ta-test-1-copy

August 2, 2023

[6]:
```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
```

[6]:
```
True
```

[7]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import RandomOverSampler, SMOTE
```

[2]:
```
#Connect the google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[8]:
```
#Data
newswire = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/newswireFinal.
 ↪csv')
ceylon = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/ceylon_data.csv')
hiru = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/hirunewsFinal.csv')
island = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/island_data.csv')
dailynews = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/dailynews.csv')
lanka = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/lankanewswebFinal.
 ↪csv')
colombogazette = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/
 ↪colomboGazetteFinal.csv')
tamilguardian = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/
 ↪tamilgurdianFinal.csv')
adaderana = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/adaderana.csv')
dailymirror = pd.read_csv('/content/drive/MyDrive/DataScraping/CSV/dailymirror.
 ↪csv')
```

[9]:
```
# Creating Source column
newswire['Source']='newswire'
ceylon['Source']='ceylon'
hiru['Source']='hiru'
island['Source']='island'
dailynews['Source']='dailynews'
lanka['Source']='lanka'
colombogazette['Source']='colombogazette'
tamilguardian['Source']='tamilguardian'
adaderana['Source']='adaderana'
dailymirror['Source']='dailymirror'
```

[10]:
```
# Data Cleaning and Preprocessing
def clean_text(text):
    # Remove markup
    text = re.sub(r'<.*?>', '', str(text))
    # Convert text to lowercase
    text = text.lower()
    return text

# Clean the text data
newswire['content'] = newswire['content'].apply(clean_text)
ceylon['Content'] = ceylon['Content'].apply(clean_text)
hiru['content'] = hiru['content'].apply(clean_text)
island['Content'] = island['Content'].apply(clean_text)
dailynews['content'] = dailynews['content'].apply(clean_text)
lanka['content'] = lanka['content'].apply(clean_text)
```

```
colombogazette['content'] = colombogazette['content'].apply(clean_text)
tamilguardian['content'] = tamilguardian['content'].apply(clean_text)
adaderana['Content'] = adaderana['Content'].apply(clean_text)
dailymirror['content'] = dailymirror['content'].apply(clean_text)
```

[11]:
```
# Tokenization and stop word removal
stop_words = set(stopwords.words('english'))
def tokenize_remove_stopwords(text):
    text = word_tokenize(text)
    text = [token for token in text if token not in stop_words and token.
 ↪isalpha()]
    return text


# Tokenize and remove stopwords from text data
newswire['content'] = newswire['content'].apply(tokenize_remove_stopwords)
ceylon['content'] = ceylon['Content'].apply(tokenize_remove_stopwords)
hiru['content'] = hiru['content'].apply(tokenize_remove_stopwords)
island['content'] = island['Content'].apply(tokenize_remove_stopwords)
dailynews['content'] = dailynews['content'].apply(tokenize_remove_stopwords)
lanka['content'] = lanka['content'].apply(tokenize_remove_stopwords)
colombogazette['content'] = colombogazette['content'].
 ↪apply(tokenize_remove_stopwords)
tamilguardian['content'] = tamilguardian['content'].
 ↪apply(tokenize_remove_stopwords)
adaderana['content'] = adaderana['Content'].apply(tokenize_remove_stopwords)
dailymirror['content'] = dailymirror['content'].apply(tokenize_remove_stopwords)
```

[12]:
```
from collections import Counter

def calculate_token_counts(data):
    # Combine all tokens into a single list
    all_tokens = [token for tokens_list in data['content'] for token in␣
 ↪tokens_list]

    # Calculate the total number of tokens
    total_tokens_count = len(all_tokens)

    # Calculate the unique number of tokens using a set
    unique_tokens_set = set(all_tokens)
    unique_tokens_count = len(unique_tokens_set)

    return total_tokens_count, unique_tokens_count

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(newswire)
```

```
print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(ceylon)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(hiru)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(island)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(dailynews)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(lanka)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(colombogazette)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(tamilguardian)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(adaderana)
```

```
print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(dailymirror)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)
```

```
Total number of tokens: 844029
Unique number of tokens: 34434
Total number of tokens: 17673
Unique number of tokens: 4224
Total number of tokens: 426359
Unique number of tokens: 27166
Total number of tokens: 608694
Unique number of tokens: 30845
Total number of tokens: 358665
Unique number of tokens: 22391
Total number of tokens: 676331
Unique number of tokens: 28494
Total number of tokens: 376758
Unique number of tokens: 20759
Total number of tokens: 180095
Unique number of tokens: 14834
Total number of tokens: 1176890
Unique number of tokens: 38051
Total number of tokens: 980303
Unique number of tokens: 35542
```

[13]:
```
# Function to calculate total and unique token counts
def calculate_token_counts(data):

    # Calculate the total number of tokens
    total_tokens_count = data['content'].apply(len).sum()

    # Calculate the unique number of tokens
    unique_tokens_set = set()
    for tokens_list in data['content']:
        unique_tokens_set.update(tokens_list)

    unique_tokens_count = len(unique_tokens_set)

    return total_tokens_count, unique_tokens_count
```

```
# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(newswire)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(ceylon)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(hiru)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(island)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(dailynews)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(lanka)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(colombogazette)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(tamilguardian)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)
```

```
# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(adaderana)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)

# Call the function with the sample data
total_tokens, unique_tokens = calculate_token_counts(dailymirror)

print("Total number of tokens:", total_tokens)
print("Unique number of tokens:", unique_tokens)
```

```
Total number of tokens: 844029
Unique number of tokens: 34434
Total number of tokens: 17673
Unique number of tokens: 4224
Total number of tokens: 426359
Unique number of tokens: 27166
Total number of tokens: 608694
Unique number of tokens: 30845
Total number of tokens: 358665
Unique number of tokens: 22391
Total number of tokens: 676331
Unique number of tokens: 28494
Total number of tokens: 376758
Unique number of tokens: 20759
Total number of tokens: 180095
Unique number of tokens: 14834
Total number of tokens: 1176890
Unique number of tokens: 38051
Total number of tokens: 980303
Unique number of tokens: 35542
```

