Executive Summary

The term classification in context to the data science world is a technique to predict a particular set of classes for the provided or retrieved data set. The classified classes in the project are called categories. In scientific terms, there are various input variables considered as 'X' and a distinct result variable as 'y'. Classification is supervised learning as it is an algorithm executed by learning from a training dataset. The execution of the classification algorithm on the training dataset is concluded when the model achieves a certain level of acceptance and accuracy. For example, the classes for a particular entity could be colors as 'Red,' 'Blue,' and 'Green.'

The dataset is obtained by using the New York Times API. The data set contains various input variables that contribute to determining classes in terms of news categories. The obtained dataset undergoes numerous data preprocessing steps. First, using TF-IDF, the relevancy measured as a score in the text analysis is used for the training data. Then, depending on the collected data, classification algorithms are used to predict the classifiers. The project uses five classification algorithms to get the best accuracy on the trained data model as follows:

- Random Forest
- Multinomial Logistic Regression
- Bi-directional LSTM
- Decision Tree
- Gaussian Naïve Bayes classifier

The project conducts the machine learning process to predict the news category for the New York Times tweet, making it simpler and swifter for the user to traverse through a plethora of tweets as per the interest. The project aims to create a hybrid solution comprising both analytical and information system.

Introduction

The New York Times is dated back to 1851 and is one of the most prestigious newspapers with an international reach. The New York Times has been famous for the editorial content and rich culture and intellectual approach to the news article published in the newspaper. The New York Times progressed early with providing the content and archives on the internet by 1995. The New York times has a global and extensive reach with 49.9 million followers to its Twitter handle. There are categories available for the articles to be separated on the website. Still, when it comes down to the tweets, they frequently handle tweets, but the original description has no category. The project aims to predict the apt category for the tweets from the Twitter hand of The New York Times.

The reports and project focus on using the archives using the API by The New York Times to analyze and classify the articles in 11 odd categories. The report research on the following questions as follows:

- The op-ed section is a platform for authors to communicate personality by storytelling and appealing experiences. The publication is not affiliated with any articles published in this section. Does the op-ed section have these kinds of articles? Or is it just another platform where disputes are bred? Does the op-ed section need more scaling down of data and separated into distinct categories? We will analyze the articles to find out whether the op-ed section answers the above questions.
- Will the number of classes affect the accuracy of the model?
- How the features(words) and labels (categories of news) correlate with each other? What best keywords can we determine the category of news?
- Is it possible to extract any time series pattern from the data collected from the archive API and help with the categories?
- How well did the model classify? Was the accuracy acceptable?

Data Overview

The data is collected from New York Times and New York Times Twitter accounts. This API will allow the non-employers to collect the data of New York Times articles. We have built a custom scrapper that will call the New York Times archive API and scrap the data from the result of the archive API and store the data in different CSV's cased on the month and year of the news article.

The library used in Python is tweepy to get the tweets from Twitter. To access Twitter API, we need to have four keys:

Consumer key	consumer secret key	access token key	access token secret key

The New York Times profile URL has the name NYTimes, which will get the relevant tweets. An empty list has been created to store the incoming tweets. The incoming data must be preprocessed to get the output data frame. A total of 239 tweets are loaded into the data frame. These tweets will be used as an input to the final machine learning model to get the category, which the tweet belongs. The overview of tweets scraped from Twitter is as shown in the below image.

Fig 1: Data frame of tweets scrapped from New York times Twitter account.

New York Times is established in the year 1851. All the articles' data since 1851 can be collected through this API. If we pass the API along with the year and the month, we will be getting all data from the provided date. This API will help us to create our own dataset for the New York Times. We collected the data of articles of the past four years, i.e., 2017-2021. The data of every month is stored as an individual CSV file. All these files are preprocessed and combined before building a classification model. Steps to collect data are as follows:

- Load all the required dependencies to collect the data.
- Mention the starting year, month, and the ending year to get the data.
- All the CSV files of every month are preprocessed and combined before training.

For combining all the files of every month, we used the inbuilt method of Python, i.e., concat(). First, concatenation combines the files into one file. Later, we converted the concatenated file into a .csv file by formatting the extension.

Thus, the preprocessed CSV file is then used for developing the classification model.

The features used in the analysis and their description is as follows:

Headline: It consists of the text representing the nature of the article. It appears at the top of the content.

Date: It consists of the day, month, and year of an article published. (Ex 2020-10-01 represents the article published on 1st of October in 2020

Doc_type: It represents the type of document of an article. It consists of four distinct categories (article, multimedia, audio, and audio container)

Material_type: It consists of categories on which platform the article is presented to the audience. It consists of n distinct categories. (Ex: Op-Ed: printed on opposite to the editorial page, Video: A mp4 format clipping explaining the article, etc.)

Section: Articles are divided into sections to group articles with similar content. (Ex: Science: The articles focusing on advancement in Science, U.S.: The articles concentrate on activities inside the United States, etc.)

News_Desk: The type of department inside New York Times working on an article (Ex: Sports: A group of a sports a journalist assigned to collect the information for article etc.)

Abstract: A summary of an article.

Keywords: The essential words in an article. Readers usually use these keywords to find related articles.

Lead_Paragraph: The opening paragraph of an article gives the reader an introduction to the article.

 Snippet: Additional information about the article.

The data consists of 449,566 rows and ten features. The data types of features are as follows.

Variable Name	Variable Type
Headline	Object
Date	datetime64
Doc_type	Object
Material_type	Object
Section	Object
News_desk	Object
abstract	Object
Keywords	Object
Lead_paragraph	Object
snippet	Object

Table 1: Variable name in the data frame

Data Preprocessing

Sections: In the data collected, the section column is one of the essential variables because the project's overall aim is to classify the textual news data into these sections.

- **Identifying Unique Sections:** There is a total of 62 unique classes in the combined data frame.


```
sections_abstract_count = data.groupby('section')['abstract'].nunique()
print(sections_abstract_count)
print(len(sections_abstract_count))

section
Admin              201
Arts             15919
At Home            394
Automobiles         52
Blogs              135
                 ...
Watching           352
Well              2713
World            21912
Your Money         645
en Español           4
Name: abstract, Length: 62, dtype: int64
62
```

- **Merging Sections:** Sections related to each other and relevant are clubbed together to create new sections/classes to decrease the overwhelming number of classes.

```
[ ] data['section'].replace(['Well', 'Health'], 'Wellness & Health', inplace = True)
    data['section'].replace(['Fashion & Style', 'Style'], 'Fashion & Style', inplace = True)
    data['section'].replace('New York', 'U.S.', inplace = True)
    data['section'].replace(['Books', 'The Learning Network'], 'Education', inplace = True)
    data['section'].replace(['Science', 'Technology'], 'Science & Technology', inplace = True)
    data['section'].replace(['Movies', 'Theater'], 'Movies & Theater', inplace = True)
```

Fig 3: Merging sections code snippet

- **Selecting Top Sections:** The number of sections reduced to 11 after this process. The aim was to identify a balanced number of classes to improve the overall process. Therefore, we selected the sections with a value count greater than or equal to 10k.

```
[ ] data = data[data.groupby('section').section.transform('count')>=10000].copy()

[ ] (data['section'].value_counts())

    U.S.                  97292
    Opinion               48640
    World                 43940
    Arts                  33914
    Business Day          28764
    Education             24250
    Sports                23538
    Fashion & Style       22842
    Movies & Theater      17602
    Science & Technology  13212
    Wellness & Health     10762
    Name: section, dtype: int64
```

Fig 4: Top sections code snippet

Missing Values: Values like 'headline', 'material_type', 'keywords', 'snippet', 'abstract', 'news_desk', 'lead_paragraph' are important for the model considering the nature of the data being text. So, it was necessary to identify if these variables have any empty values. If there are, we need to remove the respective rows from the data not to affect the model implementation.

```
[ ]  data.isnull().any()

     headline        True
     date            False
     doc_type        False
     material_type   True
     section         False
     news_desk       True
     abstract        True
     keywords        True
     lead_paragraph  True
     snippet         True
     dtype: bool


[ ]  data.isnull().sum()

     headline            14
     date                 0
     doc_type             0
     material_type    10420
     section              0
     news_desk         9986
     abstract          1304
     keywords         26082
     lead_paragraph    3190
     snippet           7766
     dtype: int64


[ ]  data.dropna(subset=['headline','material_type', 'keywords', 'snippet','abstract','news_desk','lead_paragraph'],axis=0,inplace=True)
     data.isnull().sum()

     headline         0
     date             0
     doc_type         0
     material_type    0
     section          0
     news_desk        0
     abstract         0
     keywords         0
     lead_paragraph   0
     snippet          0
     dtype: int64
```

Fig 5: Identifying missing values code snippet

Text: This is a new feature created by merging important text features' headline', 'abstract,' 'lead_paragraph' and 'keywords.' This step was implemented so that we can pass one single document into the model.

```
data['text'] = data['headline'] + " "+ data['abstract'] + " "+ data['lead_paragraph'] +" "+ data['keywords']
data['text'][0]

'Anyone Else Want to See Trump 'Shut Up'? Our president as a terrible toddler. When the nation looks back on the presidential debate we witnessed this week, do you think it'll be remembered as: Presidential Election of 2020,Debates (Political),'
```

Fig 6: Merging text code snippet

Parsed Text: A new variable called 'parse_text' is created by converting all the words in the text feature into the lower case by removing special characters and tokenizing the stop words. Stop words in English like the, an, me, etc., are removed to reduce the redundancy of words that are not crucial for the models.

Section Encode: All the categorical values in the column 'section' are manually encoded and converted into numerical values. A new feature section_codes is created as shown below.

section_codes = {'U.S.': 0,

'Arts': 1,

'World': 2,

'Opinion': 3,

'Business Day': 4,

'Sports': 5,

'Fashion & Style': 6,

'Movies & Theater': 7,

'Education': 8,

'Science & Technology': 9,

'Wellness & Health': 10

}

One Hot Encoding: This word embedding technique is used to split the text into different words and indexed it based on the word position. The indexed positions of the words are converted into the binary matrix, which is used to signify the number of dimensions of that document. The quantified terms then act as the input for the Bi-directional LSTM models. In this scenario, pre-padding in the project helped fill dummy 0s for shorter document sizes.

```
onehot_repr=[one_hot(words,voc_size)for words in corpus]
onehot_repr
```

```
[[1189,
  885,
  1199,
  251,
  652,
  870,
  372,
  566,
  226,
  126,
  428,
  700,
  986,
  198,
  393,
  968,
  1075,
  287,
  986,
  10,
  198,
  577],
 [108,
  1072,
  538,
  148,
  1494,
  868,
  975,
  108,
  1494,
  1319,
```

Fig 7: One hot encoding code snippet

Stemming: During data preprocessing, this technique is implemented to extract the root formation of the input words. Stemming is comparatively faster but creates words that might not be meaningful when compared to lemmatization. In our case, the meaning of the language was not important, whereas we focused on classification based on keywords, so stemming was the best method.

