Classifying Tweets Using NLP and Machine Learning

MODULE TITLE: Machine Learning and Data Analytics

University of Sunderland

AYODEJI MOSES ROTIMI

STUDENT NUMBER: 229783515

Table of Contents

In today's digital age, social media platforms like Twitter have become invaluable sources of real-time information and public sentiment. Twitter is one of the social media platforms where people share lots of opinions, discussions about different key concepts through a common name called tweets. With these vast amounts of data shared across Twitter, much information can be distilled from it, for example, by classifying tweets into different categories. Classification of tweets holds significant implications across various domains. By classifying tweets, businesses, and governments can analyze the sentiments and opinions of people on various topics, events, or entities to gauge public perception. Also, classification allows for filtering tweets based on category. However, one essential thing related to tweets classification is that, due to the vast amount of information, manually classifying this data can be a cumbersome task.

With the advancement in technology, natural language processing and machine learning can be used to distill meaningful information from tweets. For example, it can be used to categorize tweets into different categories. Research has been carried out in the aspect of using NLP for tweet classification, for example, evident in the work of Mondal et al. (2020), where the researchers developed a machine learning algorithm to classify COVID-19 tweets into three different categories. The results of their experimentation showed that the ML model utilized achieved an F1-score of approximately 98.0%.

This research delves into the methodologies and approaches employed in classifying tweets using the NLP pipeline. This research explores various NLP processing steps to convert the raw text data into a format suitable for machine learning algorithms. Additionally, this research experimented with different feature extraction techniques, such as bag-of-words, TF-IDF, and N-gram. Lastly, different machine learning models, including logistic regression, support vector classifier, multinomial naive Bayes, and multilayer perceptron, were experimented with to see the best model.

Through this comprehensive analysis, this research aims to provide insights into the experimentation of using NLP to classify tweets. Then, it provides recommendations on the best-performing NLP pipeline that generates the best result.

Methodology

Business Understanding

The business problem is to develop an NLP pipeline capable of solving a multiclass classification problem. The business problem involves experimenting with different NLP techniques and machine learning models to form a pipeline and eventually determining the pipeline that best performs classifying tweets into various classes.

Data Understanding

The dataset contains 6,443 tweets related to different classes. The dataset comprises 5 features about each tweet, including the tweet text, the date at which tweets were generated, the encoded label on which the tweet is categorized, the tweet's ID and the label name. This dataset was suitable for this task, as it requires building an NLP pipeline to classify tweets into different classes.

Exploratory Data Analysis

Exploratory data analysis of the dataset showed that the label feature of the dataset mainly consists of pop culture, sport, and gaming. i.e., the dataset consists mainly of pop culture tweets, approximately 39%,

and sport and gaming tweets, approximately 36%, as shown in the figure below. Additionally, a word cloud bag-of-words analysis was performed on the tweets to show the words that occur most frequently in the dataset. The figure 1 below shows the word cloud analysis of the tweets.

Data Preparation

Data Cleaning

The dataset was prepared and cleaned to ensure it was ready for the machine-learning model by following some of the steps proposed by Kadhim(2018). Initially, the two features of interest were extracted from the dataset: the text column and the label column. Then, the text column was lowercase so all the text was uniform, and duplicate values could be caught if present. Upon doing that, it was observed that the dataset contained duplicate values. So, these duplicate values were dropped from the dataset. After doing that, emoji characters were removed from the text column. Then, the username and the URL placeholders were removed from the dataset. Next, it was observed that all the text had used inverted commas for cases of contraction. This was deemed inappropriate because stop words will not be caught with this. So, these inverted commas were changed to apostrophes so that they could be removed. Then, stop words were removed from the dataset. Next, it was observed that the text column contained excessive whitespaces between the text. These excessive whitespaces were removed and then converted to a single whitespace. Lastly, trailing and ending whitespaces were removed from the dataset.

Data Preparation for ML

Word Normalization Technique

Two text normalization techniques were experimented with to reduce the number of dimensions in the numerical input vectors passed into the machine learning model and to evaluate how these techniques will improve the model's performance. The two-word normalization techniques that were experimented with include stemming and lemmatization. These techniques utilized include:

Stemming: Stemming is a text processing technique used in natural language processing (NLP) to reduce inflected words to their base or root form, often called the stem. It works by removing prefixes or suffixes from words based on a set of rules. For example, removing "ing" from "running".

Lemmatization: Lemmatization, compared to stemming, is a more sophisticated approach in NLP for reducing inflected words to their base form, also known as the lemma. Unlike stemming's rule-based approach, lemmatization uses morphological analysis to map a word to its base form considering its context and grammatical function.