[14]:
```
stemmer = PorterStemmer()

def stem_text(text):
  stemmed_words = [stemmer.stem(word) for word in text]
  return " ".join(stemmed_words)

# Apply stemming to text data
newswire['content'] = newswire['content'].apply(stem_text)
ceylon['content'] = ceylon['Content'].apply(stem_text)
hiru['content'] = hiru['content'].apply(stem_text)
island['content'] = island['Content'].apply(stem_text)
dailynews['content'] = dailynews['content'].apply(stem_text)
lanka['content'] = lanka['content'].apply(stem_text)
```

```
colombogazette['content'] = colombogazette['content'].apply(stem_text)
tamilguardian['content'] = tamilguardian['content'].apply(stem_text)
adaderana['content'] = adaderana['Content'].apply(stem_text)
dailymirror['content'] = dailymirror['content'].apply(stem_text)
```

[15]:
```
import nltk
from nltk.stem import SnowballStemmer
from nltk.tokenize import word_tokenize

# Download necessary resources (only needs to be done once)
nltk.download('punkt')
nltk.download('snowball_data')  # Snowball stemmer data

# Initialize the Snowball Stemmer with the desired language
stemmer = SnowballStemmer(language='english')  # Replace 'english' with the␣
 ↪appropriate language if needed

def snowball_stem_text(text):
  stemmed_words = [stemmer.stem(word) for word in text]
  return " ".join(stemmed_words)

# Apply stemming to text data
newswire['content'] = newswire['content'].apply(snowball_stem_text)
ceylon['content'] = ceylon['Content'].apply(snowball_stem_text)
hiru['content'] = hiru['content'].apply(snowball_stem_text)
island['content'] = island['Content'].apply(snowball_stem_text)
dailynews['content'] = dailynews['content'].apply(snowball_stem_text)
lanka['content'] = lanka['content'].apply(snowball_stem_text)
colombogazette['content'] = colombogazette['content'].apply(snowball_stem_text)
tamilguardian['content'] = tamilguardian['content'].apply(snowball_stem_text)
adaderana['content'] = adaderana['Content'].apply(snowball_stem_text)
dailymirror['content'] = dailymirror['content'].apply(snowball_stem_text)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package snowball_data to /root/nltk_data…
```

[16]:
```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# Download WordNet if not already downloaded
nltk.download('wordnet')

# Initialize the WordNet Lemmatizer
lemmatizer = WordNetLemmatizer()
```

```
def lemmatize_text(text):
  lemmatized_words = [lemmatizer.lemmatize(word, pos='v') for word in text]  #␣
 ↪Use 'v' for verbs
  return " ".join(lemmatized_words)

# Apply stemming to text data
newswire['content'] = newswire['content'].apply(lemmatize_text)
ceylon['content'] = ceylon['Content'].apply(lemmatize_text)
hiru['content'] = hiru['content'].apply(lemmatize_text)
island['content'] = island['Content'].apply(lemmatize_text)
dailynews['content'] = dailynews['content'].apply(lemmatize_text)
lanka['content'] = lanka['content'].apply(lemmatize_text)
colombogazette['content'] = colombogazette['content'].apply(lemmatize_text)
tamilguardian['content'] = tamilguardian['content'].apply(lemmatize_text)
adaderana['content'] = adaderana['Content'].apply(lemmatize_text)
dailymirror['content'] = dailymirror['content'].apply(lemmatize_text)
```

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

[17]:
```
# Lemmatization
lemmatizer = WordNetLemmatizer()
def lemmatize_text(text):
  lemmatized_words = [lemmatizer.lemmatize(word) for word in text]
  return " ".join(lemmatized_words)

# Apply lemmatization to text data
newswire['content'] = newswire['content'].apply(lemmatize_text)
ceylon['content'] = ceylon['Content'].apply(lemmatize_text)
hiru['content'] = hiru['content'].apply(lemmatize_text)
island['content'] = island['Content'].apply(lemmatize_text)
dailynews['content'] = dailynews['content'].apply(lemmatize_text)
lanka['content'] = lanka['content'].apply(lemmatize_text)
colombogazette['content'] = colombogazette['content'].apply(lemmatize_text)
tamilguardian['content'] = tamilguardian['content'].apply(lemmatize_text)
adaderana['content'] = adaderana['Content'].apply(lemmatize_text)
dailymirror['content'] = dailymirror['content'].apply(lemmatize_text)
```

