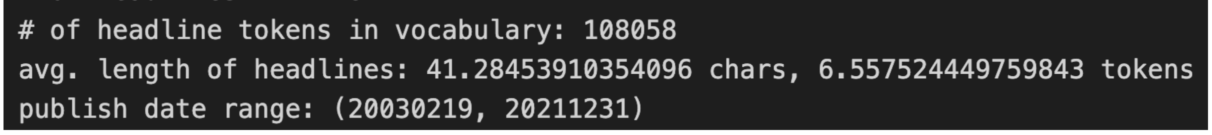
In this report, we use several topic modelling methods, including the baselines and neural topic models, to observe the clustering results. We aim to explore these methods' efficiency and the process of natural language understanding. The report is structured as follows: the dataset's basic statistics; how we preprocess the data; the implementation of 2 baseline models (LSA, LDA), a neural topic model (BERT) and autoencoder + K-means clustering; the evaluation of performance, improvement for the basic models and finally the conclusion.

2 Data Analysis

As the content of the dataset has text of the headline in Ascii, English, Lowercase, it is a problem of analysis Natural Language Processing which may make what we learnt from the course such as clustering, Text ML into real practice. Therefore, it's important to analyse and preprocess the data in a good way.

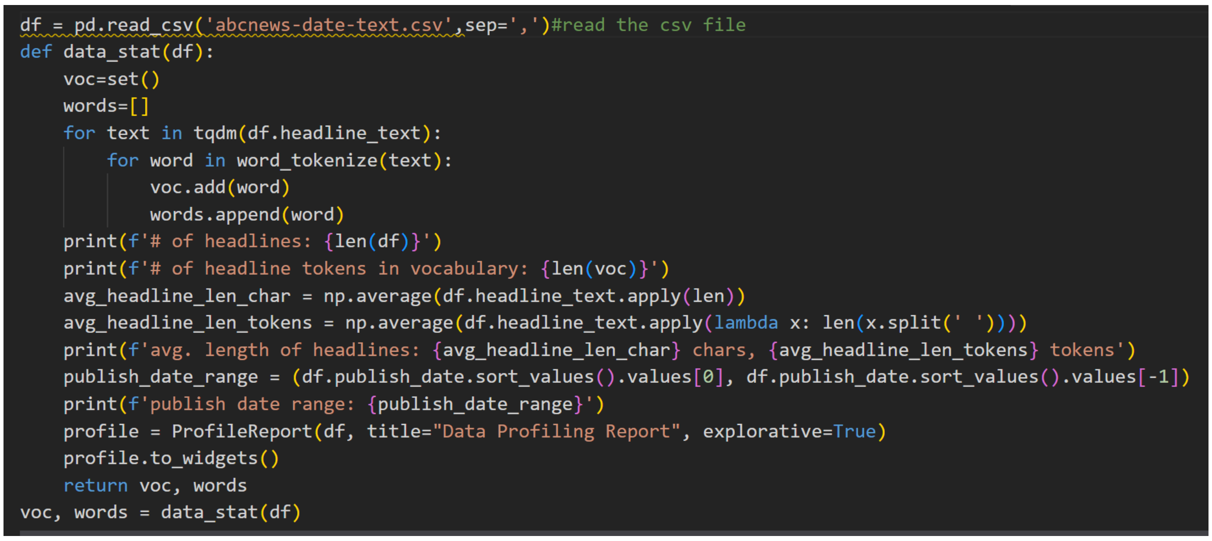
The dataset is available in CSV format and consists of 2 columns: “publish_date” and “headline_text”. The “publish_date” is the publication date of the news article, and the “headline_text” is the headline of the news article.

Performing detailed data analysis is crucial in understanding the data structure. The basic data statistic is shown below:



```
# of headlines: 1244184
# of headline tokens in vocabulary: 108058
avg. length of headlines: 41.28453910354096 chars, 6.557524449759843 tokens
publish date range: (20030219, 20211231)
```

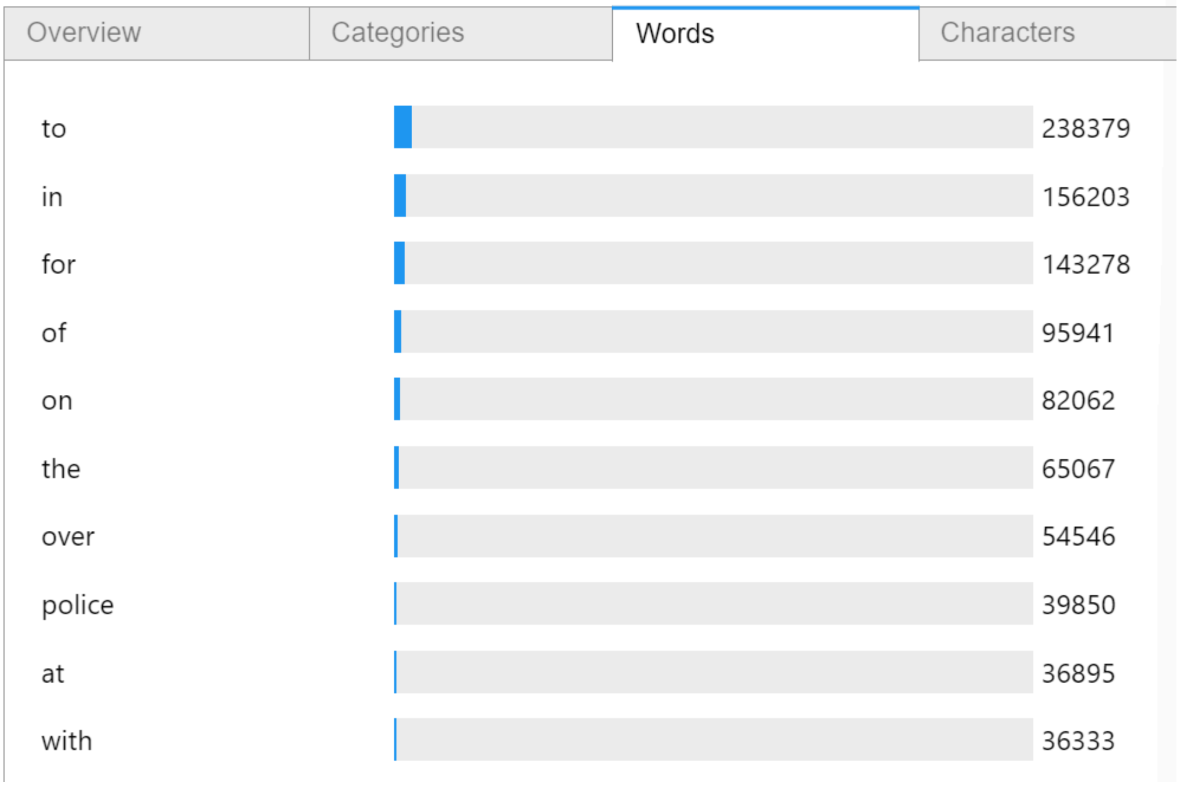
Data analysis involves examining various aspects of the data, such as the number of headlines that we need to analyse and the number of the tokens. To accomplish this, we can leverage the power of the Python powerful natural language processing library, nltk. With the help of the function *word_tokenize* under package *nltk.tokenize*, we can simply find the numbers of tokens.



```
df = pd.read_csv('abcnews-date-text.csv',sep=',')#read the csv file
def data_stat(df):
    voc=set()
    words=[]
    for text in tqdm(df.headline_text):
        for word in word_tokenize(text):
            voc.add(word)
            words.append(word)
    print(f'# of headlines: {len(df)}')
    print(f'# of headline tokens in vocabulary: {len(voc)}')
    avg_headline_len_char = np.average(df.headline_text.apply(len))
    avg_headline_len_tokens = np.average(df.headline_text.apply(lambda x: len(x.split(' '))))
    print(f'avg. length of headlines: {avg_headline_len_char} chars, {avg_headline_len_tokens} tokens')
    publish_date_range = (df.publish_date.sort_values().values[0], df.publish_date.sort_values().values[-1])
    print(f'publish date range: {publish_date_range}')
    profile = ProfileReport(df, title="Data Profiling Report", explorative=True)
    profile.to_widgets()
    return voc, words
voc, words = data_stat(df)
```