Feature Extraction -- Text Vectorizer

Since feeding a machine learning model with text is impossible, two feature extraction techniques were experimented with to convert the text into a numerical representation. Feature extraction techniques, including the bag of words (BoW) technique and the term frequency-inverse frequency document (TF-IDF) technique, were experimented with.

Also, the n-gram of each technique was experimented with to evaluate which n-gram range performed best on each feature extraction technique. The n-grams experimented with include unigram, bigram, and unigram and bigram range.

Modelling

Model selection: Four different machine learning models were experimented with as part of the pipeline construction. The machine learning models experimented with include logistic regression, support vector classifier, multi-layer perceptron, and multinomial naïve bayes. The machine learning models were experimented with to see which model performs best in classifying the tweets.

Model evaluation metrics: Six evaluation metrics were used to evaluate the performance of the models, specifically in the model selection phase, i.e., selecting the best-performing model in the model experimentation phase. The primary evaluation metric utilized is the F1 macro-score. Other evaluation metrics used in evaluating the model performance include accuracy, precision, recall, and even more complex evaluation metrics like confusion matrix and ROC curve.

Dataset splitting: The dataset was split into two sets for the modelling aspect, including training and test sets. 80% of the dataset was randomly assigned to the training set, while the remaining 20% was assigned to the test set for evaluating the model performance. Also, the dataset was split based on the proportion of classes in the dataset to account for the imbalance in the dataset.

Deployment

The best NLP pipeline will be deployed as a web app so businesses can interact with it. After deploying the model as a web app, companies will be able to input new tweets, and the web app with the NLP pipeline at the backend will classify the tweets accordingly.

Experimental Design

In this phase, the iterative experimental design employed to assess the performance of the NLP pipeline based on different NLP representations, including text normalization and feature extraction techniques, including machine learning, is outlined.

Initial Experimentation – Baseline NLP pipeline: BoW feature extraction and Logistic Regression

In the initial experimentation, after performing all the necessary data cleaning and processing on the data as discussed in the data cleaning section, the BoW representation of the text was performed and trained on a logistic regression model. This served as the baseline for the experiments.

Integration of Text Normalization Techniques

To improve model performance, the baseline pipeline was further advanced by incorporating two different normalization techniques to reduce the dimensionality of the input vector for the bag-of-words and evaluate their performance on the logistic regression model. The two different normalization techniques compared include stemming and lemmatization.

Evaluation with TF-IDF Feature Extraction

Building upon the baseline established with BoW, the experimentation was repeated, using TF-IDF as feature extraction. Also, the text normalization techniques stated were experimented with by using them to preprocess the dataset before extracting the feature using TF-IDF and evaluating their performance.

Model Comparison and Analysis

Following the BoW and TF-IDF representation experimentation, the four machine learning models were also compared across the NLP representation to see the best model.

Result

Baseline NLP Pipeline (Non-Normalized Dataset + Bag of Words + Logistic Regression)

The baseline NLP pipeline contained: a preprocessed and cleaned dataset, unnormalized data (in terms of text or word normalization) and the features were extracted using a bag of words. Finally, the model was built using logistic regression.

Table 1: classification report of base NLP pipeline built upon bag of words and logistic regression.

	Precision	Recall	F1-Score
0	0.4286	0.1034	0.1667
1	0.6429	0.3158	0.4235
2	0.7576	0.8946	0.8204
3	0.6901	0.5537	0.6144
4	0.8508	0.8843	0.8672
5	0.7857	0.5077	0.6168
Accuracy Score of Model: 78.1%			

NLP Pipeline (Stemmed Dataset + Bag of Words + Logistic Regression)

Building on top the baseline pipeline. The text column was processed using stemming in order to reduce the magnitude of dimensionality

Table 2: classification report of NLP pipeline built upon stemmed dataset + bag of words and logistic regression.

	Precision	Recall	F1-Score

0	0.5556	0.1734	0.2632
1	0.6111	0.3860	0.4731
2	0.7598	0.8867	0.8183
3	0.6549	0.5254	0.5831
4	0.8621	0.8734	0.8677
5	0.7255	0.5692	0.6379
Accuracy score of Model: 77.8%			

NLP Pipeline (Lemmatized Dataset + Bag of Words + Logistic Regression)

Using Lemmatization for Reducing Text Dimensionality

Table 3: classification report of NLP pipeline built upon lemmatized dataset + bag of words and logistic regression.

	Precision	Recall	F1-Score
0	0.6000	0.2069	0.3077
1	0.6667	0.3860	0.4889
2	0.7575	0.9006	0.8229
3	0.7206	0.5537	0.6262
4	0.8602	0.8734	0.8667
5	0.7660	0.5538	0.6429
Accuracy Score of Model: 78.7%			

After reducing the dimensionality of text. Different feature extraction was experimented with based on the different text normalization i.e. Stemming and Lemmatization and including when text was not normalized.