[19]:
```
##Method1:Resampling
#Class Weights

from sklearn.utils import resample
from sklearn.utils.class_weight import compute_class_weight

# Assuming you have the DataFrame 'articles_data' with columns 'content' and␣
 ↪'news_agency'
```

```
data_frames = [newswire, ceylon, hiru, island, dailynews , lanka,␣
 ↪colombogazette, tamilguardian,adaderana,dailymirror]
articles_data = pd.concat(data_frames, ignore_index=True)

# Print class distribution before resampling
print("Class Distribution before Resampling:")
print(articles_data['Source'].value_counts())

# Step 1: Count the number of instances per news agency
news_agency_counts = articles_data['Source'].value_counts()

# Step 2: Calculate the class weights for each news agency
class_weights = compute_class_weight(class_weight='balanced',␣
 ↪classes=news_agency_counts.index, y=articles_data['Source'])
class_weights_dict = dict(zip(news_agency_counts.index, class_weights))

# Step 3: Apply resampling to balance the class distribution
balanced_articles_data = pd.DataFrame()
for news_agency in news_agency_counts.index:
    df_subset = articles_data[articles_data['Source'] == news_agency]

    # Choose either oversampling or undersampling based on the class weights
    if class_weights_dict[news_agency] > 1.0:
        # Oversample the minority class
        df_subset_resampled = resample(df_subset, replace=True,␣
 ↪n_samples=news_agency_counts.max(), random_state=42)
    else:
        # Undersample the majority class
        df_subset_resampled = resample(df_subset, replace=False,␣
 ↪n_samples=news_agency_counts.min(), random_state=42)

    balanced_articles_data = pd.concat([balanced_articles_data,␣
 ↪df_subset_resampled])


# Now 'balanced_articles_data' contains the resampled dataset with balanced␣
 ↪class distribution.
# Print class distribution after resampling
# Calculate the class distribution (number of instances per news agency)
class_distribution = balanced_articles_data['Source'].value_counts()

# Print the class distribution
print("Class Distribution:")
print(class_distribution)

# Calculate the percentage of instances for each news agency
```

```
percentage_distribution = class_distribution / len(articles_data) * 100

# Print the percentage distribution
print("\nPercentage Distribution:")
print(percentage_distribution)
```

```
Class Distribution before Resampling:
dailymirror       9029
adaderana         8453
newswire          6894
hiru              4141
island            3997
lanka             3898
colombogazette    2695
dailynews         2500
tamilguardian     1248
ceylon             325
Name: Source, dtype: int64
Class Distribution:
hiru              9029
island            9029
lanka             9029
colombogazette    9029
dailynews         9029
tamilguardian     9029
ceylon            9029
dailymirror        325
adaderana          325
newswire           325
Name: Source, dtype: int64

Percentage Distribution:
hiru              20.910144
island            20.910144
lanka             20.910144
colombogazette    20.910144
dailynews         20.910144
tamilguardian     20.910144
ceylon            20.910144
dailymirror        0.752663
adaderana          0.752663
newswire           0.752663
Name: Source, dtype: float64
```

[15]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import nltk
from nltk.tokenize import word_tokenize
import gensim.downloader as api

# Assuming you have the DataFrame 'balanced_articles_data' with columns␣
 ↪'content' and 'news_agency'

# Step 1: Remove empty documents
balanced_articles_data =␣
 ↪balanced_articles_data[balanced_articles_data['content'].apply(lambda x:␣
 ↪len(x.strip()) > 0)]

# Step 2: Sparse Vector Representation using TF-IDF with custom tokenizer
def custom_tokenizer(text):
    return word_tokenize(text)

tfidf_vectorizer = TfidfVectorizer(tokenizer=custom_tokenizer,␣
 ↪stop_words='english')

# Fit and transform the text data to obtain sparse TF-IDF vectors
sparse_tfidf_vectors = tfidf_vectorizer.
 ↪fit_transform(balanced_articles_data['content'])

# Step 3: Dense Vector Representation using Pre-trained Word Embeddings
word2vec_model = api.load("word2vec-google-news-300")

def get_average_word_embedding(text):
    tokens = word_tokenize(text)
    word_embeddings = [word2vec_model[word] for word in tokens if word in␣
 ↪word2vec_model]
    if not word_embeddings:
        return [0] * 300
    return sum(word_embeddings) / len(word_embeddings)

balanced_articles_data['word_embedding'] = balanced_articles_data['content'].
 ↪apply(get_average_word_embedding)

# Step 4: Split the data into training and testing sets for both representations
X_train_sparse, X_test_sparse, y_train, y_test =␣
 ↪train_test_split(sparse_tfidf_vectors, balanced_articles_data['Source'],␣
 ↪test_size=0.2, random_state=42)
X_train_dense, X_test_dense, _, _ =␣
 ↪train_test_split(list(balanced_articles_data['word_embedding']),␣
 ↪balanced_articles_data['Source'], test_size=0.2, random_state=42)
```

```python
# Step 5: Train classifiers for both representations (e.g., Logistic Regression)
sparse_classifier = LogisticRegression()
sparse_classifier.fit(X_train_sparse, y_train)

dense_classifier = LogisticRegression()
dense_classifier.fit(X_train_dense, y_train)

# Step 6: Make predictions and evaluate the classifiers
y_pred_sparse = sparse_classifier.predict(X_test_sparse)
y_pred_dense = dense_classifier.predict(X_test_dense)

accuracy_sparse = accuracy_score(y_test, y_pred_sparse)
accuracy_dense = accuracy_score(y_test, y_pred_dense)

print("Accuracy using Sparse Vector (TF-IDF):", accuracy_sparse)
print("Accuracy using Dense Vector (Word Embeddings):", accuracy_dense)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:528:
UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is
not None'
  warnings.warn(

[==================================================-] 99.9% 1661.3/1662.8MB
downloaded

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
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-
regression
  n_iter_i = _check_optimize_result(

Accuracy using Sparse Vector (TF-IDF): 0.837542662116041
Accuracy using Dense Vector (Word Embeddings): 0.7868031854379978

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
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-
regression
```