Codes for getting the data information

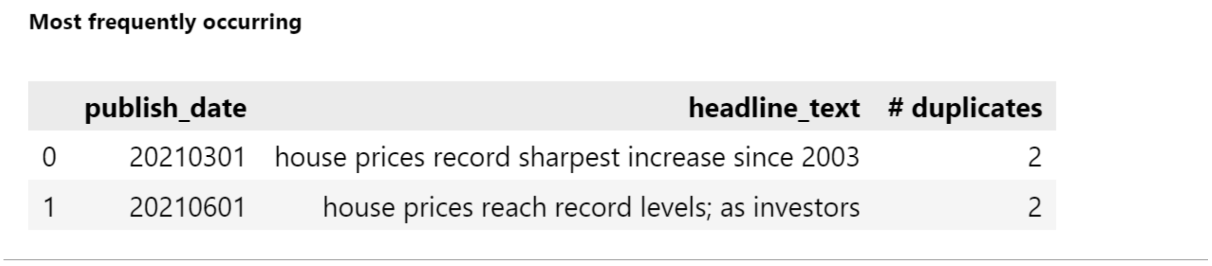
With the help of function *ProfileReport* under package *pandas_profiling*, we can make the information of all data in this dataset clear about the information in the formation of a report. For example, we can intuitively find out which words have the most frequency in the headline text in the dataset which are more likely to be filtered out later for that they are much insignificant than other words.



to	238379
in	156203
for	143278
of	95941
on	82062
the	65067
over	54546
police	39850
at	36895
with	36333

Most frequent words given by the report

Moreover, it can help us find out whether there are duplicate pieces of data which should be removed to ensure that each piece of data is unique and suitable for analysing performance. For this dataset, it has two pieces of data with two duplicates each.



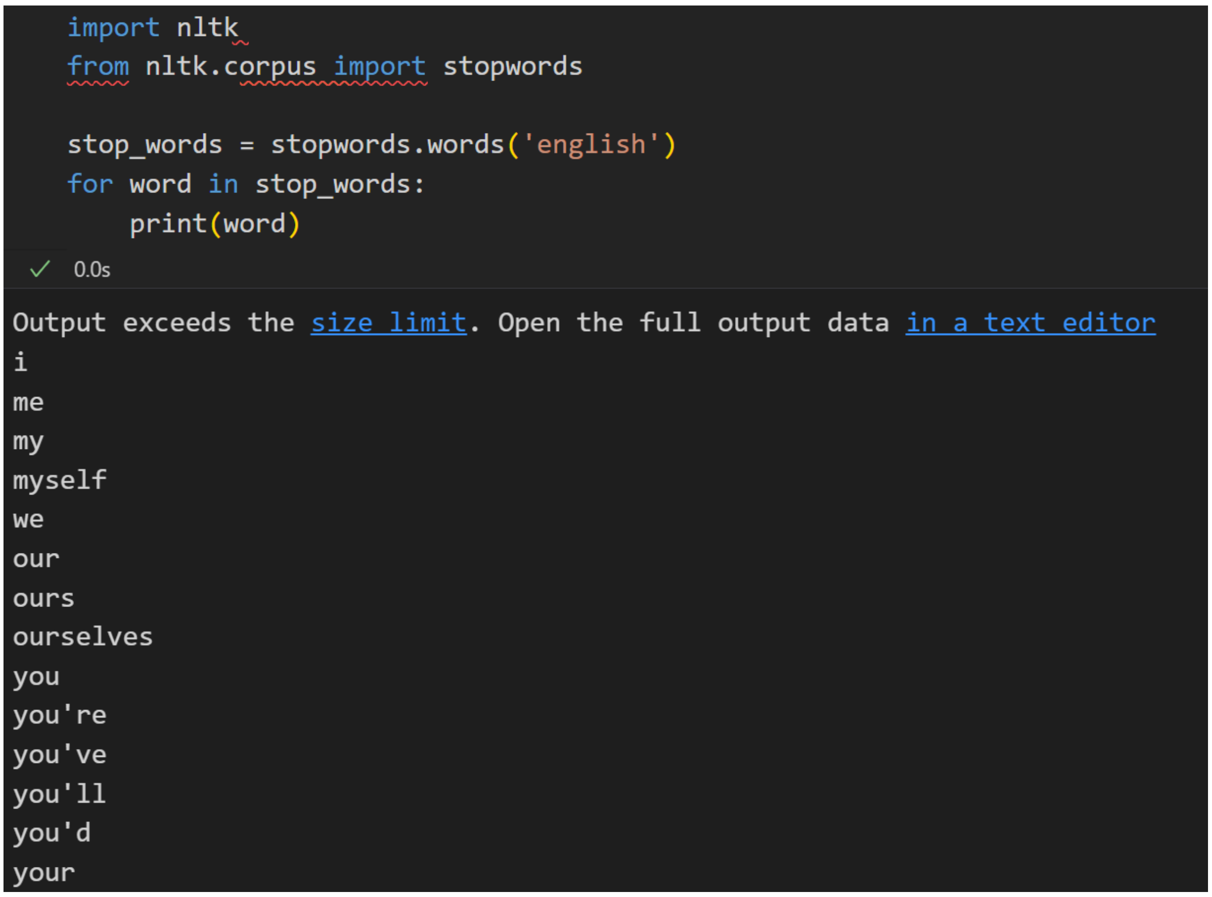
Most frequently occurring

	publish_date	headline_text	# duplicates
0	20210301	house prices record sharpest increase since 2003	2
1	20210601	house prices reach record levels; as investors	2

Duplicates found by the data report

3 Data Preprocessing

For the mostly appeared words we see from the analysis part, we can figure out that the preposition words are more likely to appear and with less importance. Therefore we can use the library given by the nltk.corpus to import stopwords which is a list of words which are common but usually provide no useful information for text analysis.