NLP Pipeline (Non-Normalized Dataset + TF-IDF + Logistic Regression)

Table 4: classification report of NLP pipeline built upon non-normalized dataset + TF-IDF and logistic regression.

	Precision	Recall	F1-Score
0	0.0000	0.0000	0.0000
1	1.0000	0.0877	0.1613
2	0.7030	0.9225	0.7979
3	0.7556	0.3842	0.5094
4	0.8110	0.8996	0.8530
5	0.9231	0.3692	0.5275
Accuracy Score of Model: 75.5%			

NLP Pipeline (Stemmed Dataset + TF-IDF + Logistic Regression)

Stemming + TF-IDF

Table 5: classification report of NLP pipeline built upon stemmed dataset + TF-IDF and logistic regression.

	Precision	Recall	F1-Score
0	0.0000	0.0000	0.0000

1	0.9091	0.1754	0.2941
2	0.7246	0.9205	0.8109
3	0.7143	0.4237	0.5319
4	0.8103	0.8952	0.8506
5	0.8929	0.3846	0.5376
Accuracy Score of Model: 76.3%			

NLP Pipeline (Lemmatized Dataset + TF-IDF + Logistic Regression)

Table 6: classification report of NLP pipeline built upon lemmatized dataset + TF-IDF and logistic regression.

	Precision	Recall	F1-Score
0	0.0000	0.0000	0.0000
1	1.0000	0.1228	0.2188
2	0.7189	0.9205	0.8073
3	0.7143	0.4520	0.5714
4	0.8087	0.8952	0.8497
5	0.9286	0.4000	0.5591
Accuracy Score of Model: 76.5%			

Accuracy result score of Changing N-grams for each Feature extraction technique

NLP Pipeline (Logistic Regression + Diff. Feature Extraction Technique Based on Unigram to Bigram across Diff. Text Normalization Technique)

Accuracy score of utilizing unigram range to bigram on each feature extraction technique across the two-text normalization technique.

Table 7: classification report of NLP pipeline built upon different feature extraction based on bigram across different text normalization.

Text Normalization / Feature Extraction	Stemming	Lemmatization
Count Vectorizer	0.7704	0.7719
TF-IDF Vectorizer	0.7510	0.7548

NLP Pipeline (Logistic Regression + Diff. Feature Extraction Technique Based on Bigram across Diff. Text Normalization Technique)

Accuracy score of bigrams on each feature extraction technique across the two-text normalization technique.

Table 8: classification report of NLP pipeline built upon different feature extraction based on unigram to bigram range across different text normalization.

Text Normalization / Feature Extraction	Stemming	Lemmatization
Count Vectorizer	0.6618	0.6563

TF-IDF Vectorizer	0.6672	0.6687

NLP Pipeline (Lemmatized Dataset + Bag of Words + Diff. ML Model) – F1-Score of Models

F1-Score and Accuracy Score of Different Model Experimentation based on Lemmatization and bag of words.

Table 9: F1-Score and Accuracy Score of Different Model Experimentation based on Lemmatization and bag of words.

Model	F1-Macro Score	Accuracy Score
Logistic Regression	0.6259	0.7874
MLP Classifier	0.6143	0.7859
Multinomial NB	0.5365	0.7773
SVC	0.4491	0.7308

Evaluation and Discussion

Different NLP Pipeline Result Evaluation

Initially, a base pipeline was built to predict the classification of tweets. This base pipeline consists of utilizing the dataset that is cleaned and then extracting features from it using Bag of Words. Finally, it is trained with a machine learning model. Upon experimentation, the base pipeline achieved an accuracy score of 78.1%, as shown in Table 1. Then, further advancements were made to the base pipeline by experimenting with two text normalization techniques, which were integrated into the base pipeline. They formed a preprocessing step before extracting features from the dataset using Bag of Words. Upon experimentation, it was observed that one of the text normalization steps did increase the performance of the base pipeline. Adding stemming text normalization to the base pipeline reduced the accuracy result to 77.8%, as shown in Table 2, while the lemmatization technique increased the performance to 78.7%, as shown in Table 3. After achieving that, the next feature extraction technique, i.e., TF-IDF, was also experimented with for three states of the dataset, i.e., non-normalized, stemmed, and lemmatized. The accuracy score of each state includes 75.5%, 76.3%, and 76.5%, as shown in Tables 4, 5, and 6. After all the experimentation, it was observed that the state of the dataset and the feature extraction that generated the best result is lemmatized and Bag of Words. So, due to the other machine learning model selected was also trained on this state. Upon doing this, as shown in Table 9, it is observed that logistic regression is the best-performing model out of all the other models used. So, the final NLP pipeline that performs best is logistic regression with NLP techniques, including lemmatization and Bag of Words feature extraction.