```
  n_iter_i = _check_optimize_result(
```

[16]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪confusion_matrix

# Train classifiers using the sparse vector representation (TF-IDF)
logistic_regression_sparse = LogisticRegression(max_iter=100,solver='saga')
logistic_regression_sparse.fit(X_train_sparse, y_train)

random_forest_sparse = RandomForestClassifier()
random_forest_sparse.fit(X_train_sparse, y_train)

svm_sparse = SVC()
svm_sparse.fit(X_train_sparse, y_train)

# Train classifiers using the dense vector representation (Word2Vec)
logistic_regression_dense = LogisticRegression(max_iter=100,solver='saga')
logistic_regression_dense.fit(X_train_dense, y_train)

random_forest_dense = RandomForestClassifier()
random_forest_dense.fit(X_train_dense, y_train)

svm_dense = SVC()
svm_dense.fit(X_train_dense, y_train)

# Evaluate the classifiers on the test set
accuracy_sparse_logreg = accuracy_score(y_test, logistic_regression_sparse.
 ↪predict(X_test_sparse))
accuracy_sparse_rf = accuracy_score(y_test, random_forest_sparse.
 ↪predict(X_test_sparse))
accuracy_sparse_svm = accuracy_score(y_test, svm_sparse.predict(X_test_sparse))

# Evaluate classifiers on the test set using sparse representation (TF-IDF)
y_pred_logreg_sparse = logistic_regression_sparse.predict(X_test_sparse)
y_pred_rf_sparse = random_forest_sparse.predict(X_test_sparse)
y_pred_svm_sparse = svm_sparse.predict(X_test_sparse)

precision_logreg_sparse = precision_score(y_test, y_pred_logreg_sparse,␣
 ↪average='weighted',zero_division=1)
precision_rf_sparse = precision_score(y_test, y_pred_rf_sparse,␣
 ↪average='weighted',zero_division=1)
precision_svm_sparse = precision_score(y_test, y_pred_svm_sparse,␣
 ↪average='weighted',zero_division=1)
```

```
recall_logreg_sparse = recall_score(y_test, y_pred_logreg_sparse,␣
 ↪average='weighted',zero_division=1)
recall_rf_sparse = recall_score(y_test, y_pred_rf_sparse,␣
 ↪average='weighted',zero_division=1)
recall_svm_sparse = recall_score(y_test, y_pred_svm_sparse,␣
 ↪average='weighted',zero_division=1)

f1_logreg_sparse = f1_score(y_test, y_pred_logreg_sparse,␣
 ↪average='weighted',zero_division=1)
f1_rf_sparse = f1_score(y_test, y_pred_rf_sparse,␣
 ↪average='weighted',zero_division=1)
f1_svm_sparse = f1_score(y_test, y_pred_svm_sparse,␣
 ↪average='weighted',zero_division=1)

confusion_matrix_logreg_sparse = confusion_matrix(y_test, y_pred_logreg_sparse)
confusion_matrix_rf_sparse = confusion_matrix(y_test, y_pred_rf_sparse)
confusion_matrix_svm_sparse = confusion_matrix(y_test, y_pred_svm_sparse)

#Dense
accuracy_dense_logreg = accuracy_score(y_test, logistic_regression_dense.
 ↪predict(X_test_dense))
accuracy_dense_rf = accuracy_score(y_test, random_forest_dense.
 ↪predict(X_test_dense))
accuracy_dense_svm = accuracy_score(y_test, svm_dense.predict(X_test_dense))

y_pred_logreg_dense = logistic_regression_dense.predict(X_test_dense)
y_pred_rf_dense = random_forest_dense.predict(X_test_dense)
y_pred_svm_dense = svm_dense.predict(X_test_dense)

precision_logreg_dense = precision_score(y_test, y_pred_logreg_dense,␣
 ↪average='weighted',zero_division=1)
precision_rf_dense = precision_score(y_test, y_pred_rf_dense,␣
 ↪average='weighted',zero_division=1)
precision_svm_dense = precision_score(y_test, y_pred_svm_dense,␣
 ↪average='weighted',zero_division=1)

recall_logreg_dense = recall_score(y_test, y_pred_logreg_dense,␣
 ↪average='weighted',zero_division=1)
recall_rf_dense = recall_score(y_test, y_pred_rf_dense,␣
 ↪average='weighted',zero_division=1)
recall_svm_dense = recall_score(y_test, y_pred_svm_dense,␣
 ↪average='weighted',zero_division=1)

f1_logreg_dense = f1_score(y_test, y_pred_logreg_dense,␣
 ↪average='weighted',zero_division=1)
```

```
f1_rf_dense = f1_score(y_test, y_pred_rf_dense,␣
 ↪average='weighted',zero_division=1)
f1_svm_dense = f1_score(y_test, y_pred_svm_dense,␣
 ↪average='weighted',zero_division=1)

confusion_matrix_logreg_dense = confusion_matrix(y_test, y_pred_logreg_dense)
confusion_matrix_rf_dense = confusion_matrix(y_test, y_pred_rf_dense)
confusion_matrix_svm_dense = confusion_matrix(y_test, y_pred_svm_dense)

print("Sparse Vector - TF-IDF")
print("Logistic Regression - Accuracy:", accuracy_sparse_logreg)
print("Logistic Regression - Precision:", precision_logreg_sparse)
print("Logistic Regression - Recall:", recall_logreg_sparse)
print("Logistic Regression - F1-Score:", f1_logreg_sparse)
print("Logistic Regression - Confusion Matrix:\n",␣
 ↪confusion_matrix_logreg_sparse)

print("\nRandom Forest - Accuracy:", accuracy_sparse_rf)
print("Random Forest - Precision:", precision_rf_sparse)
print("Random Forest - Recall:", recall_rf_sparse)
print("Random Forest - F1-Score:", f1_rf_sparse)
print("Random Forest - Confusion Matrix:\n", confusion_matrix_rf_sparse)

print("\nSVM - Accuracy:", accuracy_sparse_svm)
print("SVM - Precision:", precision_svm_sparse)
print("SVM - Recall:", recall_svm_sparse)
print("SVM - F1-Score:", f1_svm_sparse)
print("SVM - Confusion Matrix:\n", confusion_matrix_svm_sparse)

print("\nDense Vector - Word2Vec Embeddings")
print("Logistic Regression - Accuracy:", accuracy_dense_logreg)
print("Logistic Regression - Precision:", precision_logreg_dense)
print("Logistic Regression - Recall:", recall_logreg_dense)
print("Logistic Regression - F1-Score:", f1_logreg_dense)
print("Logistic Regression - Confusion Matrix:\n",␣
 ↪confusion_matrix_logreg_dense)

print("\nRandom Forest - Accuracy:", accuracy_dense_rf)
print("Random Forest - Precision:", precision_rf_dense)
print("Random Forest - Recall:", recall_rf_dense)
print("Random Forest - F1-Score:", f1_rf_dense)
print("Random Forest - Confusion Matrix:\n", confusion_matrix_rf_dense)

print("\nSVM - Accuracy:", accuracy_dense_svm)
print("SVM - Precision:", precision_svm_dense)
print("SVM - Recall:", recall_svm_dense)
print("SVM - F1-Score:", f1_svm_dense)
```

```
print("SVM - Confusion Matrix:\n", confusion_matrix_svm_dense)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
Sparse Vector - TF-IDF
Logistic Regression - Accuracy: 0.837542662116041
Logistic Regression - Precision: 0.8472041521126611
Logistic Regression - Recall: 0.837542662116041
Logistic Regression - F1-Score: 0.8153799847731121
Logistic Regression - Confusion Matrix:
 [[1407    0    0    0    0    0    2]
 [   2 1180    0    0    0    0  232]
 [   1   51    0    0    0    0   14]
 [  57    0    0    0    0    0    0]
 [   0   40    0    0    0    0   16]
 [   0   32    0    0    0    0   20]
 [   0  247    0    0    0    0 1094]]

Random Forest - Accuracy: 0.9569965870307168
Random Forest - Precision: 0.9411249907602085
Random Forest - Recall: 0.9569965870307168
Random Forest - F1-Score: 0.9416676193768047
Random Forest - Confusion Matrix:
 [[1409    0    0    0    0    0    0]
 [   0 1405    0    0    1    0    8]
 [   0   35    3    0    5    4   19]
 [  11    0    0   46    0    0    0]
 [   0   32    0    0    1    1   22]
 [   0   20    1    0    1    5   25]
 [   0    4    0    0    0    0 1337]]

SVM - Accuracy: 0.8641638225255973
SVM - Precision: 0.8723420009961659
SVM - Recall: 0.8641638225255973
SVM - F1-Score: 0.8412437852077673
SVM - Confusion Matrix:
 [[1409    0    0    0    0    0    0]
 [   0 1234    0    0    0    0  180]
 [   0   54    0    0    0    0   12]
 [  57    0    0    0    0    0    0]
 [   0   40    0    0    0    0   16]
 [   0   34    0    0    0    0   18]
 [   0  186    0    0    0    0 1155]]

Dense Vector - Word2Vec Embeddings
```

```
Logistic Regression - Accuracy: 0.7899886234357224
Logistic Regression - Precision: 0.801697011733211
Logistic Regression - Recall: 0.7899886234357224
Logistic Regression - F1-Score: 0.7688178689463159
Logistic Regression - Confusion Matrix:
 [[1389    6    0    0    0    0   14]
 [  32 1084    0    0    0    0  298]
 [   2   49    0    0    0    0   15]
 [  55    1    0    0    0    0    1]
 [   0   45    0    0    0    0   11]
 [   0   34    0    0    0    0   18]
 [   6  336    0    0    0    0  999]]

Random Forest - Accuracy: 0.9465301478953356
Random Forest - Precision: 0.9310819579172147
Random Forest - Recall: 0.9465301478953356
Random Forest - F1-Score: 0.9269720646831204
Random Forest - Confusion Matrix:
 [[1409    0    0    0    0    0    0]
 [   0 1400    1    0    1    0   12]
 [   0   63    1    1    0    0    1]
 [  29    8    0   20    0    0    0]
 [   0   54    0    0    0    0    2]
 [   0   49    0    0    0    1    2]
 [   0   12    0    0    0    0 1329]]

SVM - Accuracy: 0.7943117178612059
SVM - Precision: 0.8098657731441882
SVM - Recall: 0.7943117178612059
SVM - F1-Score: 0.7731490113628909
SVM - Confusion Matrix:
 [[1387   14    0    0    0    0    8]
 [  21 1157    0    0    0    0  236]
 [   3   52    0    0    0    0   11]
 [  55    1    0    0    0    0    1]
 [   0   49    0    0    0    0    7]
 [   0   39    0    0    0    0   13]
 [   6  388    0    0    0    0  947]]
```