Example: "' anyon els want to see trump shut presid terribl toddler nation look back presidenti debat wit week think rememb presidenti elect debat polit"

Term Frequency Inverse Document Frequency: TF-IDF is used to change the text document into numbers used by ML (Machine Learning) algorithms in this project. The number of word frequency to its index as a vector helps our algorithms to improve the performance–indexed word dictionaries for both test and train features as shown below.

```
[ ] ngram_range = (1,2)
    min_df = 10
    max_df = 1.
    max_features = 5000

[ ] tfidf = TfidfVectorizer(encoding='utf-8',
                            ngram_range = ngram_range,
                            stop_words = None,
                            lowercase = False,
                            max_df = max_df,
                            min_df = min_df,
                            max_features = max_features,
                            norm = 'l2',
                            sublinear_tf = True
                            )
    train_features = tfidf.fit_transform(X_train).toarray()
    train_label = y_train

    test_features = tfidf.transform(X_test).toarray()
    test_label = y_test
```

Fig 8: TFIDF code snippet

Random Under Sampling: This technique removes randomly selected values from the classes with majority data. In our case, we needed to reduce the humongous amount of data from sections like U.S., Opinion, Arts which led to an imbalance in the data. So, under-sampling helped the project to bring down the data in sections as shown below.

```
[ ] from sklearn.datasets import make_classification
    from imblearn.under_sampling import RandomUnderSampler

    under = RandomUnderSampler(sampling_strategy={0:5000,1:5000,2:5000,3:5000, 4: 5000, 5: 5000, 6:5000, 7:5000}, random_state=10)
    #under.fit(train_features, train_label)
    train_features, train_label = under.fit_sample(train_features, train_label)
```

Fig 9: Random sampling code snippet

Data Visualization

Sections vs. Article Count: The bar graph below shows the final categories in the section column plotted against the number of articles in each category.
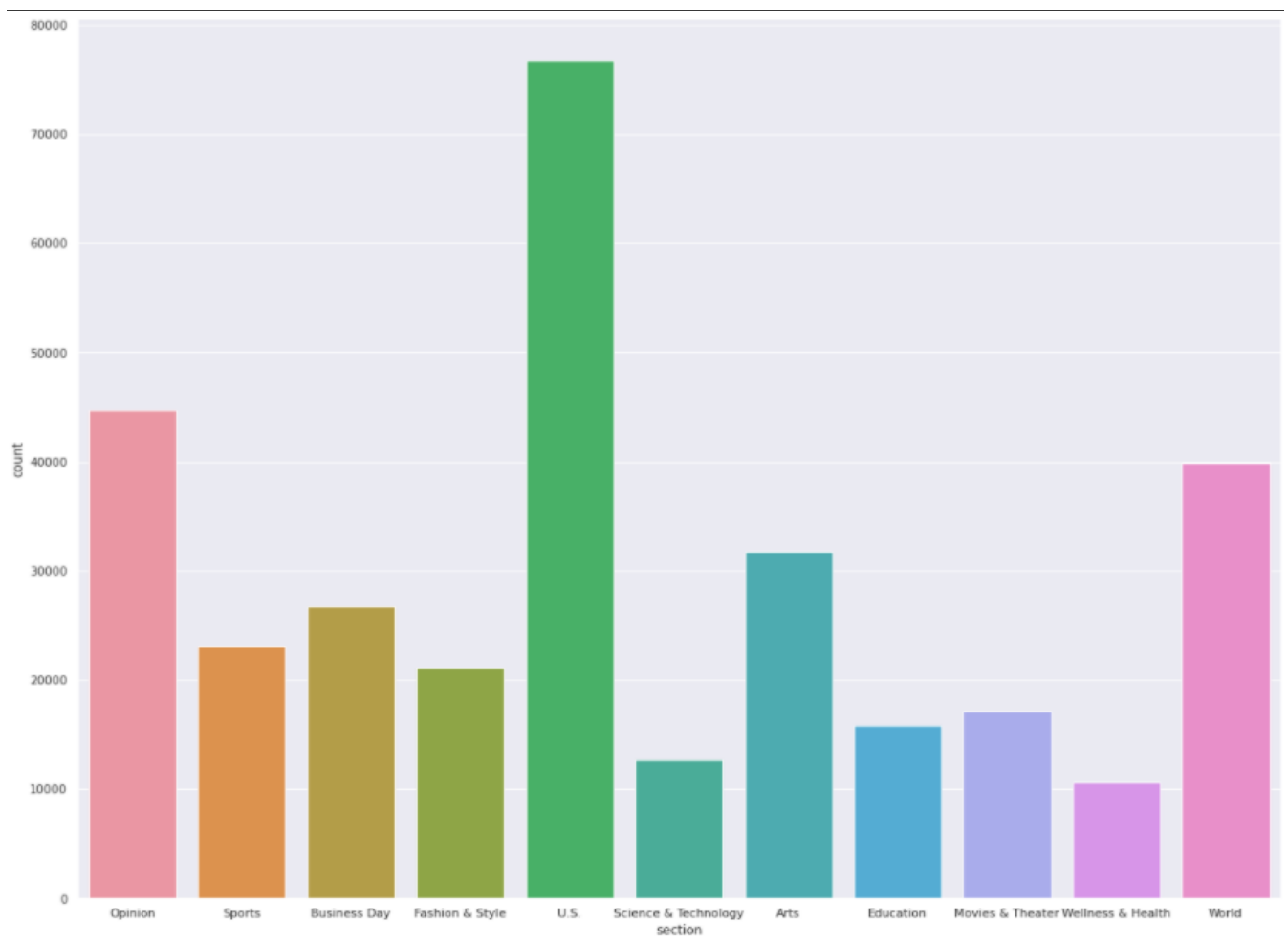
Fig 10: Section vs articles count visualization

Text Length vs. Length Count: The histogram explains the frequency of text document length with the number of bins = 70.
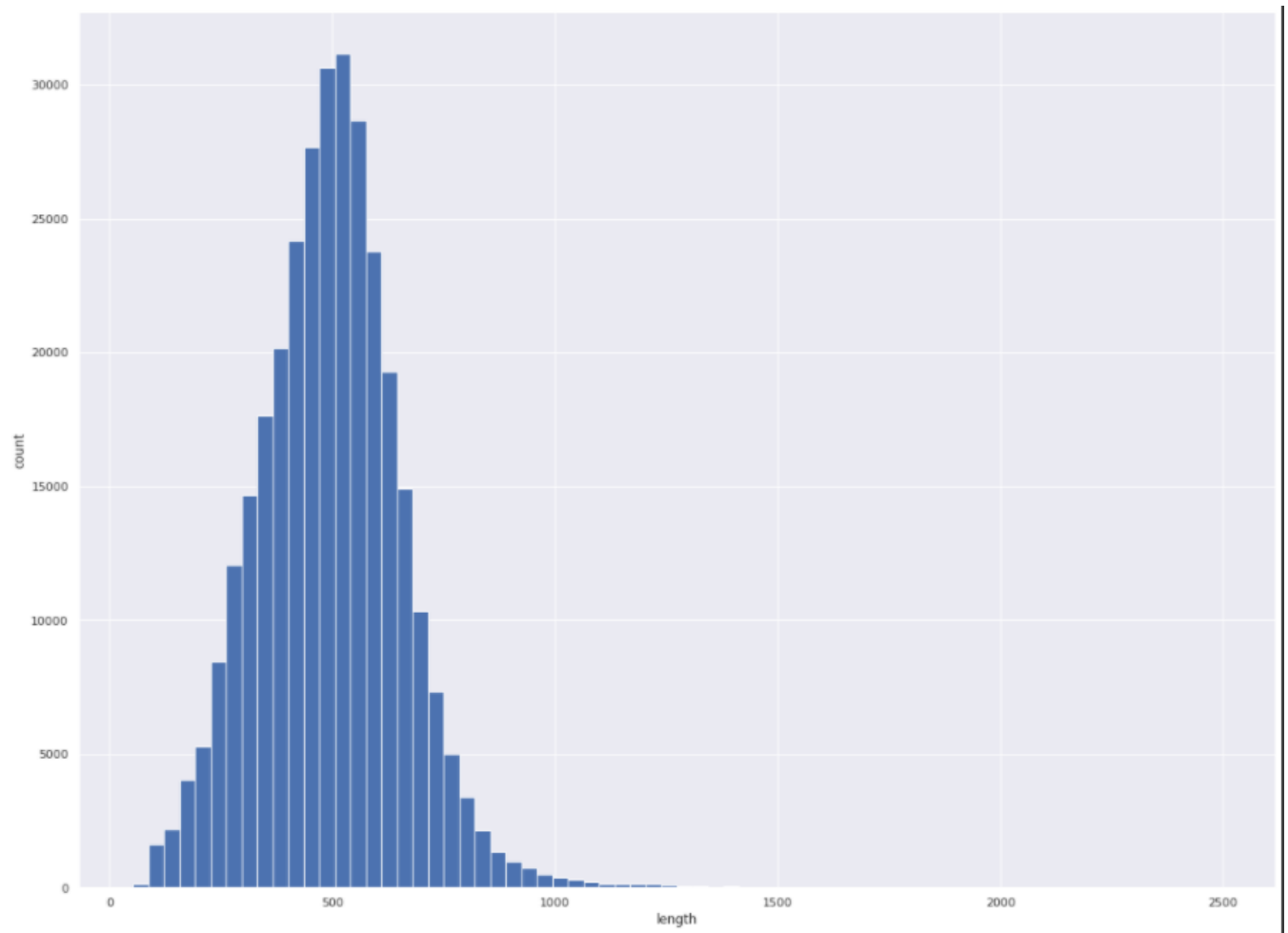
Fig 11: Text vs length count data visualization

Word Cloud: All the categories in section column are converted to word cloud. This helps us to identify essential words in the text document for each section.