Classification Report of Best Pipeline

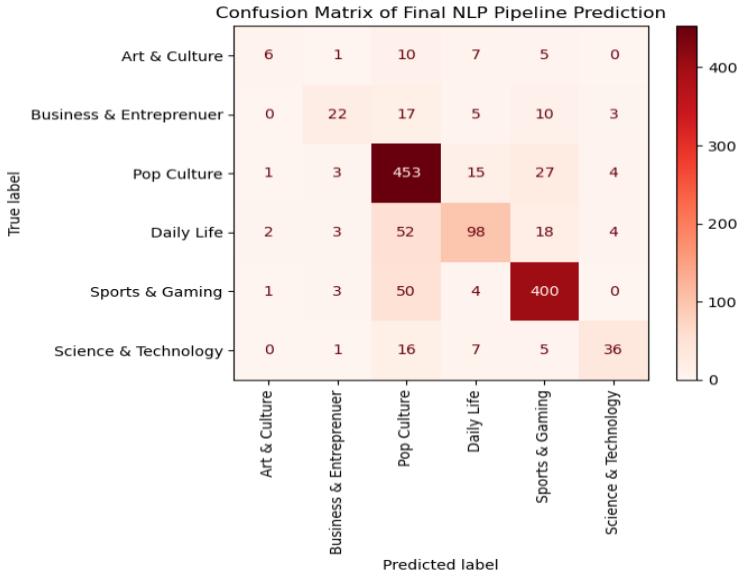
The classification report of the best-performing pipeline shows the nuances in the prediction of the pipeline across all classes. It is seen that the pipeline achieves better accuracy in making decisions and being able to capture instances for the majority classes such as Sports & Gaming, and Pop Culture. While for the minority classes such as Arts & Culture, Daily Life, and Science & Technology, the pipeline achieved

a lower recall score, indicating that the model is not able to capture all instances for these classes accurately.

	Precision	Recall	F1-Score
0 (Arts & Culture)	0.6000	0.2069	0.3077
1 (Business & Entrepreneur)	0.6667	0.3860	0.4889
2 (Pop Culture)	0.7575	0.9006	0.8229
3 (Daily Life)	0.7206	0.5537	0.6262
4 (Sports & Gaming)	0.8602	0.8734	0.8667
5 (Science & Technology)	0.7660	0.5538	0.6429
Accuracy Score of Model: 78.7%			

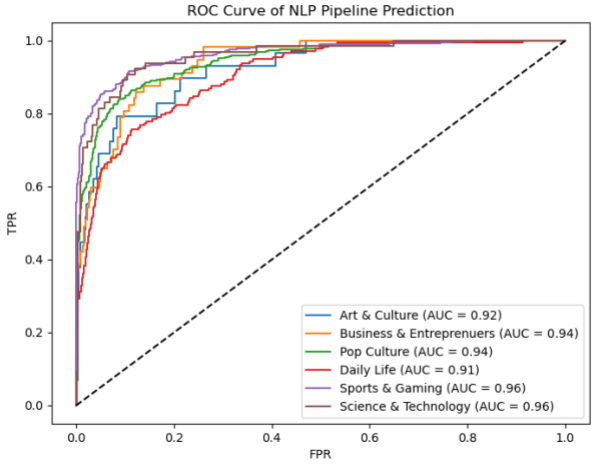
Confusion Matrix of Best NLP Pipeline

The confusion matrix below shows the prediction of the final NLP on how it correctly and incorrectly predicts the instances of each class. It highlights key metrics: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). As shown in the figure below for one of the minority classes, Art and Culture, when evaluating the true positive and false negative for this class, it was seen that the NLP pipeline has a low recall for this class as only 21% of all the class were captured as true positive. The remaining 79% percent of the class tweets were incorrectly classified as other classes (false negative). This demonstrates the pipeline's underperformance in capturing all the instances of this class correctly.



ROC Curve of Best NLP Pipeline

The ROC curve below shows that the pipeline has high discriminative power and can effectively distinguish among the classes. As it can be seen, the area under the curve for each class is all above 0.90. The ROC curve also shows that the pipeline particularly performs well in distinguishing the science & technology and sports & gaming classes from others, as these classes had the highest area under the curve performance.



Effect of Word Normalization on Model Performance

During the baseline pipeline, the logistic regression model was trained solely on the entirety of the dataset's text. Then, the logistic regression model was advanced by introducing two text normalization techniques. Upon doing so, it was observed that normalizing the text actually improved the model's performance compared to just training the model on the non-normalized dataset. Text normalization techniques reduce text vocabulary, thereby ensuring that the model is able to capture more intricate patterns instead of being overwhelmed by numerous noises that might be present in the raw text dataset, which could potentially hinder the model's performance.