[17]:
```
pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.31.0-py3-none-any.whl (7.4 MB)
                             7.4/7.4 MB
22.7 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.12.2)
Collecting huggingface-hub<1.0,>=0.14.1 (from transformers)
```

```
  Downloading huggingface_hub-0.16.4-py3-none-any.whl (268 kB)
                             268.8/268.8 kB
30.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (1.22.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2022.10.31)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers) (2.27.1)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers)
  Downloading
tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(7.8 MB)
                             7.8/7.8 MB
62.5 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading
safetensors-0.3.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(1.3 MB)
                             1.3/1.3 MB
63.1 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.10/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from huggingface-hub<1.0,>=0.14.1->transformers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.14.1->transformers) (4.7.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2023.7.22)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers) (3.4)
Installing collected packages: tokenizers, safetensors, huggingface-hub,
transformers
Successfully installed huggingface-hub-0.16.4 safetensors-0.3.1
tokenizers-0.13.3 transformers-4.31.0
```

[18]:
```
import pandas as pd
import numpy as np
import gensim.downloader as api
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional,␣
 ↪Dropout
from transformers import BertTokenizer, BertModel
import torch
import spacy

# Step 2: Encode the target labels
label_encoder = LabelEncoder()
balanced_articles_data['label'] = label_encoder.
 ↪fit_transform(balanced_articles_data['Source'])

# Step 3: Tokenize and pad the text data for deep learning models
max_len = 100  # Maximum sequence length for padding
tokenizer = Tokenizer()
tokenizer.fit_on_texts(balanced_articles_data['content'])
X_sequences = tokenizer.texts_to_sequences(balanced_articles_data['content'])
X_padded = pad_sequences(X_sequences, maxlen=max_len)

# Step 4: Load pre-trained Word2Vec embeddings
word2vec_model = api.load("word2vec-google-news-300")

# Step 5: Load GloVe embeddings
glove_vectors = api.load("glove-wiki-gigaword-300")

# Step 6: Build deep learning models with different architectures

# Create a custom embedding matrix based on the words available in the␣
 ↪word2vec_model
word_index = tokenizer.word_index
embedding_dim = 300
num_words = min(len(word_index) + 1, len(word2vec_model.key_to_index))
embedding_matrix_w2v = np.zeros((num_words, embedding_dim))
for word, i in word_index.items():
    if i >= num_words:
        continue
    if word in word2vec_model:
        embedding_matrix_w2v[i] = word2vec_model[word]

# Create a custom embedding matrix based on the words available in the␣
 ↪glove_vectors
```

```
embedding_matrix_glove = np.zeros((num_words, embedding_dim))
for word, i in word_index.items():
    if i >= num_words:
        continue
    if word in glove_vectors:
        embedding_matrix_glove[i] = glove_vectors[word]
```

```
[==================================================] 100.0% 376.1/376.1MB
downloaded
```