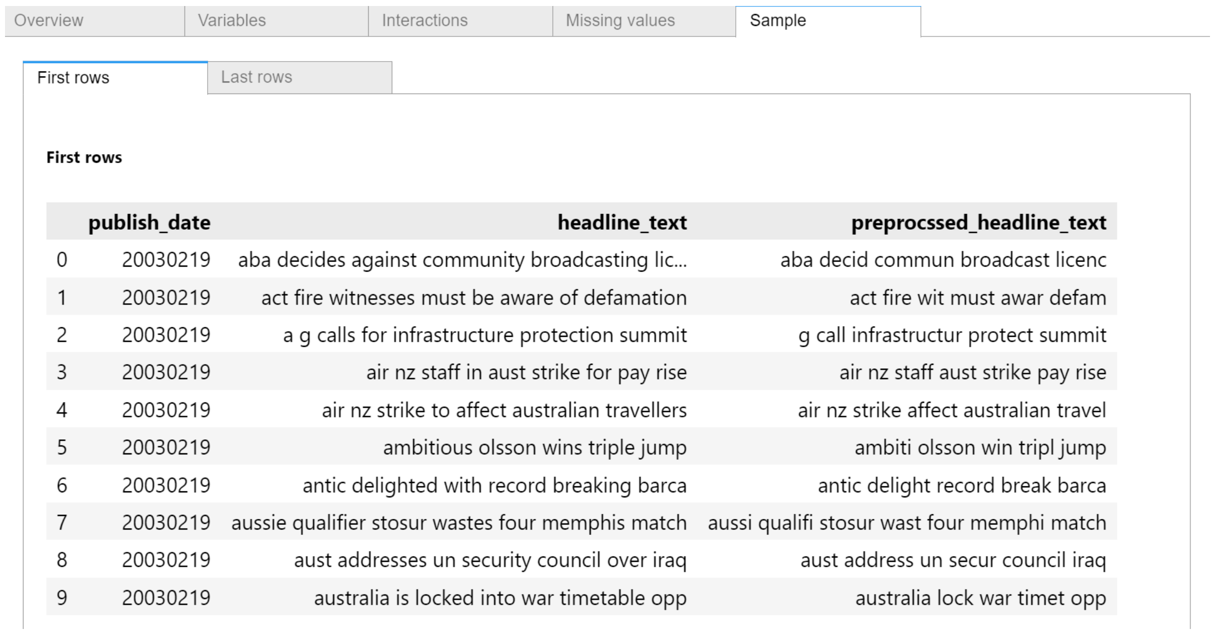
```
import nltk
from nltk.corpus import stopwords

stop_words = stopwords.words('english')
for word in stop_words:
    print(word)
```

✓ 0.0s

```
Output exceeds the size limit. Open the full output data in a text editor
i
me
my
myself
we
our
ours
ourselves
you
you're
you've
you'll
you'd
your
```

Stopwords list prepared in library

After filtering out the unimportant words, we need to make the text be more terminology standard by using the *PorterStemmer* function which is also in the library of nltk. To get the processed dataset, we still need to address remaining duplicate data which we had figured out before by removing it from the dataset. By doing these steps, we can get the processed data.



First rows

	publish_date	headline_text	preprocssed_headline_text
0	20030219	aba decides against community broadcasting lic...	aba decid commun broadcast licenc
1	20030219	act fire witnesses must be aware of defamation	act fire wit must awar defam
2	20030219	a g calls for infrastructure protection summit	g call infrastructur protect summit
3	20030219	air nz staff in aust strike for pay rise	air nz staff aust strike pay rise
4	20030219	air nz strike to affect australian travellers	air nz strike affect australian travel
5	20030219	ambitious olsson wins triple jump	ambiti olsson win tripl jump
6	20030219	antic delighted with record breaking barca	antic delight record break barca
7	20030219	aussie qualifier stosur wastes four memphis match	aussi qualifi stosur wast four memphi match
8	20030219	aust addresses un security council over iraq	aust address un secur council iraq
9	20030219	australia is locked into war timetable opp	australia lock war timet opp

Sample both the initial data and preprocessed data

4 Models implementation

4.1 Applying LDA and LSA in Topic Modeling

In this section, we discuss two popular approaches to topic modeling: Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA). Both methods have been widely used for analyzing news headlines to discover underlying topics.

4.1.1 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) is a generative probabilistic model introduced by Blei, Ng, and Jordan in 2003. LDA assumes that documents are composed of a mixture of topics, where each topic is a probability distribution over words. The primary objective of LDA is to estimate the topic-word and document-topic distributions.

LDA begins by initializing the topic assignments for each word in the documents randomly. It then iteratively refines these assignments by sampling new topic assignments for each word, taking into account the current assignments of other words in the same document and the words in the same topic. This process is continued until a convergence criterion is met, resulting in a set of topics that best explain the observed documents.

In the context of news headlines, LDA can be used to automatically discover the main themes present in a large corpus of headlines. The output of LDA can be used to gain insights into the news landscape, identify emerging trends, or provide a high-level summary of the news corpus.

4.1.2 Latent Semantic Analysis (LSA)

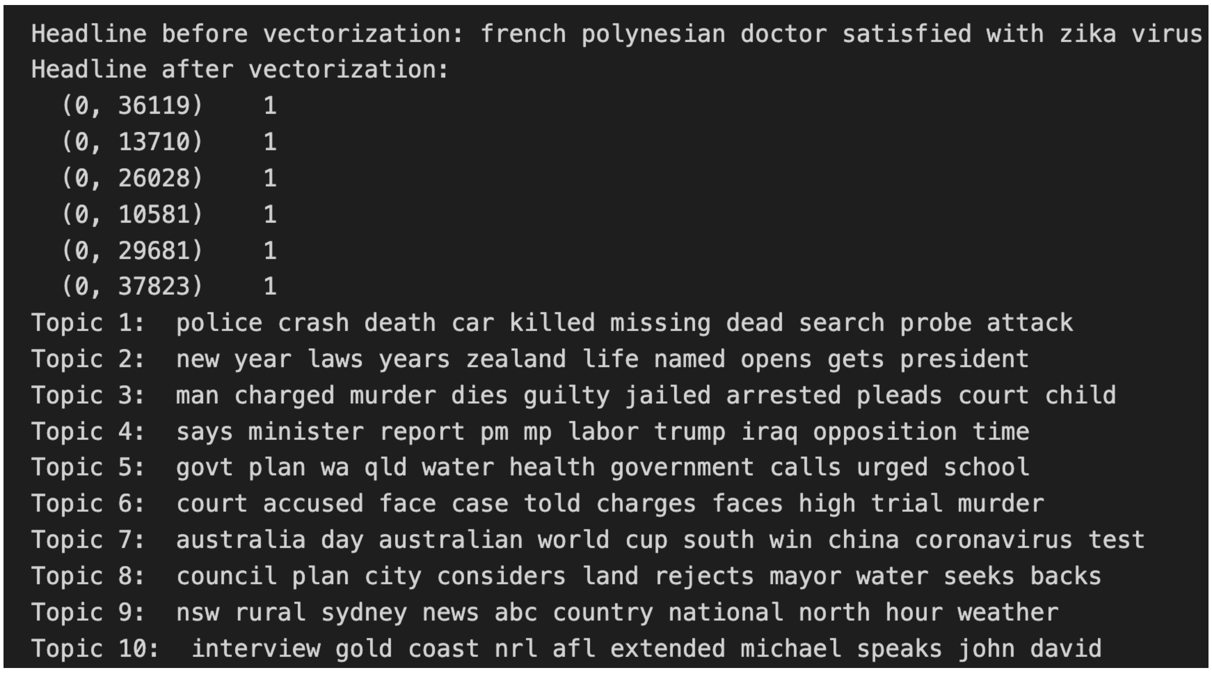
Latent Semantic Analysis (LSA), also known as Latent Semantic Indexing (LSI), is a linear algebra-based technique that aims to capture the underlying semantic structure of a collection of documents. LSA was introduced by Deerwester et al. in 1990 as a method to overcome the limitations of the vector space model, which is sensitive to the use of synonyms and polysemy in documents.