However, out of the text normalization techniques used, it was observed that lemmatization performed better than stemming. As argued by Pramana and Anderies (2022), lemmatization offers better results than stemming in tasks such as sentence similarities. Therefore, the experimentation also showed that lemmatization performed better than stemming.

Effect of N-Gram on Feature Extraction Technique

Three different n-grams were experimented with in the two feature extraction techniques utilized. The three n-gram ranges experimented with involved unigram, bigram, and uni-bigram range. Upon experimentation, it was seen that the best-performing n-gram in all the feature extraction techniques is unigram. During experimentation, it was seen that, for example, as shown in Table 3 when logistic regression was used on Unigram Bag of words. It achieved 78.7% while for the uni-bigram range as shown in Table 7, the logistic regression model achieved an accuracy score of 77.2% as shown in Table. However, when the bigram n-gram was used instead, as shown in Table 8, the logistic regression achieved 65.6%. This shows that the machine learning model performs when there is a sense of unigram, i.e., individual words are considered as part of the classification rather than considering other n-gram. As it can be uni-bigram achieved more performance rather than solely relying on bigram, but considering unigram only achieved the best performance.

Performance of Model Experimentation

Four machine learning models were experimented with. The result of the experimentation showed that the best-performing machine learning model is logistic regression with an approximate 62.6% F1-macro score and 78.7% accuracy score, while the most underperforming model is SVC with an F1-score of 44.9% and 73.1% accuracy score. Following closely behind is the multilayer perceptron with an F1-score of 61.4% and 78.6% accuracy.

Performance Comparison of Best NLP Pipeline to Base Pipeline

After all experimentation, the base NLP pipeline achieved an accuracy score of 78.1%, and the best NLP pipeline achieved an accuracy score of 78.7%. This shows that the best NLP pipeline had a 0.77% change increase from the baseline NLP pipeline.

Conclusion and Future Work

This research has achieved its aim of building an NLP pipeline to classify tweets into different categories. Upon experimentation, it was observed that the final NLP pipeline achieving the best results involved a lemmatized dataset and features extracted using the bag-of-words technique. Then, a logistic regression model was trained on this data. While this pipeline has demonstrated accurate measures in classifying tweets, some areas still need to be addressed, i.e., potential future work.

Upon examining the proportion of label classes in the dataset, it was noted that minority classes are present. So future work involving obtaining more tweets related to the minority classes is essential. This would make the dataset more balanced and resolve any issues of imbalance that may exist within the dataset. Additionally, in terms of feature extraction, only two techniques were employed. Future work also involves exploring more advanced feature extraction methods, such as word embedding. Finally, exploring more advanced deep learning models for accurately classifying tweets rather than traditional machine learning models would be beneficial.

References

Kadhim, A.I., 2018. An evaluation of preprocessing techniques for text classification. *International Journal of Computer Science and Information Security (IJCSIS)*, *16*(6), pp.22-32.

Khyani, D., Siddhartha, B.S., Niveditha, N.M. and Divya, B.M., 2021. An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*, *22*(10), pp.350-357.

Mondal, A., Mahata, S., Dey, M. and Das, D., 2021, June. Classification of COVID19 tweets using machine learning approaches. In *Proceedings of the Sixth Social Media Mining for Health (# SMM4H) Workshop and Shared Task* (pp. 135-137).

Pramana, R., Subroto, J.J. and Gunawan, A.A.S., 2022, November. Systematic literature review of stemming and lemmatization performance for sentence similarity. In *2022 IEEE 7th International Conference on Information Technology and Digital Applications (ICITDA)* (pp. 1-6). IEEE.

Zhang, W., Yoshida, T. and Tang, X., 2011. A comparative study of TF* IDF, LSI and multi-words for text classification. *Expert systems with applications*, *38*(3), pp.2758-2765.

Appendices

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import re

from wordcloud import STOPWORDS
import warnings
warnings.filterwarnings("ignore")

# ## Data Understanding

# In[2]:

# read data
twitter_data = pd.read_json("CETM47-23_24-AS2-Data.json")
twitter_data.head()

# In[3]:
```

```
twitter_data.shape

# ### Proportion of Classes

# In[4]:

# frequency of classes
twitter_data["label_name"].value_counts(normalize=True)

# In[5]:

twitter_data["label_name"].value_counts(ascending=True, normalize=True).plot.barh(title="Proportion
of Classes in Dataset")
plt.savefig("label_.png", );

# ### Word Cloud Before Preprocessing

# In[6]:

# import libraries for wordcloud
from wordcloud import WordCloud
from PIL import Image

# In[7]:

import random

# In[8]:

def blue_color_func(word, font_size, position, orientation, random_state=None,**kwargs):
    return "hsl(240, 100%%, %d%%)" % random.randint(20, 40)