Opinion:

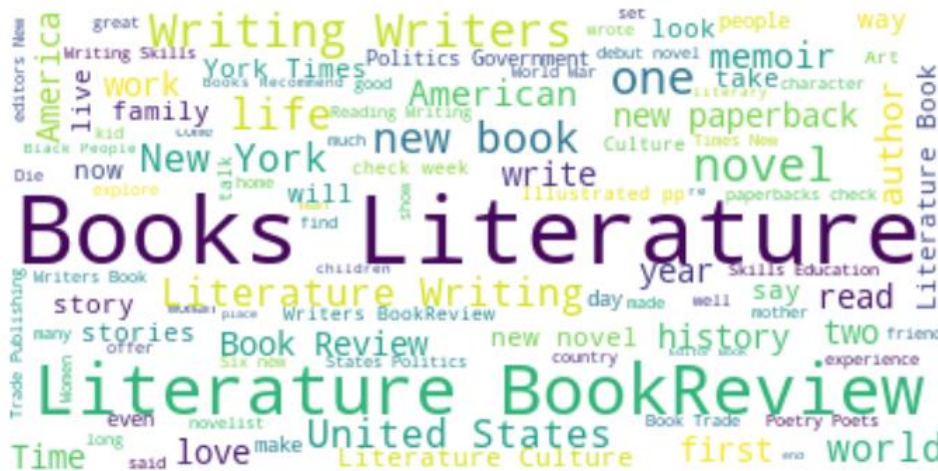Fig 12: Opinion word cloud data visualization

Education:

Fig 13: Education word cloud data visualization

Arts:

Fig 14: Arts word cloud data visualization

Fashion & Style:

Fig 15: Fashion & Style word cloud data visualization

Movies & Theatre:

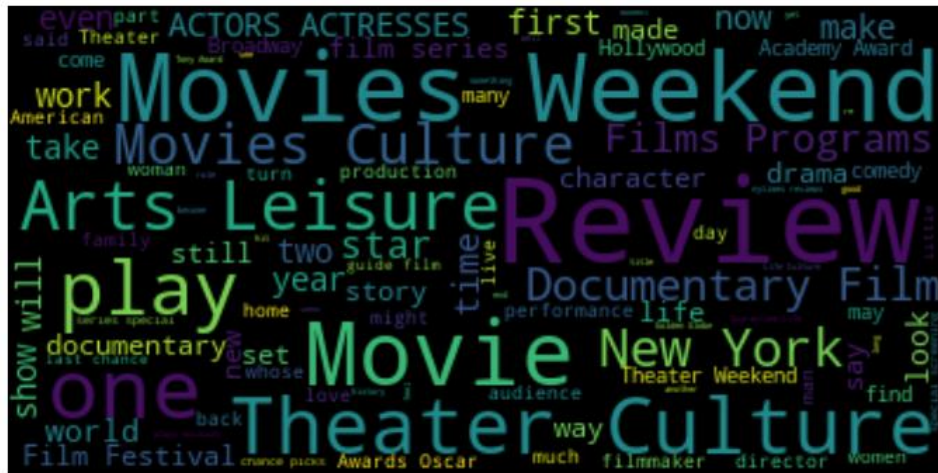Fig 16: Movies & Theatre cloud data visualization

Sports:

Fig 17: Sport word cloud data visualization

U.S.

Fig 18: U.S cloud data visualization

Science & Technology:
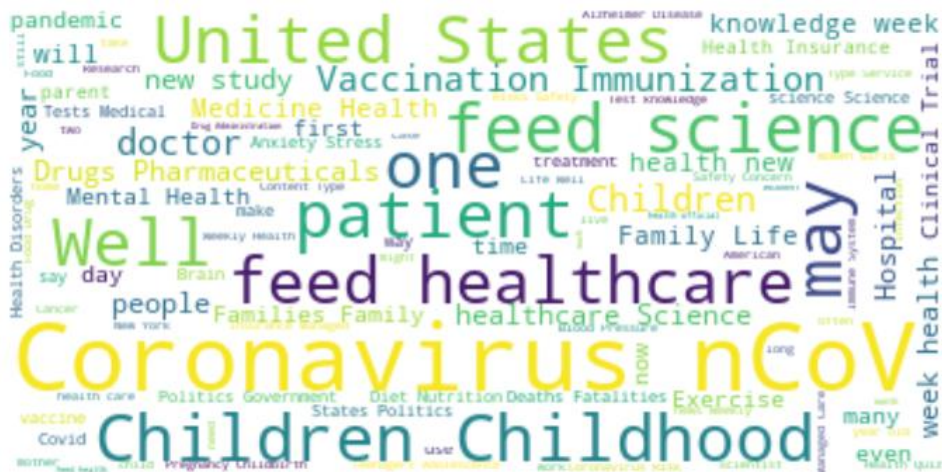

Fig 19: Science & Technology cloud data visualization

World:

Fig 20: World word cloud data visualization

Wellness & Health:

Fig 21: Wellness & Health word cloud data visualization

Business Day:

Fig 22: Business Day World word cloud data visualization

Model Building

To solve the problem of news classification, we will try several different models and select the best model based on the confusion matrix to classify tweets further. The biggest challenge we faced is about the number of categories. When we scraped the data for four years, there were more than 62 news sections. It would be hard to process the model in so many classes. Further we tried to test with multiple classes and models to see the accuracy.

Tested with classes more than 2000 record:

We tried to build the model including the classes with more than 2000 records. There were 30 classes to be classified. We got a low accuracy on the validation data about 60%. But further we tried to classify tweets by scrapping the twitters data using tweepy; the models were not suitable to classify the tweets as there was a concise length of text fed to the model.

Tested by combining the categories and more than 2000 records:

Later we tried to club the classes like Movies, podcasts, and Theater into a single class of Entertainment. Similarly, we did it for well, health, technology, science, books and learning network. Below is a snapshot that shows the clubbing of the categories.

```python
data['section'].replace(['Well', 'Health'], 'Wellness & Health', inplace = True)
data['section'].replace(['Fashion & Style', 'Style'], 'Fashion & Style', inplace = True)
data['section'].replace('New York', 'U.S.', inplace = True)
data['section'].replace(['Books', 'The Learning Network'], 'Education', inplace = True)
data['section'].replace(['Science', 'Technology'], 'Science & Technology', inplace = True)
data['section'].replace(['Movies', 'Theater'], 'Movies & Theater', inplace = True)
```

After combining these categories, we were left with 21 classes to classify. We trained multiple models for the classes like Random Forest, Bi-Directional LSTM, LSTM model, Decision tree, and Logistic regression. We got an acceptable accuracy of around 70-75%. We selected Bi-directional LSTM, Logistic regression, and Random Forest to classify the tweets further. The output for the tweet classification was not much accurate. Most of the tweets were categorized as U.S., Opinion, or Entertainment.

Final classification by under sampling and 11 classes –

After trying several combinations for the classes and testing the accuracy of the New York Times, we observed that due to an imbalance of the dataset, most of the tweets were classified incorrectly. We decided to collect the data for 4 years and club the classes to reduce the number of classes, and later, we under sampled the data for the classes with records of more than 5000. We used RandomUnderSampler from imblearn and later build multiple models and went ahead with classifying the tweets. This time the results for the tweet classification were acceptable and reasonable.

So, we further selected 11 categories to build our model. In the selection process, we dropped the classes with less than 5000 records so that our data should not be highly imbalanced, and we merged certain classes in one with a similar context. Further, we tried to solve this problem statement by using Machine learning and Deep learning.

Machine Learning
In machine learning, we have used supervised learning techniques, which involves the building of two models, which is:
Random Forest:
As the name says, random forest is an ensemble technique of decision tree formed using the bagging method. It is known to give better accuracy as we try to combine the output from every individual decision tree to predict.
To solve our Tweets classification problem, we have used Random Forest and the default tuning parameter to get the desired output. Here is a snippet of what our model looks like.

```
0.7553289504505208
              precision    recall  f1-score   support

         0       0.85      0.69      0.76     15459
         1       0.85      0.81      0.83      6354
         2       0.71      0.76      0.74      8007
         3       0.80      0.51      0.62      8866
         4       0.73      0.66      0.69      5257
         5       0.85      0.95      0.89      4658
         6       0.89      0.78      0.83      4263
         7       0.79      0.94      0.86      3379
         8       0.79      0.99      0.88      3100
         9       0.52      0.97      0.68      2586
        10       0.41      0.97      0.58      2108

  accuracy                           0.76     64037
 macro avg       0.75      0.82      0.76     64037
weighted avg     0.78      0.76      0.76     64037
```

Fig 24: Classification matrix for Random Forest

We got an accuracy of 75.53% on the test data. So, we decided to go further and test the data for tweets on N.Y. Times twitter account.

Multinomial Logistic Regression:

Multinomial logistic regression is used to classify the multiclass problem using logistic regression. Logistic regression is designed for the binary classification problem. It is used to classify the value as 0 or 1. The multinomial logistic regression is designed to provide the multinomial probability distribution, and it used the distribution to predict the class. To solve our problem statement, we have used multinomial logistic regression with the default parameters. Below is the snapshot of the classification matrix of the multinomial logistic regression model.

```
0.7874197729437669
              precision    recall  f1-score   support

         0       0.88      0.75      0.81     15459
         1       0.86      0.82      0.84      6354
         2       0.81      0.81      0.81      8007
         3       0.78      0.61      0.69      8866
         4       0.72      0.70      0.71      5257
         5       0.91      0.94      0.93      4658
         6       0.83      0.79      0.81      4263
         7       0.84      0.91      0.88      3379
         8       0.83      0.97      0.89      3100
         9       0.50      0.89      0.64      2586
        10       0.46      0.91      0.61      2108

  accuracy                           0.79     64037
 macro avg       0.77      0.83      0.78     64037
weighted avg     0.81      0.79      0.79     64037
```

Fig 25: Classification matrix for Multinomial Logistic Regression

The multinomial logistic regression has given us an accuracy of 78.74% overall for classifying different classes. We can check from the confusion matrix that the model is good in classifying class 5 with the precision of 91%, and it is a little weak in classifying class 10 with a precision of 46%

Naïve Bayes:

The working theory behind the Naive Bayes classifier is the Bayes theorem. Here we used Gaussian Naive Bayes considering Gaussian distribution. Bayes theorem states as follow: $P(A|B) = P(B|A) \ P(A)/P(B)$

```
0.5747458500554367
              precision    recall  f1-score   support

           0       0.83      0.50      0.62     15459
           1       0.74      0.42      0.53      6354
           2       0.63      0.77      0.69      8007
           3       0.73      0.26      0.38      8866
           4       0.62      0.46      0.52      5257
           5       0.61      0.92      0.73      4658
           6       0.56      0.72      0.63      4263
           7       0.33      0.87      0.48      3379
           8       0.73      0.68      0.71      3100
           9       0.42      0.57      0.48      2586
          10       0.26      0.85      0.40      2108

    accuracy                           0.57     64037
   macro avg       0.59      0.64      0.56     64037
weighted avg       0.67      0.57      0.57     64037
```

Fig 26: Classification matrix for Naïve Bayes

The Naïve Bayes has given us an accuracy of 57.47% overall for classifying different classes. We can check from the confusion matrix that the model is good in classifying class 0 with the precision of 83%, and it is a little weak in classifying class 7 with a precision of 33%

Decision Tree:

The decision tree is used for regression as well as a classification problem. It is a supervised learning algorithm. Our goal is to develop a decision tree that can develop the class of a given news article. We created the decision tree, fitted the tree on train data, and tested our model on the test data set. We got an acceptable accuracy of 67.05%.

```
0.6705966862907382
              precision    recall  f1-score   support

           0       0.77      0.55      0.64     15459
           1       0.76      0.71      0.74      6354
           2       0.62      0.59      0.61      8007
           3       0.60      0.51      0.55      8866
           4       0.54      0.59      0.56      5257
           5       0.80      0.85      0.83      4658
           6       0.72      0.76      0.74      4263
           7       0.77      0.86      0.82      3379
           8       0.81      0.98      0.89      3100
           9       0.51      0.93      0.66      2586
          10       0.45      0.93      0.60      2108

    accuracy                           0.67     64037
   macro avg       0.67      0.75      0.69     64037
weighted avg       0.69      0.67      0.67     64037
```

Fig 27: Classification matrix for Decision Tree

The Decision Tree classifier has given us an overall accuracy of 67.05% for classifying different classes. We can check from the confusion matrix that the model is good in classifying class 8 with the precision of 81%, and it is a little weak in classifying class 10 with a precision of 45%

Deep Learning Models

Bidirectional LSTM:

Bidirectional Long Short-Term memory is proven to be good to categorize the NLP classification problem. This project has tried to solve the problem statement using a bi-directional LSTM model by using a one-dimensional Convolution network. In the model, we have used ten layers of neurons. The first layer is our embedding layer, where we have embedded each input according to the vocabulary size of 1500. In the next layer, we have used Bidirectional LSTM. After it, we have used a one-dimensional convolution layer along with max-pooling with 128 neurons. After taking the output from the convolution layer, we have used two dense layers with 64 neurons and added a dropout layer by dropping 30% of dead neurons so that we do not overfit the model on train data with the activation relu. Although we tried the activation function as relu and elu as the relu performed well, we choose to move forward with relu. The final output layer consists of 11 neurons as the output function softmax. Softmax will provide us with a probability

of text belonging to each class and classify the text in the class with maximum probability.