LSA employs Singular Value Decomposition (SVD) to decompose the term-document matrix into three lower-dimensional matrices: the term-topic matrix, the singular values matrix, and the topic-document matrix. By keeping only the top-k singular values and corresponding columns/rows from the decomposed matrices, LSA reduces the dimensionality of the original term-document matrix while retaining the most significant semantic information.

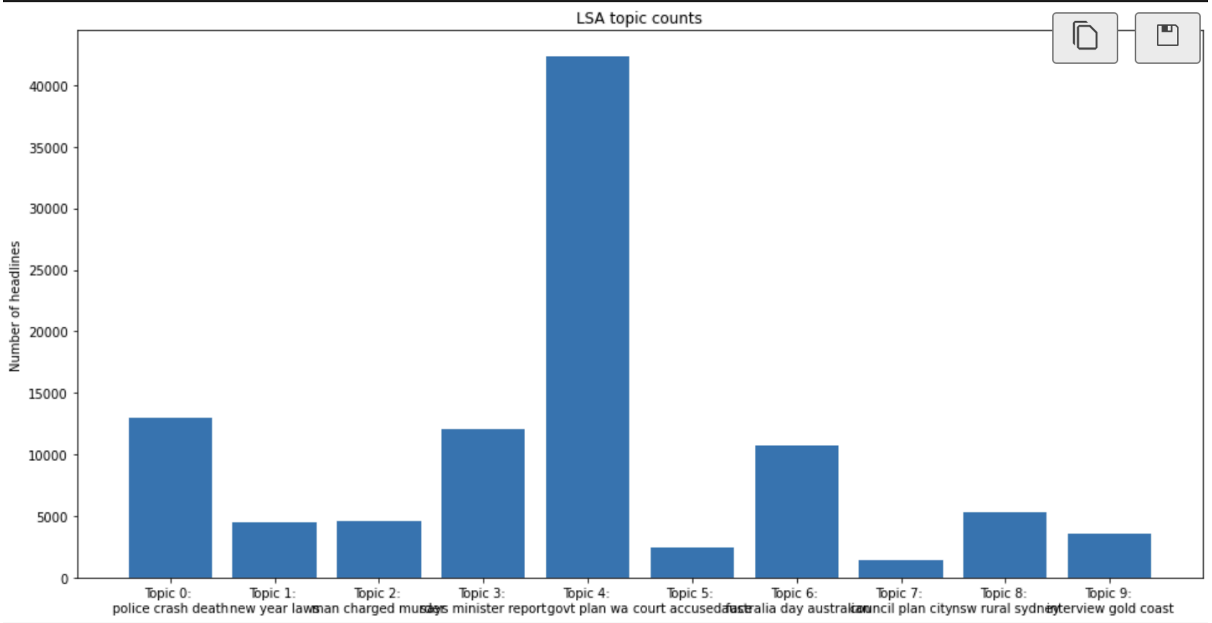
In the context of news headlines, LSA can be used to identify latent topics and relationships between headlines that are not readily apparent. The reduced dimensional representation of headlines can be used for clustering, classification, or visualization purposes.

4.1.3 Result

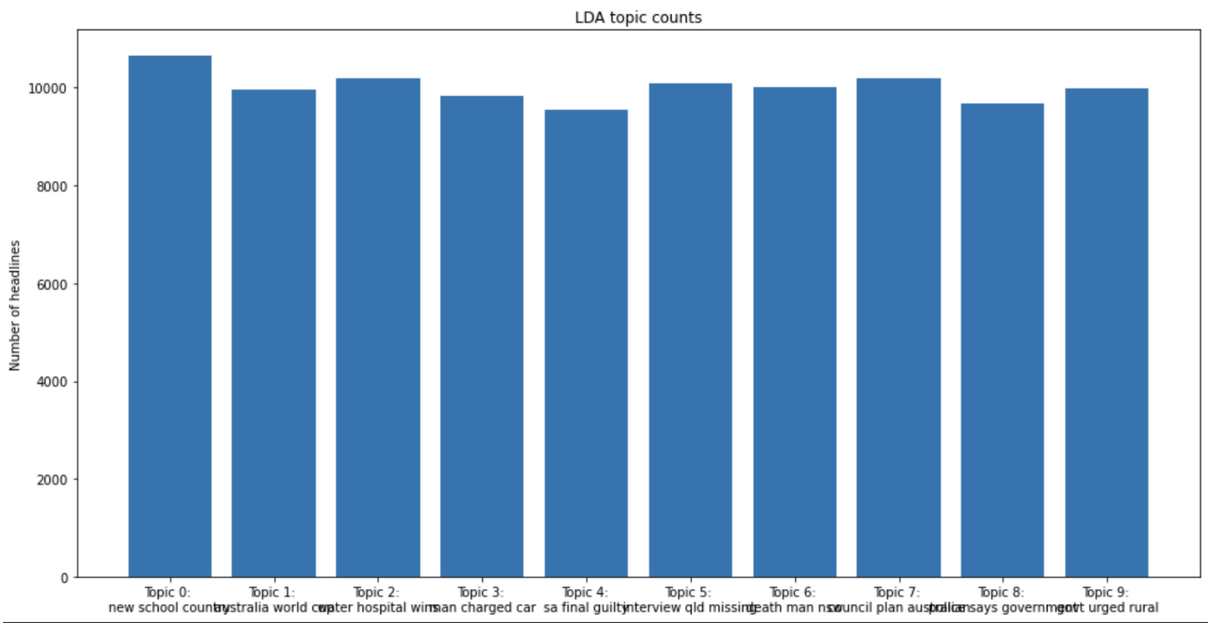
The headline before vectorization is shown below:



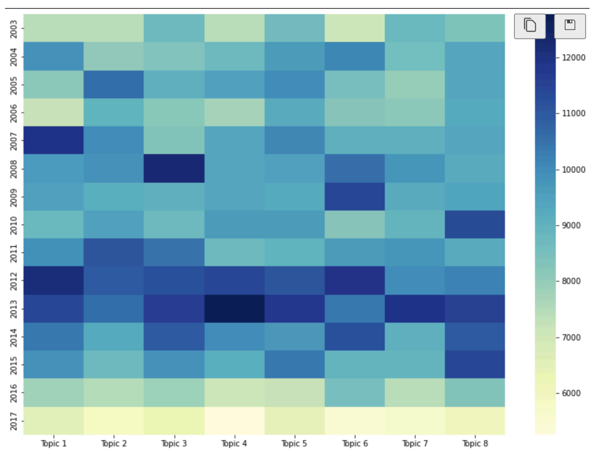
```
Headline before vectorization: french polynesian doctor satisfied with zika virus
Headline after vectorization:
  (0, 36119)	1
  (0, 13710)	1
  (0, 26028)	1
  (0, 10581)	1
  (0, 29681)	1
  (0, 37823)	1
Topic 1:  police crash death car killed missing dead search probe attack
Topic 2:  new year laws years zealand life named opens gets president
Topic 3:  man charged murder dies guilty jailed arrested pleads court child
Topic 4:  says minister report pm mp labor trump iraq opposition time
Topic 5:  govt plan wa qld water health government calls urged school
Topic 6:  court accused face case told charges faces high trial murder
Topic 7:  australia day australian world cup south win china coronavirus test
Topic 8:  council plan city considers land rejects mayor water seeks backs
Topic 9:  nsw rural sydney news abc country national north hour weather
Topic 10:  interview gold coast nrl afl extended michael speaks john david
```