# In[9]:

# plot wordcloud
wordcloud_masked = WordCloud(random_state=3, width=800, height=400,
background_color='white').generate(" ".join(twitter_data["text"]))
plt.figure(figsize=(18, 16))
plt.imshow(wordcloud_masked.recolor(color_func=blue_color_func, random_state=42),
           interpolation="bilinear")
plt.axis('off')
plt.savefig("word_cloud.png")
plt.show()

# # Data Cleaning and Preprocessing

# In[10]:
```

```
# extract relevant features
twitter_data = twitter_data[["text", "label"]]

# In[11]:

# normalize text to lower case
twitter_data.loc[:, "text"] = twitter_data["text"].str.lower().values

# In[12]:

# check for duplicates
twitter_data[twitter_data.duplicated()]

# In[13]:

# drop duplicate value
twitter_data.drop_duplicates(inplace=True)

# In[14]:

twitter_data.head()

# In[15]:

#remove emoji and unicode characters.
def remove_special_characters(text):
    pattern = re.compile("["
                         u"\U0001F600-\U0001F64F"
                         u"\U0001F300-\U0001F5FF"
                         u"\U0001F680-\U0001F6FF"
                         u"\U0001F1E0-\U0001F1FF"
                         u"\U00002600-\U000027BF"
                         u"\U0001f300-\U0001f64F"
                         u"\U0001f680-\U0001f6FF"
                         u"\U0001f1e0-\U0001f1ff"
                         u"\U00002500-\U00002BEF"
                         u"\U00002702-\U000027B0"
                         u"\U00002702-\U000027B0"
                         u"\U000024C2-\U0001F251"
                         u"\U0001f926-\U0001f937"
                         u"\U00010000-\U0010ffff"
                         u"\u2640-\u2642"
                         u"\u2600-\u2B55"
                         u"\u200d"
                         u"\u23cf"
                         u"\u23e9"
                         u"\u231a"
                         u"\ufe0f"
```

```
                    u"\u3030"
                    "]+", flags=re.UNICODE)
    return pattern.sub(r'', text)

# remove emoji and other special characters from text
twitter_data["text"] = twitter_data["text"].apply(remove_special_characters)

# In[16]:

twitter_data.head()

# In[17]:

# remove @ url and @ username placeholder

# username
username_pattern = r"\{\@.*?@\}"
twitter_data["text"] = twitter_data["text"].apply(lambda x: re.sub(username_pattern, "", x))

# url pattern removal
url_pattern = r"\{\{.*?\}\}"
twitter_data["text"] = twitter_data["text"].apply(lambda x: re.sub(url_pattern, "", x))

# In[18]:

# replace inverted commas to apostrophe

twitter_data["text"] = twitter_data["text"].apply(lambda x: x.replace("’", "'"))

# In[19]:

twitter_data["text"].loc[3]

# In[20]:

# remove stop words
stop_words = STOPWORDS

" ".join([word for word in re.split(r'[\s\-,\.;!?]+', twitter_data["text"].loc[3]) if word not in stop_words])

# In[21]:

# remove stop words
stop_words = STOPWORDS

twitter_data["text"] = twitter_data["text"].apply(lambda text: " ".join([word for word in re.split(r'[\s\-
,\.;!?]+', text) if word not in stop_words]))
```

```
# In[22]:

twitter_data.head()

# In[23]:

twitter_data["text"].head(10)

# In[24]:

# extract numbers and alphabetical characters
pattern = r"[^a-zA-Z0-9\s]"
twitter_data["text"] = twitter_data["text"].apply(lambda x: re.sub(pattern, "", x))

# In[25]:

twitter_data["text"].loc[3]

# In[26]:

# remove excessive whitespaces
twitter_data["text"] = twitter_data["text"].apply(lambda x:re.sub("\s+"," ", x))

# In[27]:

#remove trailing and ending whitespaces
twitter_data["text"] = twitter_data["text"].apply(lambda x: x.strip())

# In[28]:

twitter_data

# ## Baseline Logistic Regression on raw data
#
# Utilizing Count Vectorizer for Feature Extraction

# In[29]:

# import additional libraries
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import f1_score, accuracy_score, classification_report

# In[30]:

# split dataset into training and test
```

```
X = twitter_data["text"]
y = twitter_data["label"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# train base pipeline
model = Pipeline(
   [("vectorizer", CountVectorizer()),
   ("model", LogisticRegression())]
)

model.fit(X_train, y_train)

# In[31]:

print(classification_report(y_test, model.predict(X_test), digits=4))