We have compiled the model with our optimizer as adam and loss function as sparse_categorical_crossentropy as it worked better with the multiclass classification problem. Below is the snapshot and summary of our model with each layer and the number of neurons.

```python
embedding_dim = 64
model = tf.keras.Sequential([
                            tf.keras.layers.Embedding(voc_size, embedding_dim),
                            tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(embedding_dim, return_sequences=True)),
                            tf.keras.layers.Conv1D(128, 5, activation='relu', input_shape = (None, 128, 1)),
                            tf.keras.layers.GlobalAveragePooling1D(),
                            tf.keras.layers.Dense(64,activation= 'relu'),
                            tf.keras.layers.Dropout(0.3),
                            tf.keras.layers.Dense(embedding_dim,activation= 'relu'),
                            tf.keras.layers.Dropout(0.3),
                            tf.keras.layers.Dense(11, activation= 'softmax')

])


model.summary()
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Model: "sequential"

Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          96000

bidirectional (Bidirectional (None, None, 128)         66048

conv1d (Conv1D)              (None, None, 128)         82048

global_average_pooling1d (Gl (None, 128)               0

dense (Dense)                (None, 64)                8256

dropout (Dropout)            (None, 64)                0

dense_1 (Dense)              (None, 64)                4160

dropout_1 (Dropout)          (None, 64)                0

dense_2 (Dense)              (None, 11)                715
=================================================================
Total params: 257,227
Trainable params: 257,227
Non-trainable params: 0
```

Fig 28: Bidirectional LSTM code snippet

Initially, we tried to train the model for a different 20 number of epochs, but after 8 epochs, the validation accuracy was the same, and train accuracy moved up to 94%. Therefore, to avoid the overfitting of the train data to the model, we restricted the number of epochs to 8, and we got a good accuracy of 85.47% on train data and 75.76% on the test data. Below is the model's output, along with the loss and accuracy for train and validation data.

```
model.fit(X_Under,Y_Under,validation_data=(X_test_lstm,y_test_lstm),epochs=8,batch_size=64)

Epoch 1/8
829/829 [==============================] - 148s 179ms/step - loss: 0.8228 - accuracy: 0.7495 - val_loss: 0.9079 - val_accuracy: 0.7098
Epoch 2/8
829/829 [==============================] - 148s 179ms/step - loss: 0.7461 - accuracy: 0.7789 - val_loss: 0.8563 - val_accuracy: 0.7295
Epoch 3/8
829/829 [==============================] - 149s 179ms/step - loss: 0.6817 - accuracy: 0.8010 - val_loss: 0.8624 - val_accuracy: 0.7261
Epoch 4/8
829/829 [==============================] - 149s 180ms/step - loss: 0.6395 - accuracy: 0.8163 - val_loss: 0.8113 - val_accuracy: 0.7389
Epoch 5/8
829/829 [==============================] - 149s 180ms/step - loss: 0.6024 - accuracy: 0.8260 - val_loss: 0.8021 - val_accuracy: 0.7492
Epoch 6/8
829/829 [==============================] - 148s 179ms/step - loss: 0.5699 - accuracy: 0.8358 - val_loss: 0.8343 - val_accuracy: 0.7439
Epoch 7/8
829/829 [==============================] - 149s 179ms/step - loss: 0.5366 - accuracy: 0.8461 - val_loss: 0.7854 - val_accuracy: 0.7517
Epoch 8/8
829/829 [==============================] - 149s 180ms/step - loss: 0.5138 - accuracy: 0.8547 - val_loss: 0.7892 - val_accuracy: 0.7576
<tensorflow.python.keras.callbacks.History at 0x7ff0181e5c50>
```

Fig 29: Training for Bidirectional LSTM

After building these five models, we extracted data from the New York Times' Twitter account using tweepy and fed the data to our Random Forest model, Multinomial logistic regression model, and Bi-Directional LSTM model. After seeing the performance of each model, we decided to go ahead with Bi-Directional LSTM and deploy the model using the Flask framework. Below is the snapshot of passing the twitter data collected using tweepy.

```
ans = (model.predict(b))
ans

array([ 3,  5,  0,  4,  0,  2,  0,  2,  1,  2,  0,  3,  0,  2,  0,  2,  0,
        8,  2,  0,  2,  0,  2,  0,  0,  0,  2,  0,  0,  2,  2,  2,  0,  2,
        2,  0,  2,  0,  0,  1,  0,  1,  0,  0,  0,  3,  0,  4,  4,  2,  2,
        2,  4,  2,  0,  2,  0,  0,  2,  2,  2,  1,  1,  2,  3,  0,  2,  0,
        0,  0,  0, 11,  2,  2,  2,  0,  2,  0,  2,  0,  2,  2,  2,  0,  1,
        2,  0,  0,  0,  0,  3,  0,  2,  0,  1,  3,  0,  2,  2,  0,  0,  3,
        3,  1,  0,  7,  0,  3,  0,  0,  2,  2,  2,  0,  2,  0,  0,  2,  2,
        2,  1,  5,  2,  0,  0,  0,  4,  0,  0,  0,  0,  1,  0,  0,  2,  0,
        0,  0,  0,  0,  6,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  2,  0,  0,  2,  0,  7,  0,  0,  0,  0,  1,  2,  0,  0,  0,
        2,  1,  0,  3,  0,  0,  4,  3,  0,  0,  3,  0,  0,  3,  0,  0,  0,
        2,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  2,  2,  0,  2,  1,  2,
        2,  2,  2,  0,  4,  0,  0,  0,  2,  0,  0,  1,  0,  2,  0,  0,  1,
        2,  3,  2,  0,  2,  2,  1,  1,  0,  0,  0,  0,  2,  2,  1,  2,  0,
        0,  3])
```

Fig 30: Example out of tweets passed to logistic regression model pickle file

The above snapshot is an example of the output of our model when we passed 240 tweets to the model extracted from tweepy and manually checked which model performs better.

Deployment

<mark>Flask App:</mark>

It is a micro web framework that is written in Python to build Rest API. It is termed as a microframework as it does not require tools or any libraries. It can be used to build complex applications.

The main components required for deployment are:

Pickle file	App.py file	Requirements.txt file	HTML/CSS file

Creating a pickle file comes after the machine learning models are developed and the best model is selected. The Bidirectional LSTM model is chosen for this classification problem to identify the category of articles. The pickle file is then created to store the object. The object here is the classifier that predicts the category of the article.

First, we need to create an instance to use the Flask app. All the requests from the clients are received and passed for handling using WSGI protocol (Web Server Gateway Interface). The requirements file contains all the libraries required for the flask app to run the model. Along with the libraries, the version number must be specified to avoid errors.

We must define a way for the client to send a request. When the client goes to the URL the function associated with it is called at the Flask app to run the specified function. The user can then perform the different functions available on the web page. Every time the user calls a request, the request gets through the Flask app, runs through the function defined and gives the output is displayed on the webpage.

```python
@app.route('/', methods=['POST','GET'])
def index():

    #show the tweetss
    predicted="abc"
    tweet_list = Man_tweet.query.all()
    return render_template('base.html', tweet_list=tweet_list)
```

Fig 31: Route function to display the webpage when URL is called.

When a URL is called, we have designed a base.html file and API_tweets file that will be used to display the webpage and prediction respectively. The model is loaded to the flask app and the text containing the tweet from New York Times Twitter page. The preprocessing steps are specified in the flask app to eliminate the stop words. The model gets the preprocessed text as an input and classifies it to one of the 11 classes. The output is a list containing the value of the class. To display the category name, we have specified a dictionary that will map the value to the category. The Python dictionary is called while returning the prediction along with the confidence score.

The designed User Interface was kept simple, responsive, efficient, and concise. The home page can be split into three functional areas.

1. The common screen header: The New York Times logo was used as a standard screen header for both the base html page and the result html page. Along with the logo, there is also a heading for what the page is responsible for.
2. The tweet entry: There is a small input text box for the tweet text. Along with the text box, there is a button which adds the tweet into the SQLAlchemy. SQLAlchemy is a database toolkit for Python.
3. Stored tweets details: This section consists of all the tweets that were entered into the database by the user. It also includes an option to delete a specific tweet. The section has a button which redirects to the results page, returning the predicted category and confidence of the prediction.

Fig 32: UI for tweets

The results or the prediction page is split into two categories:

1. The common screen header.
2. The result: This section gives the result for the tweet that was selected from the previous page. It contains two variables the main category of the tweet and the confidence of the result.

Fig 33: UI for predicted tweet

The New York Times

For Tweet:

Vaccines haven't been approved for children under 12, but 85% of respondents said it would still likely be safe to gather for Fourth of July. A slightly higher share said schools could fully open in the fall, and that families could gather indoors for the winter holidays

Category: Science & Technology

Confidence: 52.77%

Fig 34: UI for predicted tweet

The New York Times

For Tweet:

Miss Universe Myanmar, Ma Thuzar Wint Lwin, is in Florida for the global pageant this week. After using her platform to protest the military coup in her country, she says she can't return home and doesn't know where she'll go after the competition ends.

Category: Fashion & Style

Confidence: 22.46%

Fig 35: UI for predicted tweet

Fig 36: UI for predicted tweet

Research Questions

RQ1

1. The op-ed section is a platform for authors to communicate personality by storytelling and appealing experiences. The publication is not affiliated with any articles published in this section. Does the op-ed section have these kinds of articles? Or is it just another platform where disputes are bred? Does the op-ed section need more scaling down of data and separated into distinct categories? We will analyze the articles to find out whether the op-ed section answers the above questions.