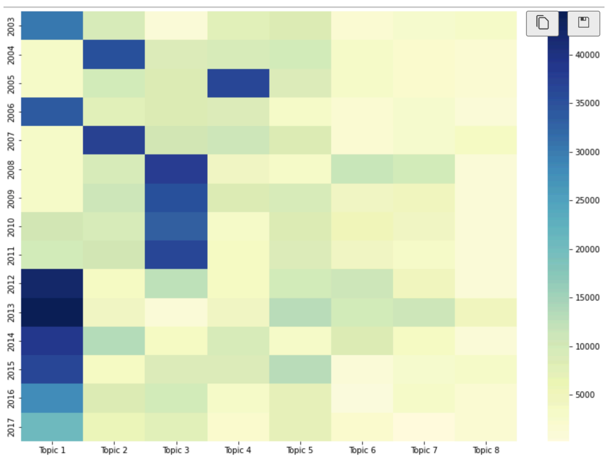


LSA topic counts



LDA topic counts





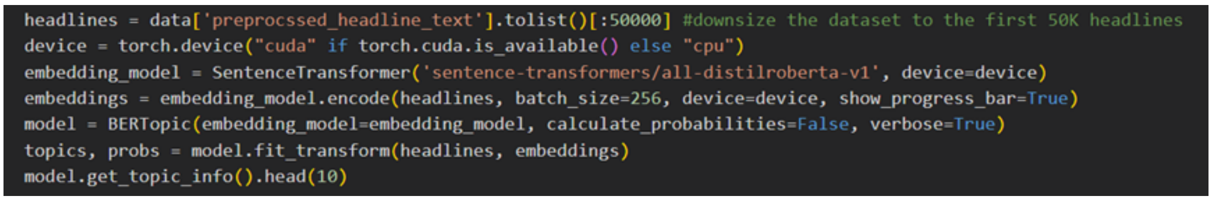
In summary, both LDA and LSA are powerful techniques for topic modeling in the context of news headlines. While LDA is a probabilistic model that explicitly models the generation of documents, LSA is a linear algebra-based technique that relies on SVD. Both approaches have been successfully applied to uncover underlying topics in news headlines, providing valuable insights into the structure and content of news corpora.

4.2 BERTopics

BERT is short for Bidirectional Encoder Representations from Transformers which is a family of masked-language models. It is widely used for analysis of natural language processing.

To perform a thorough analysis of the data, we should do the data preprocessing first which we had mentioned before which can ensure that our analysis is accurate, reliable, and meaningful.

BERT can generate high-quality embeddings that capture the meaning and context of words and sentences, which can then be used for clustering. To perform clustering using BERT embeddings, we first generate the embeddings for each data point, which can be achieved using the BERT model's encoding function.



```
headlines = data['preprocssed_headline_text'].tolist()[:50000] #downsize the dataset to the first 50K headlines
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
embedding_model = SentenceTransformer('sentence-transformers/all-distilroberta-v1', device=device)
embeddings = embedding_model.encode(headlines, batch_size=256, device=device, show_progress_bar=True)
model = BERTopic(embedding_model=embedding_model, calculate_probabilities=False, verbose=True)
topics, probs = model.fit_transform(headlines, embeddings)
model.get_topic_info().head(10)
```

Codes for implementing BERTopic model

After generating the datasets, we try to use c-TF-IDF which refers to Class-based TF-IDF to represent the dataset's performance. In c-TF-IDF, instead of considering the frequency of each word across all documents, we consider the frequency of each word within each class or group of related documents. This adjusted representation can give the distance between documents of different clusters.

c-TF-IDF

For a term **x** within class **c**:

$$W_{x,c} = \|\mathbf{tf}_{x,c}\| \times \log\left(1+\frac{A}{f_x}\right)$$

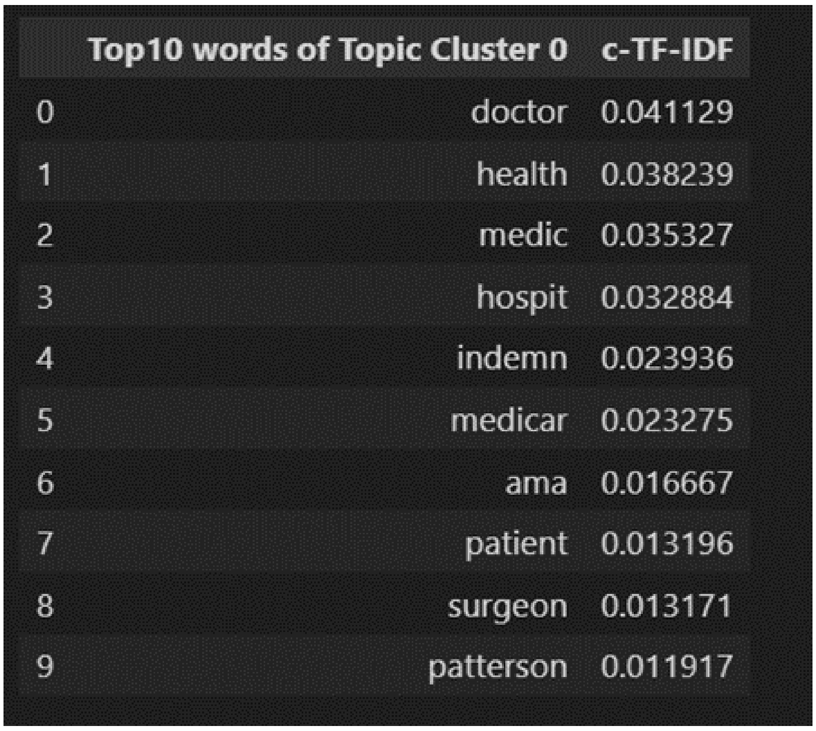
$\mathbf{tf}_{x,c}$ = frequency of word **x** in class **c**

$\mathbf{f}_x$ = frequency of word **x** across all classes

A = average number of words per class

Definition of c-TF-IDF[1]