# # Moving on

# Working on more text processsing technique
#
# Reducing text number of dimensionality using stemming or lemmatization

# ### Class for Stemming
# Define a class for stemming from nltk so that it can work with Sklearn pipeline

# In[32]:

# import additional libraires
import nltk
from nltk.stem import PorterStemmer
from sklearn.base import BaseEstimator, TransformerMixin

# In[33]:

# NLTK Stemmer Transformer
class NLTKStemmer(BaseEstimator, TransformerMixin):
   def __init__(self):
      self.stemmer = PorterStemmer()
   def fit(self, X, y=None):
      return self
   def transform(self, X):
      stemmed_texts = []
      for text in X:
         tokens = nltk.word_tokenize(text)
         stemmed_text = " ".join([self.stemmer.stem(token) for token in tokens])
         stemmed_texts.append(stemmed_text)
      return stemmed_texts
```

```
# In[34]:

# train pipeline base on stemmer
model = Pipeline(
   [("stemmer", NLTKStemmer()),
    ("vectorizer", CountVectorizer()),
   ("model", LogisticRegression())]
)

model.fit(X_train, y_train)

# In[35]:

print(classification_report(y_test, model.predict(X_test), digits=4))

# ## Class for Lemmatization
# Define a class for lemmatization from nltk so that it can work with Sklearn pipeline

# In[36]:

from nltk.stem import WordNetLemmatizer

# In[37]:

# NLTK Lemmatizer
class NLTKLemmatization(BaseEstimator, TransformerMixin):
   def __init__(self):
      self.lemmatization = WordNetLemmatizer()
   def fit(self, X, y=None):
      return self
   def transform(self, X):
      stemmed_texts = []
      for text in X:
         tokens = nltk.word_tokenize(text)
         stemmed_text = " ".join([self.lemmatization.lemmatize(token) for token in tokens])
         stemmed_texts.append(stemmed_text)
      return stemmed_texts


# In[38]:

# train pipeline base on lemmatization
model = Pipeline(
   [("lemmatization", NLTKLemmatization()),
    ("vectorizer", CountVectorizer()),
   ("model", LogisticRegression())]
```

```
)

model.fit(X_train, y_train)

# In[39]:

print(classification_report(y_test, model.predict(X_test), digits=4))

# Experimentation of stemming and lemmatization for normalizing text before feeding into the base
machine learning model showed that there was an improvement. The result of stemming showed an
improvement from 76% accuracy to 77% accuracy. While for lemmatization is from 76% to 78%.
#
# Next step is to compare different feature extraction technique and not just CountVectorizer to see how
it compare against different text normalization technique.

# ### Moving on -- Different Feature Extraction Technique (TF-IDF)

# ### TF-IDF + Non Normalized Text

# In[40]:

# train pipeline base on TFIDF
model = Pipeline(
   [("vectorizer", TfidfVectorizer()),
   ("model", LogisticRegression())]
)

model.fit(X_train, y_train)

# In[41]:

# classification report
print(classification_report(y_test, model.predict(X_test), digits=4))

# ### TF-IDF  + Stemming

# In[42]:

# train pipeline base on TF-IDF and stemming
model = Pipeline(
   [("stemmer", NLTKStemmer()),
    ("vectorizer", TfidfVectorizer()),
   ("model", LogisticRegression())]
)

model.fit(X_train, y_train)

# In[43]:
```

```
# classification report
print(classification_report(y_test, model.predict(X_test), digits=4))

# ### TF-IDF + Lemmatization

# In[44]:

# TF-IDF and lemmatizatio
model = Pipeline(
   [("stemmer", NLTKLemmatization()),
    ("vectorizer", TfidfVectorizer()),
   ("model", LogisticRegression())]
)

model.fit(X_train, y_train)

# In[45]:

# classification report
print(classification_report(y_test, model.predict(X_test), digits=4))

# Experimenting with different feature extraction technique and text normalization technique showed
that utilizing Count vectorizer feature extraction and using Lemmatization normalization technique
performed best. it achieve an accuracy score of 78% approximately.

# ### Changing Number of Grams for Each Feature Extraction Technique
#
# Considering Both Unigram and Bi-gram

# In[46]:

# accuracy result of n-gram base on uni-bigram range
feature_extraction = [CountVectorizer(ngram_range=(1, 26)), TfidfVectorizer(ngram_range=(1, 2))]
stemming = [NLTKStemmer(), NLTKLemmatization()]

result = []
for extraction in feature_extraction:
   stemming_result = []
   for stemming_technique in stemming:
      model = Pipeline(
         [
            ("stemming", stemming_technique),
            ("extraction", extraction),
            ("model", LogisticRegression())

         ]
         )
```

```
        model.fit(X_train, y_train)
        accuracy_result = accuracy_score(y_test, model.predict(X_test))

        stemming_result.append(accuracy_result)
    result.append(stemming_result)