The above question can be answered by analyzing headlines of the articles in the Op-Ed section. The fields that can help us answer this question are -

- material_type
- snippet

Most Common Words in Op-Ed Section
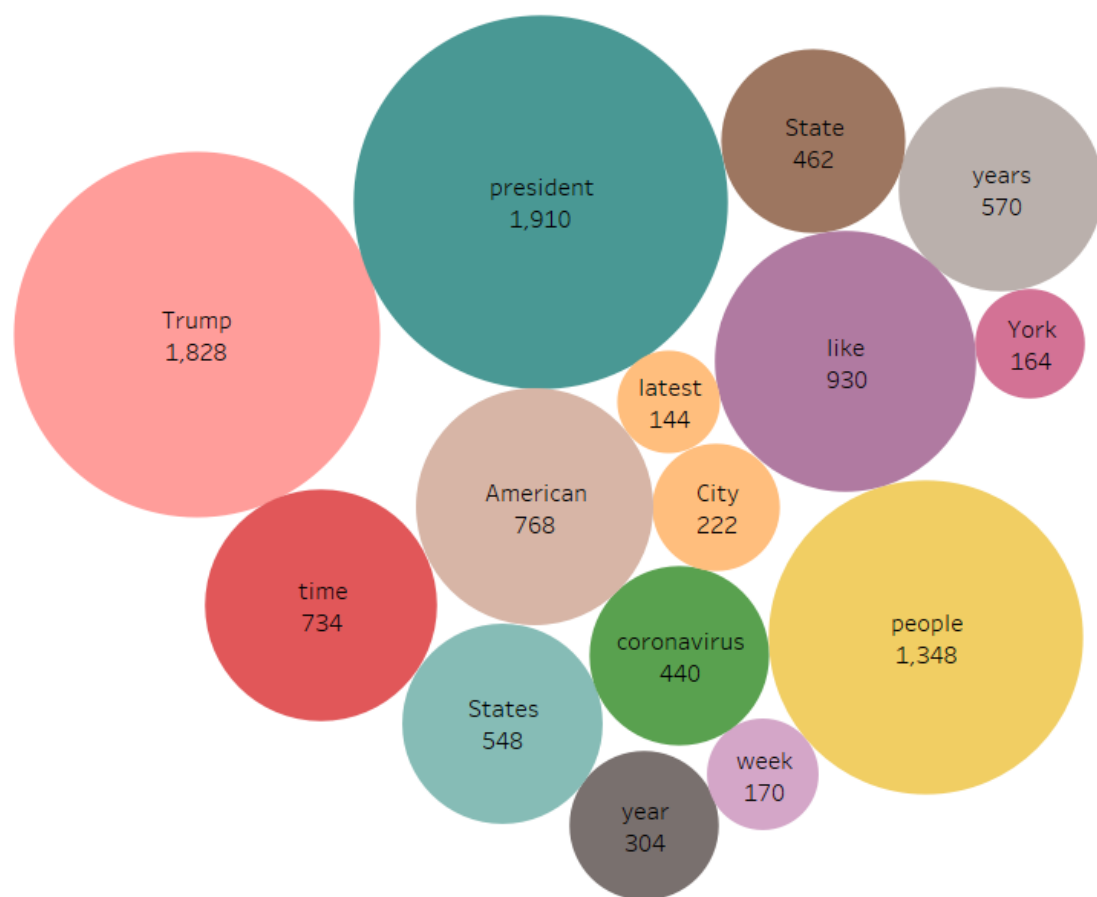
Fig 37: Bubble chart most common words in Op-ed

The above word-cloud shows the most familiar words in the headlines of articles coming under the Op-Ed section.

From the word cloud, we can see that the Op-Ed section is flooded with political articles. Op-ed is the short form for "opposite the editorial page." It does not mean that it is a designated space for "opinions and editorials." Still, it is typically a space for content published by the magazine or newspaper expressing an author's opinion. These authors are not a part of the publisher's editorial board.

For the past few years, this section has been a breeding ground for political disputes. The term 'trump' was in the headline of about 1,828 op-ed articles. Other words like 'Democrats,' 'Biden,' and 'Election' were also seen in most articles. Although many agencies try to break up the opinion section into distinct categories, it is a hassle to seek

the separation between these sections for many people who want to read something new.

Based on analysis, it can make some conclusions and decisions to overcome the issue. Printing and publishing articles under this section that may spark outrage or convey any sign of hatred to the public should be avoid.

Even if the articles are published in the opinions section, it is often not distinctive to the public who disagree with what is written in it. This often leads to criticism towards the publisher itself, even when the author is not affiliated with the publishing board. Therefore, it is necessary to further filter the articles into more distinct categories.

RQ2

2. Will the number of classes affect the accuracy of the model?

Initially, data were collected for four years from N.Y. Time Archive API. The data consisted of 62 news categories. With such a massive number of classes, it was tough to classify the data correctly. In addition, some categories with fewer than 50 news caused the data to be highly imbalanced as some sections had 48000 records. Therefore, we eliminated the classes with less than 700 news articles and lowered the number of classes to 40 and tried to build the classification. However, the accuracy of the model was not good.

As mentioned in the model building section, we tried several combinations with the classes to lower the class by combining multiple classes with similar meaning or dropping the classes with fewer occurrences.

Finally, we build the model with 11 classes of news category as it was most successful in categorizing the news.

As the number of classes increase in the data, the model's accuracy was dropping significantly. Finally, we had to reach an optimal number of classes in which our model will give an acceptable output.

So, by implementing different combinations of the classes for a successfully classifying news section, we concluded that the number of classes plays an essential role in the model accuracy. Below is the snapshot of the classes on which the model was built.

Fig 38: Total number of articles in each section

RQ3

3. How the features(words) and labels (categories of news) correlate with each other? What best keywords can we determine the category of news?

From the word clouds in the Data Visualization section, we can derive a relation between a particular set of words and specific categories. For instance, the Sports section has more instances of words like 'team',' player,' 'season,' and 'world-cup'. This could drastically help in determining which category a particular article belongs to.

These are the most prevailing words in the sections:

1. Opinion – Trump, Government, Politics, United States.
2. Sports – Team, Player, World Cup, Season.
3. Business Day – nCov, Company, Stock, Bonds, Economy.
4. Fashion & Style – Wedding, Society, Apparel, Engagements.
5. U.S. - Politics, President, Trump, United States.
6. Science & Technology – Computers, Internet, social media.
7. Arts – Pop, Rock, Classical Music, Television.
8. Education – Book, Literature, Book Review.
9. Movies & Theater - Leisure, Review, Culture, Arts, Play.
10. Wellness & Health – Coronavirus, Science, Patients, nCov.
11. World – United States, Defense, International Relations, Military Force.

These familiar words in the above categories can be used to determine the types of articles containing these words more effectively. The correlation between words and certain categories can be seen overall in all the categories.

These words can also be further used to gain insights into what really is trending in a specific category during a particular period. For example, in the dataset, we have data ranging from June 2017 to May 2021. We can see that the World section is full of words depicting a war-like situation. Words like 'Defense,' 'International Relations,' 'Military Force' tell us that the World section during this time was full of articles based on bad relations between some set of bodies.

RQ4

4. Is it possible to extract any time series pattern from the data collected from the archive API and help with the categories?

In statistical analyses, time-series data is collected from specific time intervals to analyze the trends and seasonality. The data is analyzed using Python and Tableau to get the required visualizations. Time Series analysis can be instrumental in making future predictions and decisions. Restricting the data to specific intervals will advantage analysts to identify trends and present the findings. Analysts can define their intervals depending on the output they are trying to achieve or the most used type of intervals.

yearly	quarterly	monthly	weekly	daily

In this project, we have the data for five years from 2017 - 2021 to analyze the article published by the New York Times. The number of articles published over the five years can be seen from the below histogram. For each calendar year, we have eleven histograms representing the category of articles.
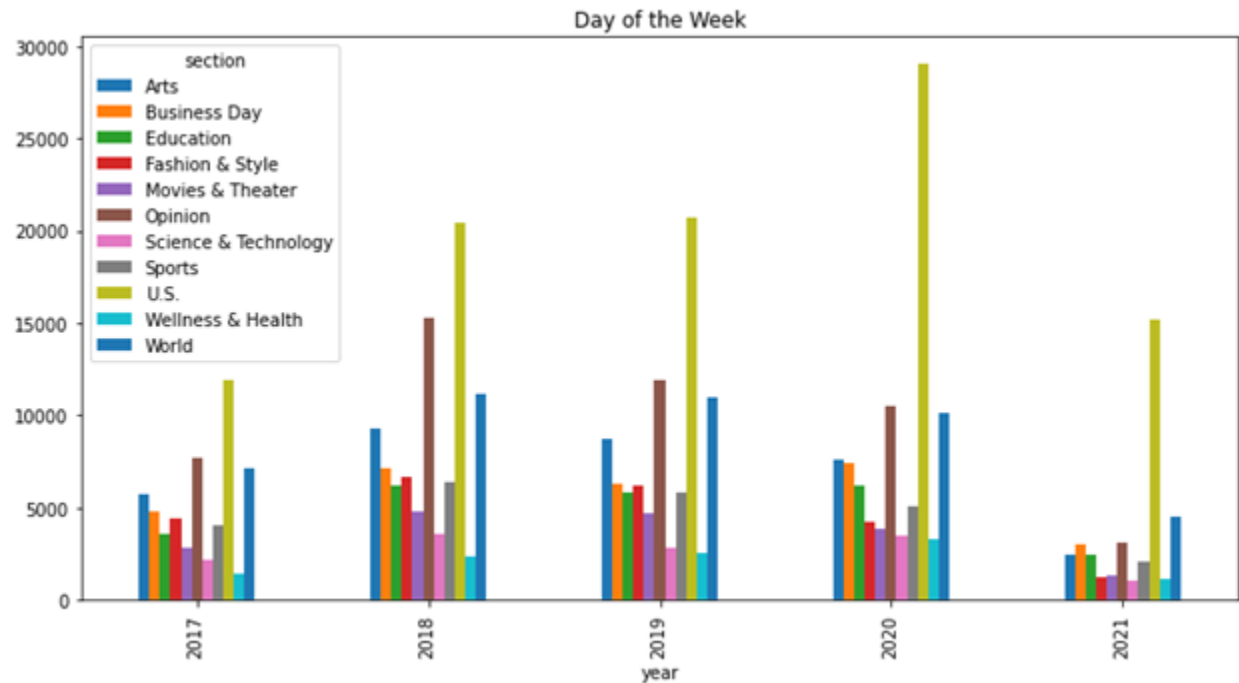
Fig 39: Frequency of articles for eleven sections over five years

We have captured the top five categories to understand the patterns hidden in the data. Arts, World News, Business, Opinion, and U.S. articles were the top 5 categories for all five years. The U.S. news remained on the top for the five years, followed by Opinion articles.

