Worksheet 8

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

Helper Function for Text Cleaning:

Implement a Helper Function as per Text Preprocessing Notebook and Complete the following pipeline.

```
import re
def remove_urls(text):
  """
  This function will try to remove URL present in our dataset and replace it wit
  Input Args:
  text: strings of text that may contain URLs.
  Output Args:
  text: URLs replaces with text
  """
  url_pattern = re.compile(r'https?://\S+|www\.\S+')
  return url_pattern.sub(r'', text)
```

```
def remove_emoji(string):
  """
  This function will replace the emoji in string with whitespace
  """
  emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
  return emoji_pattern.sub(r' ', string)
```

```
def removeunwanted_characters(document):
  """
  This function will remove all the unwanted characters from the input dataset.
  Input Args:
  document: A text data to be cleaned.
  Return:
  A cleaned document.
  """
  # remove user mentions
  document = re.sub("@[A-Za-z0-9_]+"," ", document)
  # remove hashtags
  document = re.sub("#[A-Za-z0-9_]+","", document)
  # remove punctuation
  document = re.sub("[^0-9A-Za-z ]", "" , document)
  #remove emojis
  document = remove_emoji(document)
  # remove double spaces
  document = document.replace('  ',"")
  return document.strip()
```

```
import nltk
nltk.download('punkt_tab')
from nltk import word_tokenize
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
from nltk.tokenize import RegexpTokenizer

from nltk.tokenize import RegexpTokenizer

def remove_punct(text):
  """
  This function removes the punctutations present in our text data.
  Input Args:
  text: text data.
  Returns:
  text: cleaned text.
  """
  tokenizer = RegexpTokenizer(r"\w+")
  lst=tokenizer.tokenize(' '.join(text))
  return lst
```

```
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
custom_stopwords = ['@', 'RT']
stop_words.update(custom_stopwords)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def remove_stopwords(text_tokens):
  """
  This function removes all the stopwords present in out text tokens.
  Input Args:
  text_tokens: tokenize input of our datasets.
  Returns:
  result_tokens: list of token without stopword.
  """

  result_tokens = []
  for token in text_tokens:
    if token not in stop_words:
       result_tokens.append(token)
  return result_tokens
```

```python
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize,pos_tag
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

def lemmatization(token_text):
  """
  This function performs the lemmatization operations as explained above.
  Input Args:
  token_text: list of tokens.
  Returns:
  lemmatized_tokens: list of lemmatized tokens.
  """
  lemma_tokens = []
  wordnet = WordNetLemmatizer()
  lemmatized_tokens = [wordnet.lemmatize(token, pos = 'v') for token in token_t

  return lemmatized_tokens
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```python
from nltk.stem import PorterStemmer

def stemming(text):
  """
  This function performs stemming operations.
  Input Args:
  token_text: list of tokenize text.
  Returns:
  stemm_tokes: list of stemmed tokens.
  """
  porter = PorterStemmer()
  stemm_tokens = []
  for word in text:
    stemm_tokens.append(porter.stem(word))
  return stemm_tokens
```

```
def lower_order(text):
  """
  This function converts all the text in input text to lower order.
  Input Args:
  token_text : input text.
  Returns:
  small_order_text : text converted to small/lower order.
  """
  small_order_text = text.lower()
  return small_order_text

# Test:
sample_text = "This Is some Normalized TEXT"
sample_small = lower_order(sample_text)
print(sample_small)
```

```
this is some normalized text
```

Build a Text Cleaning Pipeline

```
def text_cleaning_pipeline(dataset, rule = "lemmatize"):
  """
  This function cleans the dataset
  """
  # Convert the input to small/lower order.
  data = lower_order(dataset)
  # Remove URLs
  data = remove_urls(data)
  # Remove emojis
  data = remove_emoji(data)
  # Remove all other unwanted characters.
  data = removeunwanted_characters(data)

  # Create tokens.
  tokens = data.split()
  # Remove stopwords:
  tokens = remove_stopwords(tokens)
  # Stemming or Lemmatization:
  if rule == "lemmatize":
    tokens = lemmatization(tokens)
  elif rule == "stem":
    tokens = stemming(tokens)
  else:
    print("Pick between lemmatize or stem")

  return " ".join(tokens)
```

Text Classification using Machine Learning Models

📝 Instructions: Trump Tweet Sentiment Classification

1. **Load the Dataset**
   Load the dataset named `"trump_tweet_sentiment_analysis.csv"` using `pandas`. Ensure the dataset contains at least two columns: `"text"` and `"label"`.
2. **Text Cleaning and Tokenization**
   Apply a text preprocessing pipeline to the `"text"` column. This should include:
   - Lowercasing the text
   - Removing URLs, mentions, punctuation, and special characters
   - Removing stopwords
   - Tokenization (optional: stemming or lemmatization)
   - "Complete the above function"
3. **Train-Test Split**
   Split the cleaned and tokenized dataset into **training** and **testing** sets using `train_test_split` from `sklearn.model_selection`.
4. **TF-IDF Vectorization**
   Import and use the `TfidfVectorizer` from `sklearn.feature_extraction.text` to transform the training and testing texts into numerical feature vectors.
5. **Model Training and Evaluation**
   Import **Logistic Regression** (or any machine learning model of your choice) from `sklearn.linear_model`. Train it on the TF-IDF-embedded training data, then evaluate it using the test set.
   - Print the **classification report** using `classification_report` from `sklearn.metrics`.

```
import pandas as pd
import numpy as np
```

1. Load the dataset

```
df = pd.read_csv('/content/drive/MyDrive/2025 - 6CS012 - AI and ML - Student/We
df.head()
```

	text	Sentiment
0	RT @JohnLeguizamo: #trump not draining swamp b...	0
1	ICYMI: Hackers Rig FM Radio Stations To Play A...	0
2	Trump protests: LGBTQ rally in New York https:...	1
3	"Hi I'm Piers Morgan. David Beckham is awful b...	0
4	RT @GlennFranco68: Tech Firm Suing BuzzFeed fo...	0

```
df.rename(columns={"Sentiment": "label"}, inplace=True)
```

```
df.columns
```

```
Index(['text', 'label'], dtype='object')
```

2. Text Cleaning and Tokenization

```
cleaned_tokens = df["text"].apply(lambda dataset: text_cleaning_pipeline(datase
```

```
df["clean_text"] = cleaned_tokens
```

```
df['clean_text'][0]
```

```
'rtnot drain swamp taxpayer dollars trip advertise properties'
```

3. Train-Test Split

```
from sklearn.model_selection import train_test_split

X = df['clean_text']
y = df['label']

test_size = 0.2
random_state = 42
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
```

4. TF-IDF Vectorization

```
from sklearn.feature_extraction.text import  TfidfVectorizer
```

```
vectorizer = TfidfVectorizer(max_features=500)

X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train_vec, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test_vec)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.92      0.86    248563
           1       0.77      0.56      0.65    121462

    accuracy                           0.80    370025
   macro avg       0.79      0.74      0.75    370025
weighted avg       0.80      0.80      0.79    370025
```

```
df.shape
```

```
(1850123, 3)
```

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LogisticRegression())
])

param_grid = {
    'tfidf__max_features': [1000, 3000, 5000, 7000, 10000]
}

grid = GridSearchCV(pipeline, param_grid, cv=3, scoring='accuracy', verbose=1)
grid.fit(X_train, y_train)

print("Best max_features:", grid.best_params_)
print("Best accuracy:", grid.best_score_)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:4
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:4
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
  n_iter_i = _check_optimize_result(
Best max_features: {'tfidf__max_features': 10000}
Best accuracy: 0.934375291365842
```