[19]:
```
from keras.regularizers import l2
from keras.callbacks import EarlyStopping

# Model 1: LSTM with Word2Vec Embeddings
model_lstm_w2v = Sequential()
model_lstm_w2v.add(Embedding(input_dim=num_words, output_dim=embedding_dim,␣
 ↪weights=[embedding_matrix_w2v], input_length=max_len, trainable=False))
model_lstm_w2v.add(LSTM(64))
model_lstm_w2v.add(Dropout(0.5))
model_lstm_w2v.add(Dense(32, activation='relu'))
model_lstm_w2v.add(Dropout(0.5))
model_lstm_w2v.add(Dense(len(label_encoder.classes_), activation='softmax'))
model_lstm_w2v.compile(loss='sparse_categorical_crossentropy',␣
 ↪optimizer='adam', metrics=['accuracy'])

# Step 7: Train the deep learning models
model_lstm_w2v.fit(X_padded, balanced_articles_data['label'], epochs=6,␣
 ↪batch_size=32, validation_split=0.2)

# Step 8: Evaluate the models on the test set
y_prob_lstm_w2v = model_lstm_w2v.predict(X_padded)
y_pred_lstm_w2v = np.argmax(y_prob_lstm_w2v, axis=1)
accuracy_lstm_w2v = np.mean(y_pred_lstm_w2v == balanced_articles_data['label'])

precision_lstm_w2v = precision_score(balanced_articles_data['label'],␣
 ↪y_pred_lstm_w2v, average='weighted', zero_division=0)
recall_lstm_w2v = recall_score(balanced_articles_data['label'],␣
 ↪y_pred_lstm_w2v, average='weighted', zero_division=0)
f1_lstm_w2v = f1_score(balanced_articles_data['label'], y_pred_lstm_w2v,␣
 ↪average='weighted', zero_division=0)
confusion_matrix_lstm_w2v = confusion_matrix(balanced_articles_data['label'],␣
 ↪y_pred_lstm_w2v)

print("LSTM with Word2Vec Embeddings Accuracy:", accuracy_lstm_w2v)
print("LSTM with Word2Vec Embeddings Precision:", precision_lstm_w2v)
print("LSTM with Word2Vec Embeddings Recall:", recall_lstm_w2v)
print("LSTM with Word2Vec Embeddings F1-Score:", f1_lstm_w2v)
```

```
print("LSTM with Word2Vec Embeddings Confusion Matrix:\n",␣
 ↪confusion_matrix_lstm_w2v)
```

```
Epoch 1/6
550/550 [==============================] - 23s 20ms/step - loss: 0.9011 -
accuracy: 0.7116 - val_loss: 0.9594 - val_accuracy: 0.6994
Epoch 2/6
550/550 [==============================] - 8s 14ms/step - loss: 0.6376 -
accuracy: 0.8142 - val_loss: 0.3465 - val_accuracy: 0.9251
Epoch 3/6
550/550 [==============================] - 8s 15ms/step - loss: 0.4999 -
accuracy: 0.8619 - val_loss: 0.2747 - val_accuracy: 0.9377
Epoch 4/6
550/550 [==============================] - 7s 13ms/step - loss: 0.4365 -
accuracy: 0.8806 - val_loss: 0.1689 - val_accuracy: 0.9768
Epoch 5/6
550/550 [==============================] - 8s 15ms/step - loss: 0.3909 -
accuracy: 0.8906 - val_loss: 0.1043 - val_accuracy: 0.9916
Epoch 6/6
550/550 [==============================] - 4s 8ms/step - loss: 0.3636 -
accuracy: 0.8968 - val_loss: 0.1265 - val_accuracy: 0.9961
687/687 [==============================] - 3s 3ms/step
LSTM with Word2Vec Embeddings Accuracy: 0.9289973146420282
LSTM with Word2Vec Embeddings Precision: 0.8915221861188684
LSTM with Word2Vec Embeddings Recall: 0.9289973146420282
LSTM with Word2Vec Embeddings F1-Score: 0.9080288135730914
LSTM with Word2Vec Embeddings Confusion Matrix:
 [[6871    0    0    0    0    0   23]
 [  12 6502    0    0    0    4  376]
 [  10   26    0    0    0    0  289]
 [ 293    6    0    0    0    0   26]
 [  10   23    0    0    0    0  292]
 [   5    9    0    0    0  194  106]
 [  13   25    0    0    0   12 6844]]
```