Top 5 categories for New York Times for day of the week:

Arts news articles per weekday

Section / Weekday
Arts
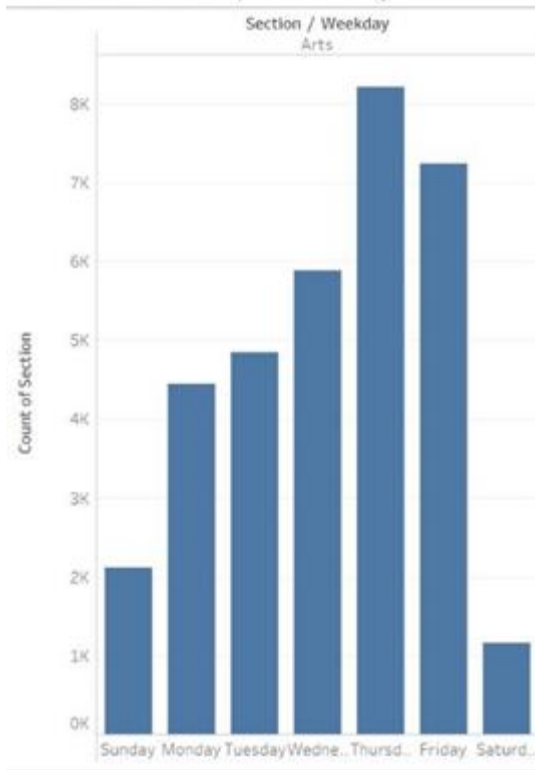
Fig 40: Articles for Arts section

World news articles per weekday

Section / Weekday
World
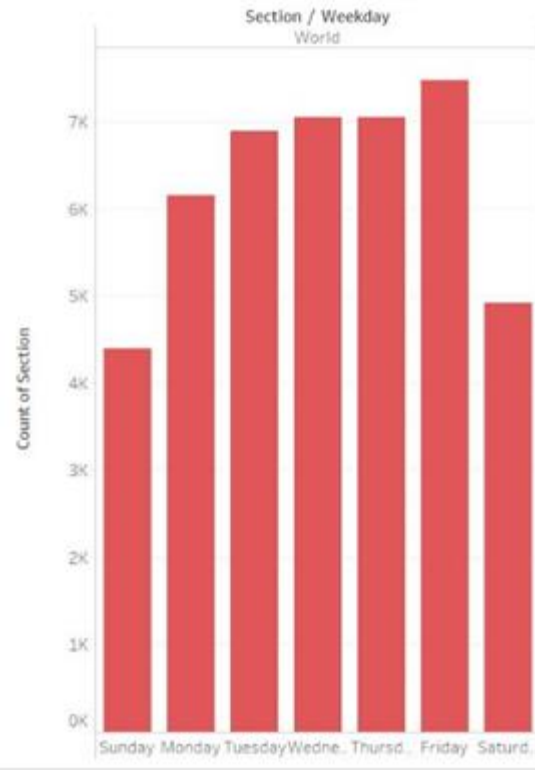
Fig 41: Articles for World News section
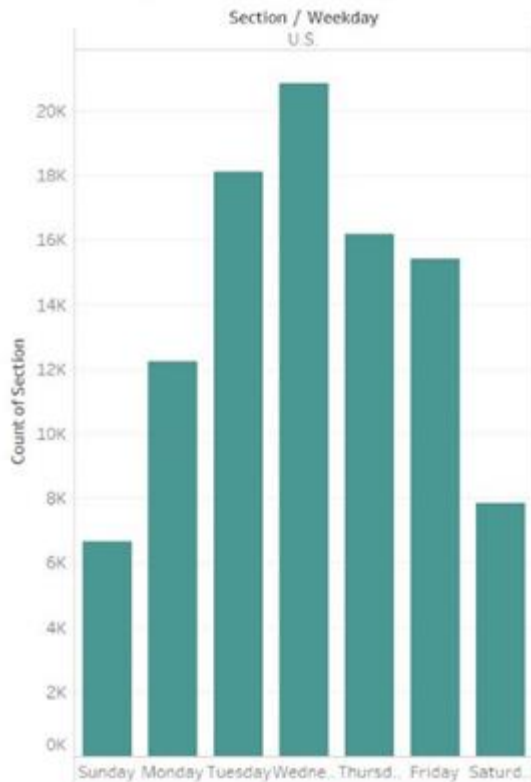
Fig 42: No. of articles for U.S. News section
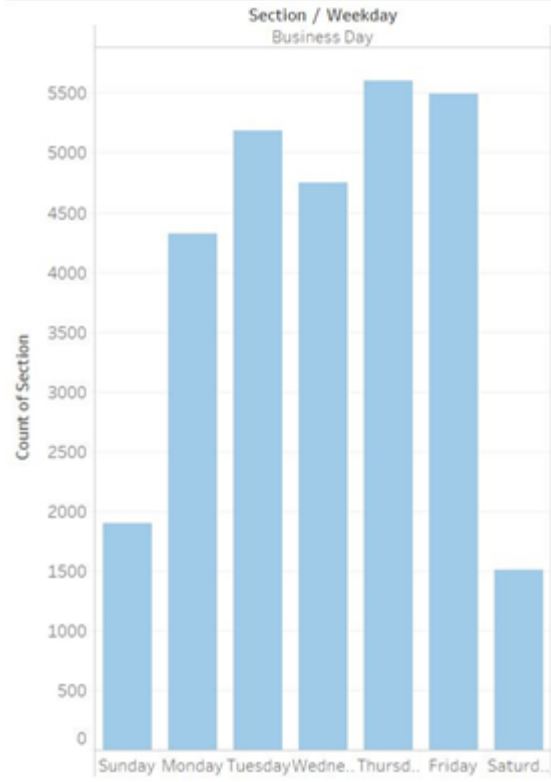
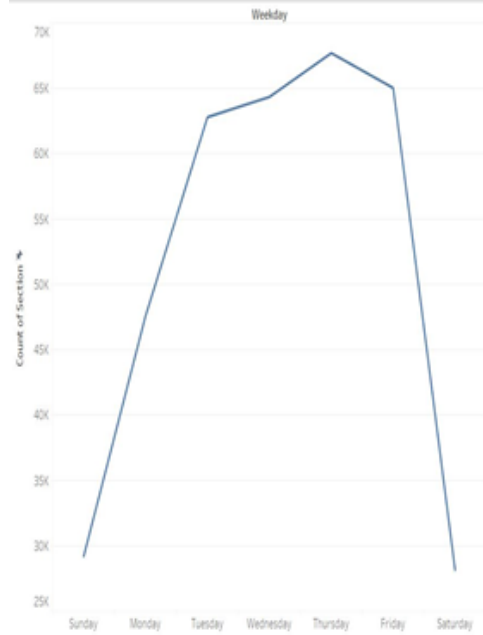

Fig 43: No. of articles for Business section
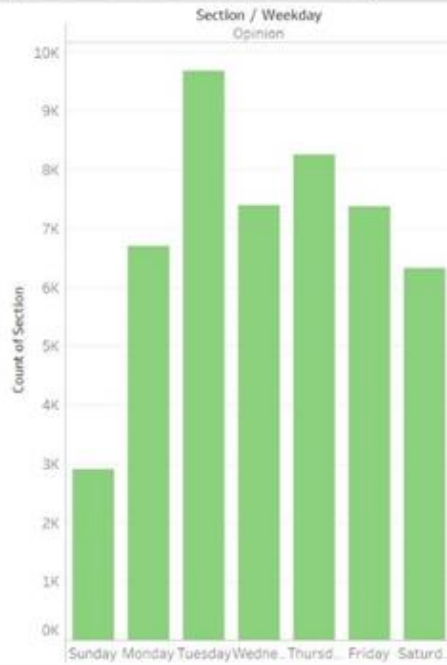
Fig 44: No. of articles for Arts section

Fig 45: No. of articles per day of week

We can observe each category has a specific weekday with most of its articles being published. A general trend was observed like articles on Arts were posted on Wednesday, article on Business was posted on Thursday, Opinion on Tuesday, U.S. on Wednesday, and World's news on Friday. This gives us a piece of evidence that the New York Times primarily focuses on Weekdays rather than the weekend to publish its articles. The reason might be the number of readers actively visiting the website on weekends is less when compared to weekdays. As we discuss New York Times follows a particular pattern while publishing its articles. The articles are published depending on the weekday to give the readers a different content they have read on the previous day.

We have also analyzed the type of articles published over the years. This analysis will help us to identify the category of articles that have gained popularity or vice versa over the years. As we know, The Corona Virus pandemic has affected many people. We tried to find the evidence from the yearly data to understand the impact of coronavirus and the news articles generated about the pandemic.

Top five categories yearly:
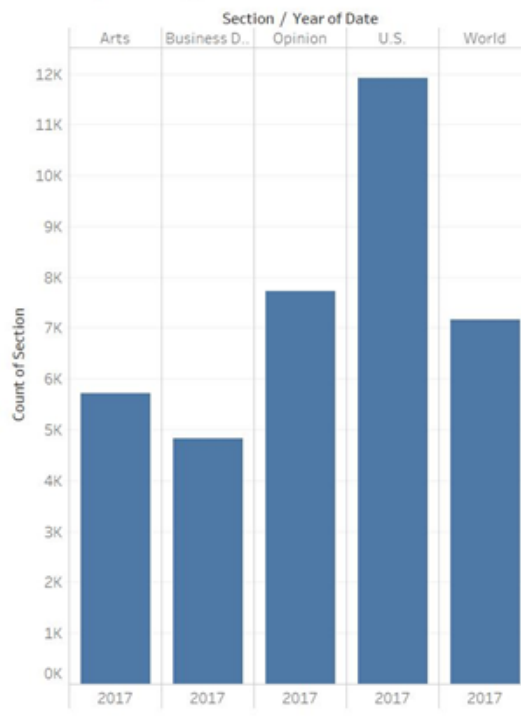
Fig 46: Top five sections frequency in year 2017
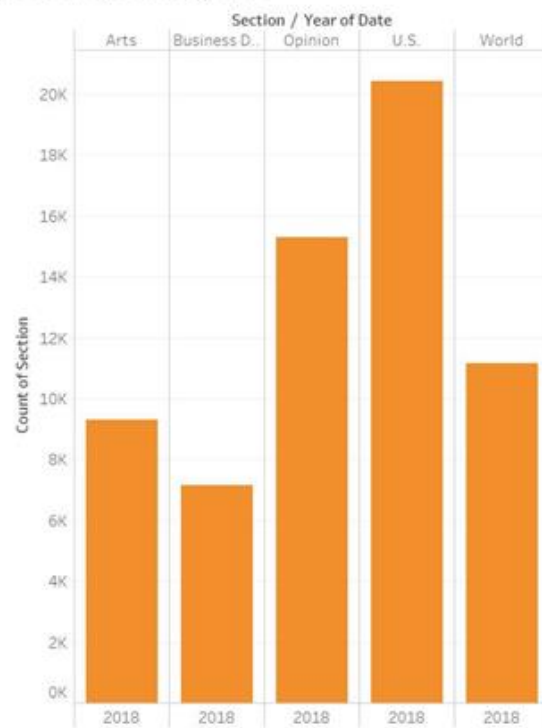
Fig 47: Top five sections frequency in year 2018
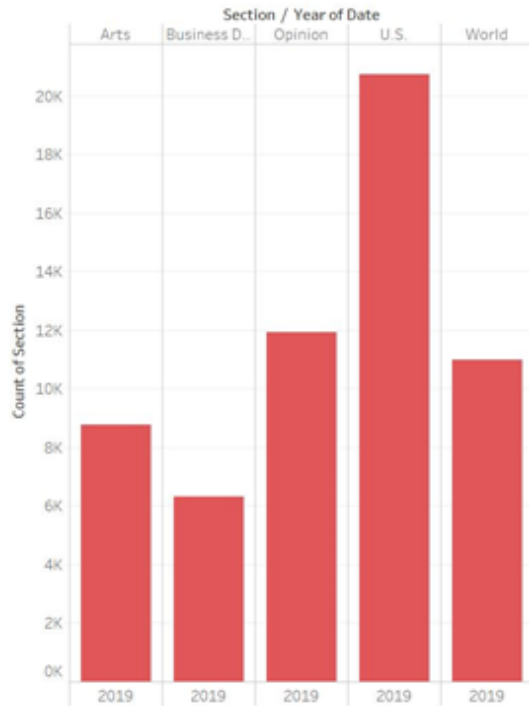
Fig 48: Top five sections frequency in year 2019
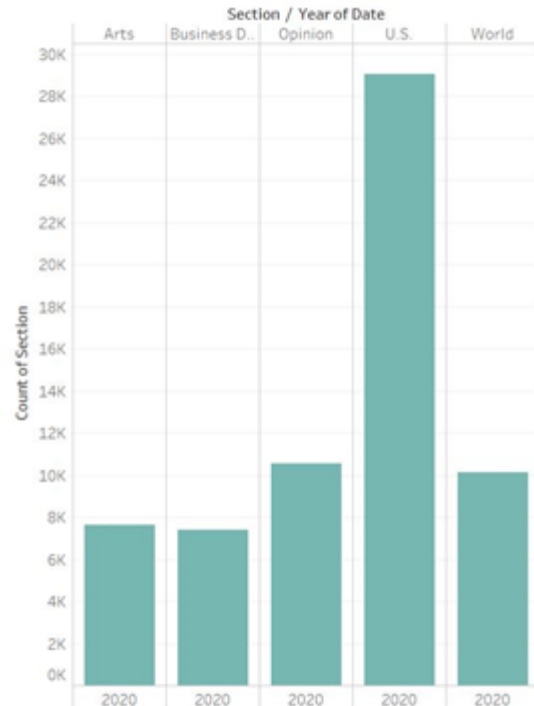