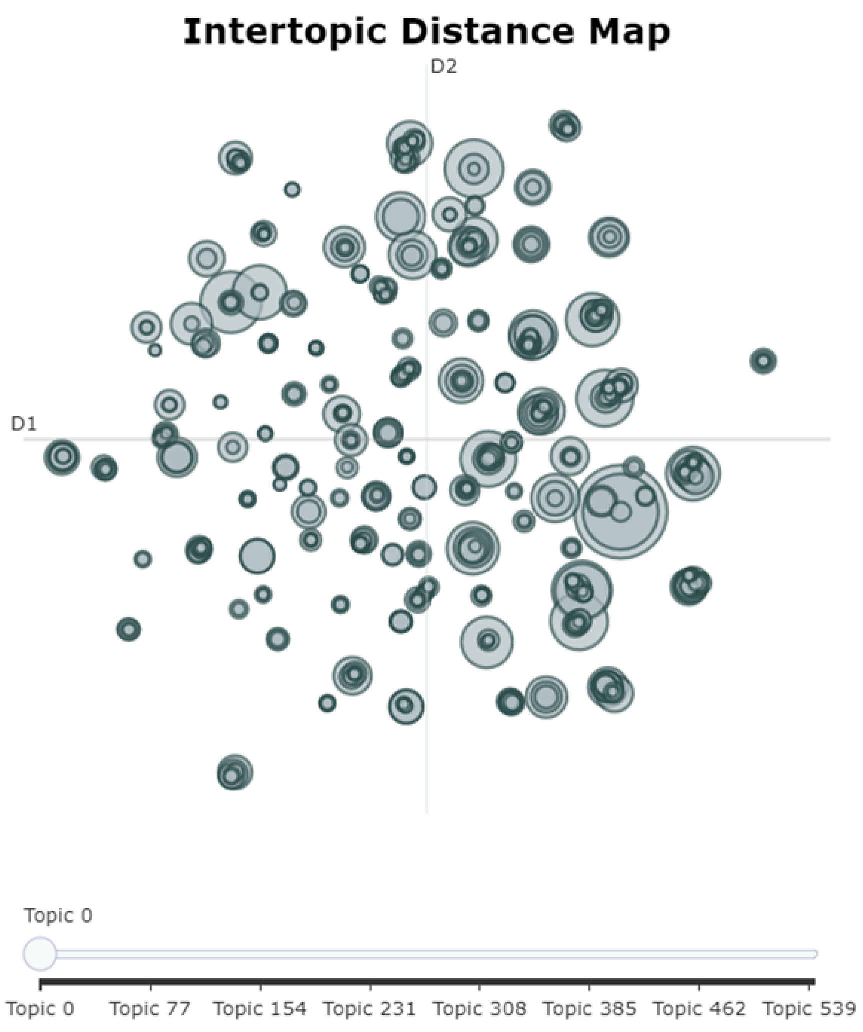
So by calculating the c-TF-IDF of each word, the model can do the clustering which makes different words become a group with similar performance.



	Top10 words of Topic Cluster 0	c-TF-IDF
0	doctor	0.041129
1	health	0.038239
2	medic	0.035327
3	hospit	0.032884
4	indemn	0.023936
5	medicar	0.023275
6	ama	0.016667
7	patient	0.013196
8	surgeon	0.013171
9	patterson	0.011917

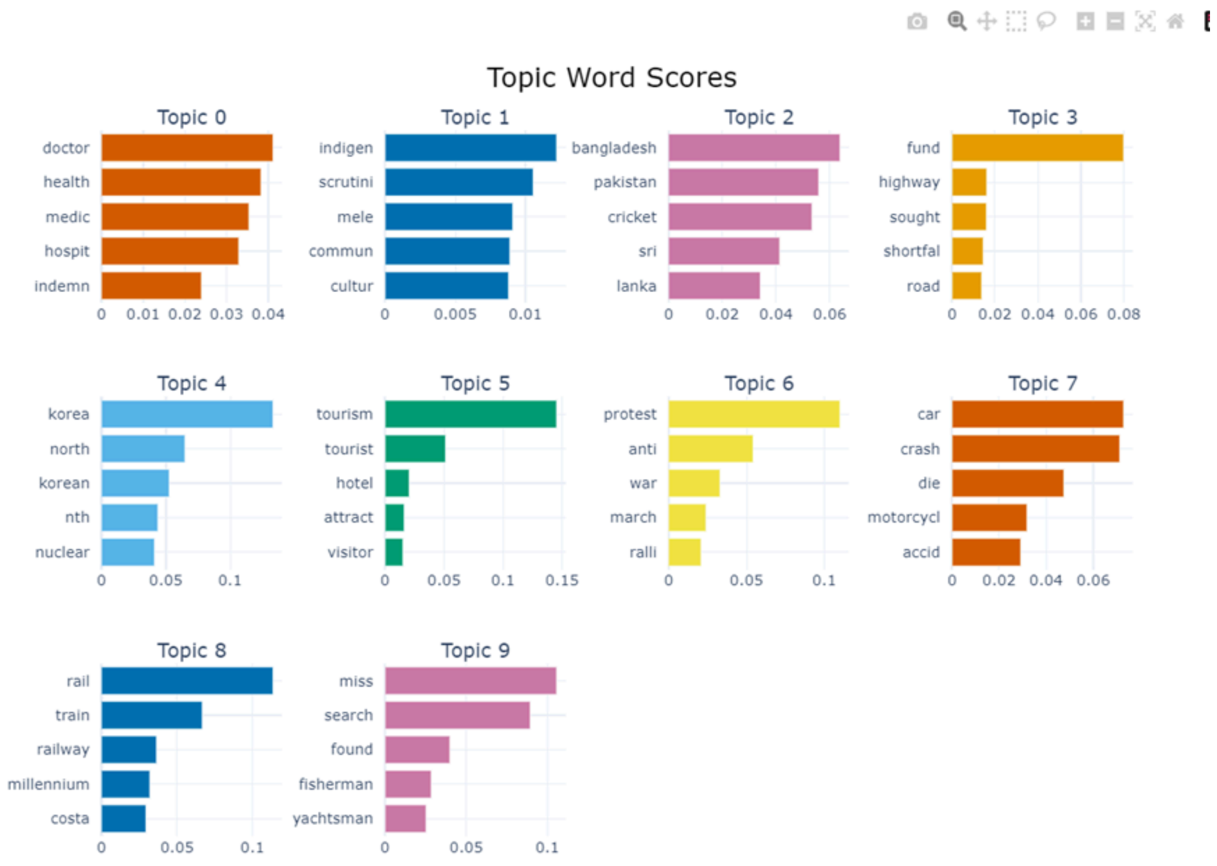
Top10 words of Topic Cluster0 with their c-TF-IDF

By using the model function which can make the clustering visualised to us, it is clearer to topic number and top names. By visualising the clustering results, we can gain a better understanding of the underlying patterns and structure of the data, as well as identify the top topics and associated keywords.