# In[47]:

result_experimentation = pd.DataFrame(result, index=["Count Vectorizer", "TF-IDF Vectorizer"],
columns=["Stemming", "Lemmatization"])

# In[48]:

result_experimentation

# Considering Only Bi-grams

# In[49]:

# accuracy result of n-gram base on bigram

feature_extraction = [CountVectorizer(ngram_range=(2, 2)), TfidfVectorizer(ngram_range=(2, 2))]
stemming = [NLTKStemmer(), NLTKLemmatization()]

result = []
for extraction in feature_extraction:
    stemming_result = []
    for stemming_technique in stemming:
        model = Pipeline(
            [
                ("stemming", stemming_technique),
                ("extraction", extraction),
                ("model", LogisticRegression())

            ]
            )
        model.fit(X_train, y_train)
        accuracy_result = accuracy_score(y_test, model.predict(X_test))

        stemming_result.append(accuracy_result)
    result.append(stemming_result)

# In[50]:

result_experimentation = pd.DataFrame(result, index=["Count Vectorizer", "TF-IDF Vectorizer"],
columns=["Stemming", "Lemmatization"])

# In[51]:
```

result_experimentation

 # Utiling only bi-grams reduced the peformance of the model significantly.

 # ## Experimenting with Different Models
#
#
 # Feature extraction technique that performed best is CountVectorizer and Lemmatization is best for logistic regression. 3 Models will be experimented with.

 # In[52]:

```
# import machine learning models
from sklearn.naive_bayes import MultinomialNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
```

 # In[53]:

```
# train diff. models base on lemmatization and bag of words
models = [MultinomialNB(), MLPClassifier(), SVC(), LogisticRegression()]
model_names = ["MultinomialNB", "MLPClassifier", "SVC", "LogisticRegression"]
f1_scores = []
accuracy_scores = []
for model in models:
   model_pipeline = Pipeline(
      [
         ("lemmatization", NLTKLemmatization()),
         ("vectorizer", CountVectorizer()),
         ("model", model)
      ]

   )
   model_pipeline.fit(X_train, y_train)

   f1_scores.append(f1_score(y_test, model_pipeline.predict(X_test), average="macro"))
   accuracy_scores.append(accuracy_score(y_test, model_pipeline.predict(X_test)))
```

 # In[54]:

```
pd.DataFrame(data=np.array([f1_scores, accuracy_scores]).T, index=model_names, columns=["F1-Score", "Accuracy Score"])
```

 # Best performing model in this experimentation is logistic regression with an approximate accuracy score of 78.7%.

 # ## Evaluation of Logistic Regression Performance

```
# In[55]:

# logistic regression pipeline
model = Pipeline(
   [("lemmatization", NLTKLemmatization()),
    ("vectorizer", CountVectorizer()),
   ("model", LogisticRegression())]
)

model.fit(X_train, y_train)

# ### Classification Report

# In[56]:

# classifation report
print(classification_report(y_test, model.predict(X_test), digits=4))

# ### Confusion Matrix

# In[57]:

from sklearn.metrics import ConfusionMatrixDisplay, precision_recall_curve, roc_auc_score,
roc_curve,auc

# In[58]:

# confusion matriz
fig, ax = plt.subplots()
ConfusionMatrixDisplay.from_predictions(y_test, model.predict(X_test),
                                display_labels=["Art & Culture", "Business & Entreprenuer",
                                     "Pop Culture", "Daily Life", "Sports & Gaming",
                                     "Science & Technology"],
                               xticks_rotation="vertical", cmap="Reds", ax=ax)
plt.title("Confusion Matrix of Final NLP Pipeline Prediction")
plt.savefig("confusion.png",  bbox_inches="tight");

# ## ROC Curve

# In[61]:

# roc curve
fpr = {}
tpr = {}
roc_auc = {}
for i in range(6):
   fpr[i], tpr[i], threshold = roc_curve((y_test==i), model.predict_proba(X_test)[:, i])
```

```
    roc_auc[i] = auc(fpr[i], tpr[i])

# In[62]:

labels = dict(zip([0, 1, 2, 3, 4, 5], ["Art & Culture", "Business & Entreprenuers", "Pop Culture", "Daily
Life", "Sports & Gaming", "Science & Technology"]))

# In[63]:

plt.figure(figsize=(8, 6))
for i in range(6):
    plt.plot(fpr[i], tpr[i], label = f"{labels[i]} (AUC = {roc_auc[i]:0.2f})")
plt.plot([0,1], [0, 1], 'k--')
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve of NLP Pipeline Prediction")
plt.legend(loc="lower right")
plt.savefig("roc_curve.png")

# In[ ]:
```