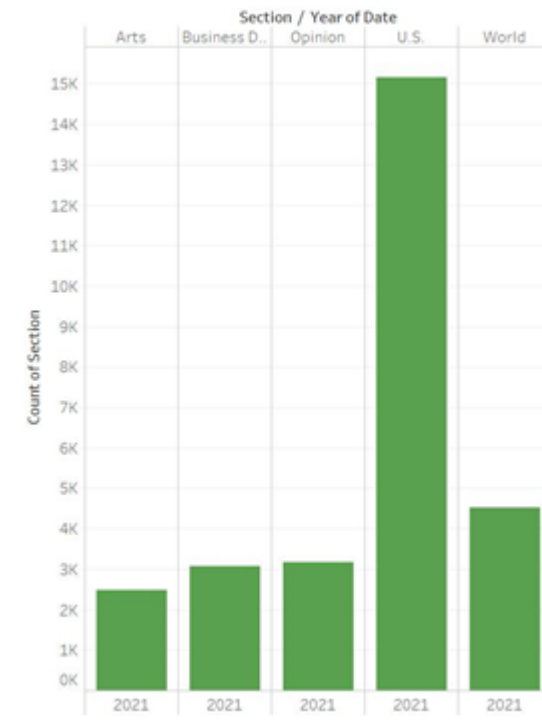Fig 49: Top five sections frequency in year 2020

Fig 50: Top five sections frequency in year 2021

As we discussed, the number of U.S. category articles published is more compared to other categories. We can observe the U.S. category articles for the years 2018 and 2019 are almost similar (>20,000). For the calendar year 2020, we can see the increase in U.S. category articles; 7000 articles were published more than in previous years. This can be evidence to understand the increase in articles explaining the people and businesses affected in the United States due to COVID -19 pandemic.

The opinion and World category articles have plummeted. The reason might be the articles focusing on tackling the COVID-19 pandemic in the United States and the presidential elections held during the year 2020 is also a reason for the increase in articles about things happening inside the United States.

RQ5

5. How well did the model classify? Was the accuracy acceptable?

Confusion Matrix – Confusion matrix is a tabular representation of how well the model performs. The matrix describes the efficiency of the classification model based on the set of data for which the actual values are known. By comparing the proportions, it becomes easy to calculate the accuracy, misclassification rate, precision, etc.

a. Accuracy: Overall, how often is the model correct?
b. Precision: If the model predicts a positive value, how often is it right?
c. Recall: The proportion of actual positive values that were identified correctly
d. F1-Score: The weighted average of recall and precision
e. Support: The number of occurrences of each class in the test data set.
f. Loss: It is the distance between the values in the problem and the values predicted by the model.

```
0.7874197729437669
              precision    recall  f1-score   support

           0       0.88      0.75      0.81     15459
           1       0.86      0.82      0.84      6354
           2       0.81      0.81      0.81      8007
           3       0.78      0.61      0.69      8866
           4       0.72      0.70      0.71      5257
           5       0.91      0.94      0.93      4658
           6       0.83      0.79      0.81      4263
           7       0.84      0.91      0.88      3379
           8       0.83      0.97      0.89      3100
           9       0.50      0.89      0.64      2586
          10       0.46      0.91      0.61      2108

    accuracy                           0.79     64037
   macro avg       0.77      0.83      0.78     64037
weighted avg       0.81      0.79      0.79     64037
```

Fig 51: Precision, Recall, F1-score, and support values for multinomial logistic regression model.

```
loss: 0.5138 - accuracy: 0.8547 - val_loss: 0.7892 - val_accuracy: 0.7576
```

Fig 52: Loss, Accuracy, Validation Loss, and Validation Accuracy for the Bi-directional LSTM Model.

The Multinomial Logistic Regression model and the Bi-directional LSTMs were the most efficient classification models. As can be seen from the above values, both models have an accuracy of over 75%. But the multinomial logistic regression model has better accuracy than all the other classification models. It implies that the model is statistically significant, and it can be accepted.

Conclusion

We successfully built news classifier using Random Forest, Multinomial Logistic Regression, Bi-directional LSTM, Decision Tree, Gaussian Naïve Bayes to categorize the tweet into a specific category. Out of these, Multinomial Logistic Regression and Bi-directional LSTM models were performing comparatively better than the others. A simple, efficient, and precise user interface was developed, which was deployed with the help of Python and Flask. The model was successful in predicting the category of tweets with a certain confidence level.

Following are some findings that can be inferred from the analysis done.

1. The New York Times can focus on the Op-Ed section and filter it into more categories so that the section feels different from the trending topics of politics and the world. They can focus on articles based on Art, Life, or Sports.
2. With the help of visualizations, it was possible to determine the most prevailing words in a particular category. This set of most familiar words in a specific category would help determine the category of an unknown article.
3. The number of categories plays a significant role as far as the accuracy of the model is concerned. The model was tested on about 40 categories, which yielded an extremely low accuracy. On the other hand, when the same model was tested on 11 generalized categories, the accuracy was much higher.
4. The New York Times primarily focuses on weekdays rather than the weekends to publish its articles. The reason behind this to gain more audience attention during weekends. Since a significant recent time, it can also be seen that the World and U.S. articles have plummeted because of reasons like the U.S. elections or the Coronavirus.
5. The classification model was statistically significant. This model tested the data based on all critical categories present in The New York Times and accurately predicted that 75% of the articles fall under the expected class.

References

[1] *Knowledge Base: Creating a Word Cloud.* (2019). https://kb.tableau.com/articles/howto/creating-a-word-cloud

[2] Jianna Park. (2020). *Word Embedding in NLP: One-Hot Encoding and Skip-Gram Neural Network*. https://towardsdatascience.com/word-embedding-in-nlp-one-hot-encoding-and-skip-gram-neural-network-81b424da58f2

[3] Mukesh Chaudhary. (2020). *TF-IDF Vectorizer scikit-learn.* https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a

[4] Hafsa Jabeen. (2018). *Stemming and Lemmatization in Python.* https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

[5] Sahil Rajput. (2018). *Install Flask and create your first web application*. https://dev.to/sahilrajput/install-flask-and-create-your-first-web-application-2dba

[6] *Module: tf.keras*. https://www.tensorflow.org/api_docs/python/tf/keras

[7] Jason Brownlee. (2017). *How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras.* https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[8] *Decision Tree Classifier Documentation*: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[9] *Random Forest Classifier Documentation*: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[10] Rohith Gandhi. (2018). *Naive Bayes Classifier.* https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