[20]:
```
# Model 2: LSTM with GloVe Embeddings
model_lstm_glove = Sequential()
model_lstm_glove.add(Embedding(input_dim=num_words, output_dim=embedding_dim,␣
 ↪weights=[embedding_matrix_glove], input_length=max_len, trainable=False))
model_lstm_glove.add(LSTM(32))
model_lstm_glove.add(Dropout(0.5))
model_lstm_glove.add(Dense(16, activation='relu'))
model_lstm_glove.add(Dropout(0.5))
model_lstm_glove.add(Dense(len(label_encoder.classes_), activation='softmax'))
model_lstm_glove.compile(loss='sparse_categorical_crossentropy',␣
 ↪optimizer='adam', metrics=['accuracy'])
```

```
model_lstm_glove.fit(X_padded, balanced_articles_data['label'], epochs=6,␣
 ↪batch_size=128, validation_split=0.2)

# Evaluate the LSTM model with GloVe embeddings on the test set
y_prob_lstm_glove = model_lstm_glove.predict(X_padded)
y_pred_lstm_glove = np.argmax(y_prob_lstm_glove, axis=1)
accuracy_lstm_glove = np.mean(y_pred_lstm_glove ==␣
 ↪balanced_articles_data['label'])

# Calculate precision, recall, and F1-score for each class
precision_lstm_glove = precision_score(balanced_articles_data['label'],␣
 ↪y_pred_lstm_glove, average='weighted',zero_division=0)
recall_lstm_glove = recall_score(balanced_articles_data['label'],␣
 ↪y_pred_lstm_glove, average='weighted',zero_division=0)
f1_lstm_glove = f1_score(balanced_articles_data['label'], y_pred_lstm_glove,␣
 ↪average='weighted',zero_division=0)
# Calculate the confusion matrix
confusion_matrix_lstm_glove = confusion_matrix(balanced_articles_data['label'],␣
 ↪y_pred_lstm_glove)

print("LSTM with GloVe Embeddings Accuracy:", accuracy_lstm_glove)
print("LSTM with GloVe Embeddings Precision:", precision_lstm_glove)
print("LSTM with GloVe Embeddings Recall:", recall_lstm_glove)
print("LSTM with GloVe Embeddings F1-Score:", f1_lstm_glove)
print("LSTM with GloVe Embeddings Confusion Matrix:\n",␣
 ↪confusion_matrix_lstm_glove)
```

```
Epoch 1/6
138/138 [==============================] - 5s 12ms/step - loss: 1.1795 -
accuracy: 0.5786 - val_loss: 1.4263 - val_accuracy: 0.0348
Epoch 2/6
138/138 [==============================] - 1s 8ms/step - loss: 0.8543 -
accuracy: 0.7451 - val_loss: 0.7635 - val_accuracy: 0.8066
Epoch 3/6
138/138 [==============================] - 1s 8ms/step - loss: 0.7146 -
accuracy: 0.8026 - val_loss: 0.2682 - val_accuracy: 0.9456
Epoch 4/6
138/138 [==============================] - 1s 8ms/step - loss: 0.6435 -
accuracy: 0.8229 - val_loss: 0.2354 - val_accuracy: 0.9618
Epoch 5/6
138/138 [==============================] - 1s 8ms/step - loss: 0.5958 -
accuracy: 0.8347 - val_loss: 0.3373 - val_accuracy: 0.9413
Epoch 6/6
138/138 [==============================] - 2s 11ms/step - loss: 0.5615 -
accuracy: 0.8400 - val_loss: 0.3653 - val_accuracy: 0.9265
687/687 [==============================] - 3s 4ms/step
LSTM with GloVe Embeddings Accuracy: 0.8767466205452642
```

```
LSTM with GloVe Embeddings Precision: 0.8426968268772437
LSTM with GloVe Embeddings Recall: 0.8767466205452642
LSTM with GloVe Embeddings F1-Score: 0.8531798563693451
LSTM with GloVe Embeddings Confusion Matrix:
 [[6407    0    0    0    0    0  487]
 [   8 5983    0    0    0    0  903]
 [   0    0    0    0    0    0  325]
 [ 237    1    0    0    0    0   87]
 [   0    3    0    0    0    0  322]
 [   0  143    0    0    0    0  171]
 [   9   12    0    0    0    0 6873]]
```

[15]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import tensorflow as tf
from transformers import BertTokenizer, TFBertModel
from tensorflow.keras.layers import Input, LSTM, Dense
from tensorflow.keras.models import Model

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Load the BERT model
bert_model = TFBertModel.from_pretrained('bert-base-uncased')

data_frames = [newswire, ceylon, hiru, island, lanka, colombogazette,␣
 ↪tamilguardian]

# Concatenate the data frames
articles_data = pd.concat(data_frames, ignore_index=True)

# Separate the features (X) and target (Y)
X = articles_data['content'].apply(lambda x: ' '.join(x))
Y = articles_data['Source']

# Tokenize the text data
def tokenize_text(X, tokenizer, max_length):
    input_ids = []
    attention_masks = []

    for text in X:
        # Tokenize the text and add special tokens for BERT
        encoded_dict = tokenizer.encode_plus(
            text,
            add_special_tokens=True,
```

```
            max_length=max_length,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='tf',
            truncation=True
        )

        input_ids.append(encoded_dict['input_ids'])
        attention_masks.append(encoded_dict['attention_mask'])

    return tf.concat(input_ids, axis=0), tf.concat(attention_masks, axis=0)

# Set the maximum sequence length for BERT
max_seq_length = 128

# Tokenize the content data and get input IDs and attention masks
X_input_ids, X_attention_masks = tokenize_text(X, tokenizer, max_seq_length)


# Step 2: Encode the target labels
label_encoder = LabelEncoder()
articles_data['label'] = label_encoder.fit_transform(Y)

def create_lstm_model():
    input_ids = Input(shape=(max_seq_length,), dtype='int32')
    attention_masks = Input(shape=(max_seq_length,), dtype='int32')

    # Get the BERT embeddings for the input IDs
    bert_output = bert_model(input_ids, attention_mask=attention_masks)[0]

    # Use LSTM to process the BERT embeddings
    lstm_output = LSTM(units=64)(bert_output)

    # Add a dense layer for classification
    output = Dense(len(label_encoder.classes_),␣
 ↪activation='softmax')(lstm_output)

    model = Model(inputs=[input_ids, attention_masks], outputs=output)