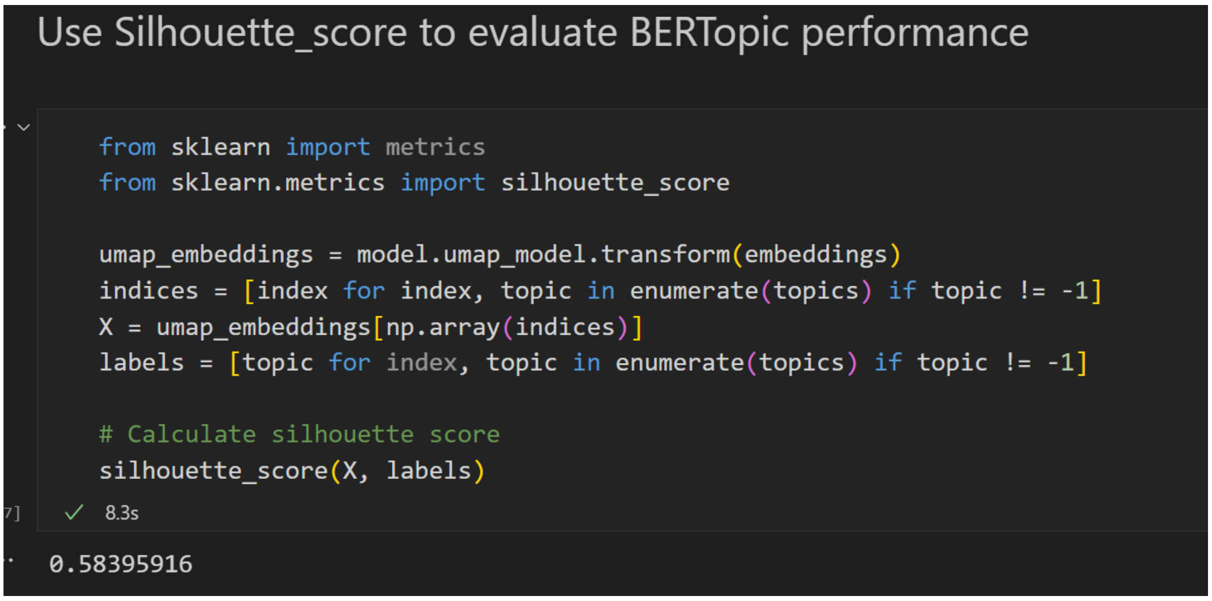
Output of visualise_topic function

By using another function prestored in the BERTopic caller *visualize_barchart*, we can intuitively get the barchart of different words in each Topic and see clearly the difference of scores between every two words in the same topic.



Barchart of the first 10 topics with the higher c-TF-IDF words

As there is silhouette_score for us to evaluate the model performance for clustering that is a method of interpretation and validation of consistency within clusters of data. For BERTopic we use the function *silhouette_score* under library package *sklearn.metrics* to calculate this score for getting the evaluation.



Use Silhouette_score to evaluate BERTopic performance

```
from sklearn import metrics
from sklearn.metrics import silhouette_score

umap_embeddings = model.umap_model.transform(embeddings)
indices = [index for index, topic in enumerate(topics) if topic != -1]
X = umap_embeddings[np.array(indices)]
labels = [topic for index, topic in enumerate(topics) if topic != -1]

# Calculate silhouette score
silhouette_score(X, labels)
```

✓ 8.3s

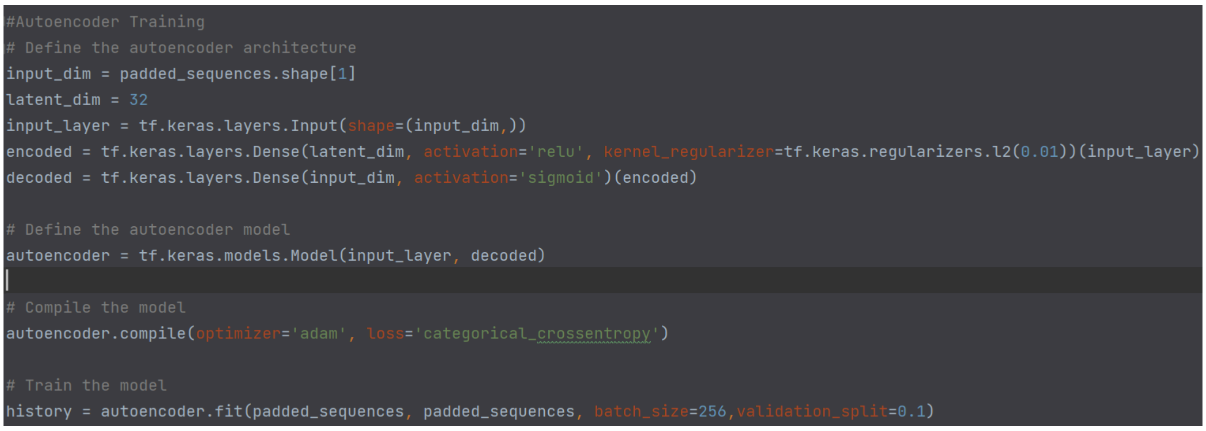
```
0.58395916
```

Sihouette_score for BERTopic

4.3 Autoencoder + K-means clustering

In this model, it combines the deep learning model and K-means clustering model to complete the topic clustering task.

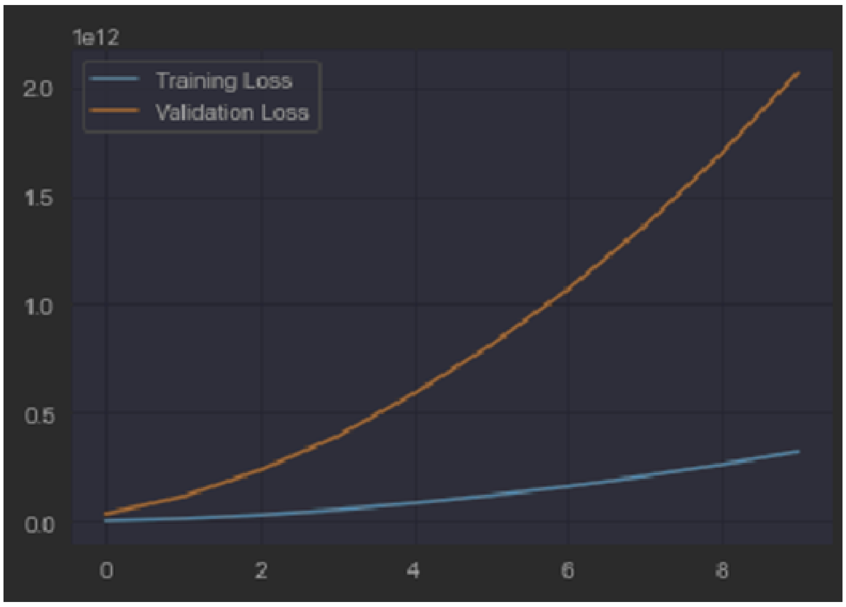
Firstly, it uses Autoencoder to compress the headline token. An autoencoder is a deep learning neural network. It can learn to compress data into a lower-dimensional representation. Then reconstructs the original data from the compressed representation. It can be useful in the task of topic modelling.



```python
#Autoencoder Training
# Define the autoencoder architecture
input_dim = padded_sequences.shape[1]
latent_dim = 32
input_layer = tf.keras.layers.Input(shape=(input_dim,))
encoded = tf.keras.layers.Dense(latent_dim, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01))(input_layer)
decoded = tf.keras.layers.Dense(input_dim, activation='sigmoid')(encoded)

# Define the autoencoder model
autoencoder = tf.keras.models.Model(input_layer, decoded)

# Compile the model
autoencoder.compile(optimizer='adam', loss='categorical_crossentropy')

# Train the model
history = autoencoder.fit(padded_sequences, padded_sequences, batch_size=256,validation_split=0.1)
```