Appendix –

Model building code

```
import os

import glob

import json

import time

import requests

import datetime

import dateutil

import pandas as pd

import re

import seaborn as sns

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

import matplotlib.pyplot as plt

from dateutil.relativedelta import relativedelta

import nltk

nltk.download('stopwords')

nltk.download('punkt')

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from sklearn import preprocessing

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.metrics import accuracy_score, classification_report

from google.colab import drive

import tensorflow as tf

from tensorflow.keras.layers import Embedding

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.text import one_hot

from tensorflow.keras.layers import LSTM

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Bidirectional

from tensorflow.keras.layers import Dropout

import nltk

import re

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from keras.models import load_model

nltk.download('stopwords')


drive.mount('/drive')


end = datetime.date.today()

start = end - relativedelta(years=4)

print(end, start)


months_in_range = [x.split(' ') for x in pd.date_range(start, end, freq='MS').strftime("%Y %-m").tolist()]

print(months_in_range)


def send_request(date):
```

```python
    '''Sends a request to the NYT Archive API for given date.'''
    base_url = 'https://api.nytimes.com/svc/archive/v1/'
    url = base_url  + date[0] + '/' + date[1] + '.json?api-key=' + '5bBmcpCW4fOGtnBURGCXLoVFo887iwWX'
    response = requests.get(url).json()
    time.sleep(6)
    return response


def is_valid(article, date):
    '''An article is only worth checking if it is in range, and has a headline.'''
    is_in_range = date > start and date < end
    has_headline = type(article['headline']) == dict and 'main' in article['headline'].keys()
    return is_in_range and has_headline


def parse_response(response):
    '''Parses and returns response as pandas data frame.'''
    data = {'headline': [],
        'date': [],
        'doc_type': [],
        'material_type': [],
        'section': [],
        'news_desk':[],
        'abstract':[],
        'keywords': [],
        'lead_paragraph':[],
        'snippet':[]}
```

```python
articles = response['response']['docs']


for article in articles:
    date = dateutil.parser.parse(article['pub_date']).date()
    if is_valid(article, date):
        data['date'].append(date)
        data['headline'].append(article['headline']['main'])
        if 'section_name' in article:
            data['section'].append(article['section_name'])
        else:
            data['section'].append(None)
        data['doc_type'].append(article['document_type'])
        if 'type_of_material' in article:
            data['material_type'].append(article['type_of_material'])
        else:
            data['material_type'].append(None)
        if 'abstract' in article:
            data['abstract'].append(article['abstract'])
        else:
            data['abstract'].append(None)
        if 'news_desk' in article:
            data['news_desk'].append(article['news_desk'])
        else:
            data['news_desk'].append(None)
        if 'lead_paragraph' in article:
            data['lead_paragraph'].append(article['lead_paragraph'])
        else:
            data['lead_paragraph'].append(None)
        if 'snippet' in article:
```

```python
            data['snippet'].append(article['snippet'])
        else:
            data['snippet'].append(None)
        keywords = [keyword['value'] for keyword in article['keywords'] if keyword['name']
== 'subject']
        new_keywords = listToString(keywords)
        data['keywords'].append(new_keywords)
    return pd.DataFrame(data)


def listToString(s):
    str1 = ""
    for ele in s:
        str1 += (ele+",")
    return str1


def word_cloud(text,color):
    wordcloud = WordCloud(max_font_size=50, max_words=100,
background_color=color).generate(text)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()


def get_data(dates):
    '''Sends and parses request/response to/from NYT Archive API for given dates.'''
    total = 0
    print('Date range: ' + str(dates[0]) + ' to ' + str(dates[-1]))
    if not os.path.exists('headlines'):
        os.mkdir('headlines')
    for date in dates:
```

```python
        response = send_request(date)
        df = parse_response(response)
        total += len(df)
        df.to_csv('headlines/' + date[0] + '-' + date[1] + '.csv', index=False)
        print('Saving headlines/' + date[0] + '-' + date[1] + '.csv...')
    print('Number of articles collected: ' + str(total))


def preprocess_text(text):
    text = text.lower().replace('\n',' ').replace('\r','').strip()
    text = re.sub(' +',' ',text)
    text = re.sub(r'[^\w\s]','',text)
    return text


def tokenize_stopwords(text):
    stop_words = set(stopwords.words('english'))
    word_token = word_tokenize(text)
    filter_sentence = [w for w in word_token if not w in stop_words]
    text = ' '.join(filter_sentence)
    return text


def stemming(messages):
    ps = PorterStemmer()
    corpus = []
    for i in range(0, len(messages)):
        print(i)
        review = re.sub('[^a-zA-Z]', ' ', messages[i])
        review = review.lower()
        review = review.split()
```

```python
        review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
        review = ' '.join(review)
        corpus.append(review)
    return corpus


get_data(months_in_range)


os.chdir("/content/headlines")
extension = 'csv'
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]
combined_csv = pd.concat([pd.read_csv(f) for f in all_filenames ])
combined_csv.to_csv( "combined_csv.csv", index=False, encoding='utf-8-sig')


data = pd.read_csv('/content/headlines/combined_csv.csv')
data.head()


len(data)


sections_abstract_count = data.groupby('section')['abstract'].nunique()
print(sections_abstract_count)
print(len(sections_abstract_count))


data['section'].value_counts()


data['section'].replace(['Well', 'Health'], 'Wellness & Health', inplace = True)
data['section'].replace(['Fashion & Style', 'Style'], 'Fashion & Style', inplace = True)
data['section'].replace('New York', 'U.S.', inplace = True)
data['section'].replace(['Books', 'The Learning Network'], 'Education', inplace = True)
```

```python
data['section'].replace(['Science', 'Technology'], 'Science & Technology', inplace = True)
data['section'].replace(['Movies', 'Theater'], 'Movies & Theater', inplace = True)

# data['section'].replace('U.S', 'U.S.', inplace = True)
# data['section'].replace('T Magazine', 'Magazine', inplace = True)
# data['section'].replace(['Movies', 'Arts', 'Theater'], 'Entertainment', inplace = True)
data['section'].value_counts()

data = data[data.groupby('section').section.transform('count')>=10000].copy()

(data['section'].value_counts())

sections_abstract_count1 = data.groupby('section')['abstract'].nunique()
sections_abstract_count1
len(data['section'].value_counts())

data.isnull().any()

data.isnull().sum()

data.dropna(subset=['headline','material_type', 'keywords',
'snippet','abstract','news_desk','lead_paragraph'],axis=0,inplace=True)
data.isnull().sum()

sns.set(rc={'figure.figsize':(20,15)})
sns.countplot(data.section)
```

```python
data['news_length'] = data['headline'].str.len() + data['abstract'].str.len() +
data['lead_paragraph'].str.len() + data['keywords'].str.len()

data['news_length']

data['news_length'].max()


data['text'] = data['headline'] + " "+ data['abstract'] + " "+ data['lead_paragraph'] +" "+
data['keywords']

data['text'][0]


sns.set()

_ = plt.hist(data['news_length'],bins=70)

_ = plt.xlabel("length")

_ = plt.ylabel("count")

plt.show()


categories = data['section'].unique()

i = 1

for category in categories:

    subset = data[data.section == category]

    sns.set(rc={'figure.figsize':(12,10)})

    text = subset.abstract.values + subset.headline.values +
subset.lead_paragraph.values + subset.keywords.values + subset.news_desk.values

    word = ' '.join(text)

    print('\n' + str(i) + '. ' + category.upper() + '\n')

    if (i % 2 == 0):

        word_cloud(word,'white')

    else:

        word_cloud(word,'black')

    i = i + 1
```

```python
data['parsed_text'] = data['text'].apply(preprocess_text)
data['parsed_text'] = data['parsed_text'].apply(tokenize_stopwords)
data.head()

section_codes = {'U.S.': 0,
'Arts': 1,
'World': 2,
'Opinion': 3,
'Business Day': 4,
'Sports': 5,
'Fashion & Style': 6,
'Movies & Theater': 7,
'Education': 8,
'Science & Technology': 9,
'Wellness & Health': 10,
}

data['section_code'] = data['section']
data = data.replace({'section_code':section_codes})

print(len(data))
y = [0,1,2,3,4,5,6,7,8,9,10,11]
for x in y:
  print(data[data['section_code'] == x])
  x +=1
print(len(data))

lstm_data = data.copy()
```

```python
voc_size = 1500

messages = lstm_data['text'].copy()
messages = messages.reset_index(drop= True)

corpus = stemming(messages)

corpus

totalLne = len(corpus)
i = 0
maxLen = 0
while i < totalLne:
  currentLen = len(corpus[i])
  if currentLen > maxLen:
    maxLen = currentLen
  i += 1
print(maxLen)

onehot_repr=[one_hot(words,voc_size)for words in corpus]
onehot_repr

sent_length=1500
embedded_docs=pad_sequences(onehot_repr,padding='pre',maxlen=sent_length)
print(embedded_docs)

train_label = data['section_code']
len(train_label.value_counts())
```

```python
train_label = train_label.reset_index(drop= True)


print(len(train_label), len(corpus))


embedding_dim = 64
model = tf.keras.Sequential([
                tf.keras.layers.Embedding(voc_size, embedding_dim),
                tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(embedding_dim,
return_sequences=True)),
                tf.keras.layers.Conv1D(128, 5, activation='relu', input_shape = (None,
128, 1)),
                tf.keras.layers.GlobalAveragePooling1D(),
                tf.keras.layers.Dense(64,activation= 'relu'),
                tf.keras.layers.Dropout(0.3),
                tf.keras.layers.Dense(embedding_dim,activation= 'relu'),
                tf.keras.layers.Dropout(0.3),
                tf.keras.layers.Dense(11, activation= 'softmax')


])


model.summary()
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

import numpy as np
X_final=(embedded_docs)
y_final=(train_label)


from sklearn.model_selection import train_test_split
```

```python
X_train_lstm, X_test_lstm, y_train_lstm, y_test_lstm = train_test_split(X_final, y_final,
test_size=0.33, random_state=14)


y_train_lstm.value_counts()


from sklearn.datasets import make_classification

from imblearn.under_sampling import RandomUnderSampler


under = RandomUnderSampler(sampling_strategy={0:5000,1:5000,2:5000,3:5000, 4:
5000, 5: 5000, 6:5000, 7:5000}, random_state=10)

under.fit(X_train_lstm, y_train_lstm)

X_Under, Y_Under = under.fit_sample(X_train_lstm, y_train_lstm)


y_test_lstm.replace('Movies & Theater', 7, inplace= True)

x_test_lstm = np.array(x_test_lstm, dtype=np.float)

y_test_lstm = np.array(y_test_lstm, dtype=np.float)


model.fit(X_Under,Y_Under,validation_data=(X_test_lstm,y_test_lstm),epochs=8,batch
_size=64)


# Commented out IPython magic to ensure Python compatibility.

# %load_ext tensorboard

# %tensorboard --logdir logs


model.save("BidirectionalLSTMLowerClasses.h5")


"""Implementing ML models"""


#splitting data into test & train
```

```python
X_train, X_test, y_train,y_test =
train_test_split(data['parsed_text'],data['section_code'],test_size = 0.2,random_state=8)

print(X_train.shape)
print(y_train.shape)
print(y_test.shape)
print(X_test.shape)

ngram_range = (1,2)
min_df = 10
max_df = 1.
max_features = 5000

tfidf = TfidfVectorizer(encoding='utf-8',
                ngram_range = ngram_range,
                stop_words = None,
                lowercase = False,
                max_df = max_df,
                min_df = min_df,
                max_features = max_features,
                norm = 'l2',
                sublinear_tf = True
                )
train_features = tfidf.fit_transform(X_train).toarray()
train_label = y_train

test_features = tfidf.transform(X_test).toarray()
test_label = y_test
```

```python
from sklearn.datasets import make_classification
from imblearn.under_sampling import RandomUnderSampler


under = RandomUnderSampler(sampling_strategy={0:5000,1:5000,2:5000,3:5000, 4:
5000, 5: 5000, 6:5000, 7:5000}, random_state=10)
#under.fit(train_features, train_label)
train_features, train_label = under.fit_sample(train_features, train_label)


print(test_label.shape)
print(train_label.shape)
print(train_features.shape)
print(test_features.shape)
# train_features.sum()


"""##Random Forest Model"""


model = RandomForestClassifier()
model.fit(train_features,train_label)
predictions = model.predict(test_features)
print(accuracy_score(test_label,predictions))
print(classification_report(test_label,predictions))


print(accuracy_score(test_label,predictions))
print(classification_report(test_label,predictions))


"""##Logistic Regression


"""
```

```python
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
model.fit(train_features,train_label)
predictions = model.predict(test_features)
print(accuracy_score(test_label,predictions))
print(classification_report(test_label,predictions))


"""# Decision Tree"""


from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(train_features,train_label)
predictions = model.predict(test_features)
print(accuracy_score(test_label,predictions))
print(classification_report(test_label,predictions))


"""# Naive Bayes"""


from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(train_features,train_label)
predictions = model.predict(test_features)
print(accuracy_score(test_label,predictions))
print(classification_report(test_label,predictions))
#print(predictions)


from sklearn.model_selection import GridSearchCV
n_estimators = [100,300,500,800]
max_depth = [5,10,15,25]
```

```python
min_samples_split = [2,5,10,15,20]

min_samples_leaf = [1,2,5,10]


hyperF = dict(n_estimators = n_estimators, max_depth =
max_depth,min_samples_split=min_samples_split,min_samples_leaf=min_samples_leaf)

hyperF


hyperF.bes


model = RandomForestClassifier()

model.fit()
```