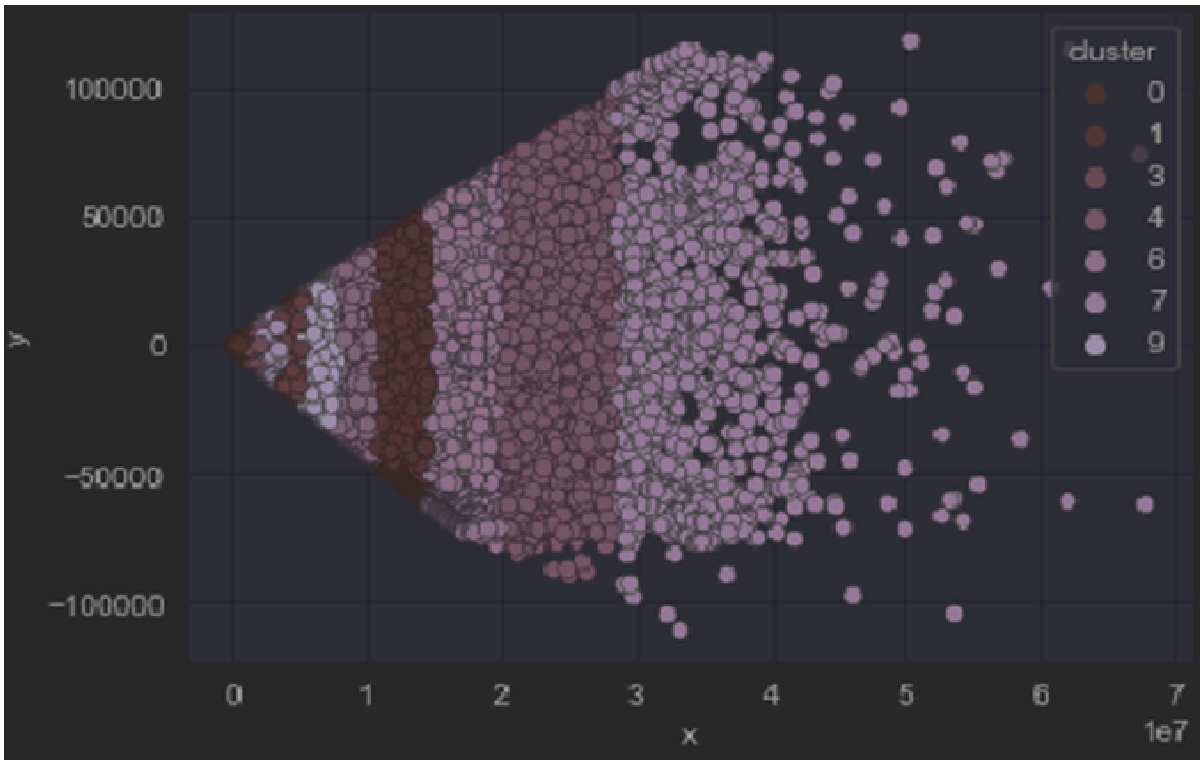
Codes for modelling

In our autoencoder construction, the input layer is the padded sequence of tokens with the same length. The output layer is a reconstruction of the input sequence. Between these two layers, there is a single hidden layer with 32 neurons. The autoencoder is compiled using the Adam optimizer and categorical cross-entropy loss. It trained for 1 epochs using the processed headlines as both the input and output. Also, I use L2 regularization to the encoding layer of the autoencoder with strength of “0.01”. We can see that the loss is increasing with increasing epochs, it could be a sign of overfitting. That is why I use L2 regularization and reduce the epochs to 1, it can help me to prevent the model from overfitting and improve the performance.



Loss of the autoencoder

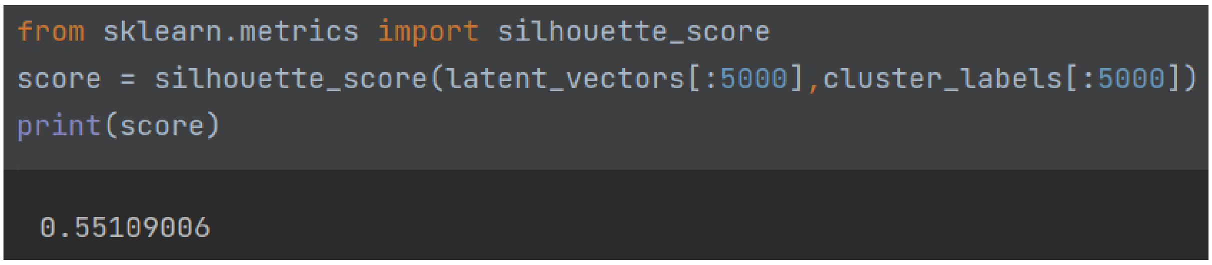
After training the autoencoder, the encoder portion of the model is used to extract the latent vectors (i.e., the compressed representations) of the headlines. These latent vectors are then clustered using K-means with 10 clusters.



Latent vector and cluster

We use a scatter plot to show the relationship between the latent vectors and the cluster result. The first step is to reduce the dimensionality of the latent vectors using TruncatedSVD and obtain a 2D representation for visualisation. This allows us to visualise how well the points from different clusters are separated in the 2D space.

We can see that there is almost no mix in each cluster group, it means that the performance of the clustering task is relatively good. But the difference in the number of each cluster is very high.



```
from sklearn.metrics import silhouette_score
score = silhouette_score(latent_vectors[:5000],cluster_labels[:5000])
print(score)
```

```
0.55109006
```

Cluster silhouette coefficient score

Then we use silhouette coefficient score to evaluate the model and clustering result. Due to the computer's hardware limitations, we just use the first 5000 data to do the evaluation. Finally, we get a silhouette coefficient score of 0.55.

5 Evaluation

Evaluating the performance of a topic model includes intrinsic and extrinsic methods. Intrinsic methods evaluate the model using the model's internal structure or the data it was trained on, while extrinsic methods evaluate the model through conducting some tasks, such as document classification and information retrieval.

Silhouette coefficient score

The silhouette coefficient score is a measure of how similar an object is to its own cluster compared to other clusters. It is used to evaluate the quality of clustering results.The score ranges from -1 to 1. For a score of 1, it means that the object is very similar to its own cluster and dissimilar to other clusters. For a score of -1, it indicates the opposite. We use it to evaluate the performance of clustering models.

Model	Score
BERTopics	0.58
Autoencoder + K-means	0.55

6 Improvement

In the BERTopic model implementation, we use c-TF-IDF instead of the basic TF-IDF to compute the performance of the text dataset. Since the words appear in different headlines which are phrases having certain meanings and we want to do the clustering for those words, it seems more accurate to use the scores which work on a cluster or topic level instead of a document level.

For using K-means clustering, instead of directly using the token sequence generated by Keras Tokenizer, we additionally add a deep learning autoencoder to compress the text token sequence. It can generate more important features of a token to get better performance.

7 Conclusion

In this project, we used a few models including LSA, LDA, Autoencode+K-means, and BERTopics to do the NLP problem of topic modelling.

In general, the performance of BERTopics and Autoencode is good with a Silhouette coefficient score of 0.58 and 0.55. It shows that these two methods may not perform perfectly in this task.

References

[1] Maarten, *c-TF-IDF*, 2023. [Online] Available:https://maartengr.github.io/BERTopic/getting_started/ctfidf/ctfidf.html#reduce_frequent_words [Accessed May. 3, 2023].