    return model

# Create the LSTM model
model = create_lstm_model()

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

```
# Train the model
model.fit(
    x=[X_input_ids, X_attention_masks],
    y=articles_data['label'],
    batch_size=32,
    epochs=5,
    validation_split=0.2
)

# After training the model, you can calculate the predicted probabilities for␣
 ↪the test set
y_pred_prob_lstm = model.predict(x=[X_input_ids, X_attention_masks])

# Get the predicted class label (index with the highest probability) for each␣
 ↪sample
y_pred_lstm = np.argmax(y_pred_prob_lstm, axis=1)

# Calculate the evaluation metrics
accuracy_lstm = accuracy_score(articles_data['label'], y_pred_lstm)
print("LSTM Model Accuracy:", accuracy_lstm)
```

```
Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

Downloading (…)okenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

Downloading (…)lve/main/config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

Downloading model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Some weights of the PyTorch model were not used when initializing the TF 2.0
model TFBertModel: ['cls.predictions.transform.dense.bias',
'cls.predictions.transform.LayerNorm.bias', 'cls.seq_relationship.weight',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight']
- This IS expected if you are initializing TFBertModel from a PyTorch model
trained on another task or with another architecture (e.g. initializing a
TFBertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model
that you expect to be exactly identical (e.g. initializing a
TFBertForSequenceClassification model from a BertForSequenceClassification
model).
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use TFBertModel for predictions without further training.

Epoch 1/5

WARNING:tensorflow:Gradients do not exist for variables
['tf_bert_model/bert/pooler/dense/kernel:0',
```

```
'tf_bert_model/bert/pooler/dense/bias:0'] when minimizing the loss. If you're
using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables
['tf_bert_model/bert/pooler/dense/kernel:0',
'tf_bert_model/bert/pooler/dense/bias:0'] when minimizing the loss. If you're
using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables
['tf_bert_model/bert/pooler/dense/kernel:0',
'tf_bert_model/bert/pooler/dense/bias:0'] when minimizing the loss. If you're
using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables
['tf_bert_model/bert/pooler/dense/kernel:0',
'tf_bert_model/bert/pooler/dense/bias:0'] when minimizing the loss. If you're
using `model.compile()`, did you forget to provide a `loss` argument?
```

```
580/580 [==============================] - 608s 956ms/step - loss: 1.4176 -
accuracy: 0.3683 - val_loss: 7.8508 - val_accuracy: 0.0000e+00
Epoch 2/5
580/580 [==============================] - 552s 952ms/step - loss: 1.4102 -
accuracy: 0.3715 - val_loss: 8.7679 - val_accuracy: 0.0000e+00
Epoch 3/5
580/580 [==============================] - 552s 952ms/step - loss: 1.4096 -
accuracy: 0.3715 - val_loss: 9.3325 - val_accuracy: 0.0000e+00
Epoch 4/5
580/580 [==============================] - 553s 953ms/step - loss: 1.4098 -
accuracy: 0.3715 - val_loss: 9.7913 - val_accuracy: 0.0000e+00
Epoch 5/5
580/580 [==============================] - 552s 953ms/step - loss: 1.4097 -
accuracy: 0.3715 - val_loss: 10.1742 - val_accuracy: 0.0000e+00
725/725 [==============================] - 209s 284ms/step
LSTM Model Accuracy: 0.29718079144753856
```

Scraping

[ ]:
```
from bs4 import BeautifulSoup
import requests
import csv
from datetime import datetime,timedelta
import pandas as pd
import pytz
import json
import time
```

[ ]:
```
# Ceylontoday news Scraping
# Create an empty list to store the news data
news_data = []

def scrape_individual_news(news_url):
```

```
    individual_news = requests.get(news_url).text
    soup2 = BeautifulSoup(individual_news, 'lxml')
    dateString = soup2.find('time', class_='entry-date updated␣
 ↪td-module-date')['datetime']
    datetime_obj = datetime.strptime(dateString, '%Y-%m-%dT%H:%M:%S%z')
    news_date = datetime_obj.strftime('%Y-%m-%d')
    news_content_element = soup2.find('div', class_='td_block_wrap␣
 ↪tdb_single_content tdi_108 td-pb-border-top td_block_template_1␣
 ↪td-post-content tagdiv-type')

    if news_content_element:
        news_content = news_content_element.get_text(strip=True)
        return news_date, news_content
    else:
        return None, None

for page in range(1, 351):
    news_website_url = f"https://ceylontoday.lk/category/local/page/{page}/"
    html_text = requests.get(news_website_url).text
    soup = BeautifulSoup(html_text, 'lxml')
    reports = soup.find_all('div', class_='tdb_module_loop td_module_wrap␣
 ↪td-animation-stack')

    for report in reports:
        news_title = report.find('div', class_='td-module-thumb').a['title']
        news_url = report.find('div', class_='td-module-thumb').a['href']
        news_date, news_content = scrape_individual_news(news_url)

        if news_date and news_content:
            news_data.append({
                'Title': news_title,
                'Date': news_date,
                'URL': news_url,
                'Content': news_content
            })
        else:
            print("No content found.")

        # Introduce a short delay between requests to avoid overloading the␣
 ↪server
        time.sleep(1)

# Specify the CSV file path
csv_file_path = 'ceylon_data.csv'

# Write the news data to the CSV file
with open(csv_file_path, 'w', newline='', encoding='utf-8') as csvfile:
```

```
    fieldnames = ['Title', 'Date', 'URL', 'Content']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    # Write the header
    writer.writeheader()

    # Write each news item as a row
    for news_item in news_data:
        writer.writerow(news_item)

print("CSV file has been created successfully.")
```

[ ]:
```
news = []
date_format = "%B %d, %Y"
target_date_str = "July 10, 2022"
target_date = datetime.strptime(target_date_str, date_format)
reached = False

for i in range(1, 300):
    print(f"Page Number: {i}")
    url = f"https://colombogazette.com/category/news/page/{i}/"
    response = requests.get(url).content
    bs = BeautifulSoup(response, 'lxml')

    listOfNews = bs.find('div', class_="td-ss-main-content")
    articles = listOfNews.find_all('div', class_="td-block-row")

    for article in articles:
        newsContainer = article.find_all('div', class_="td-block-span6")
        for data in newsContainer:
            title = data.find('h3').text.strip()
            contentURL = data.find('h3').find('a')['href']
            dateString = data.find('div', class_="td-module-meta-info").
 ↪find('time').text.strip()
            formatedDate = datetime.strptime(dateString, date_format)

            if target_date <= formatedDate:
                print(formatedDate)
                moreInfo = requests.get(contentURL).content
                bs2 = BeautifulSoup(moreInfo, 'lxml')
                contentContainer = bs2.find('div', class_="td-theme-wrap").
 ↪find('article').find('div', class_="td-post-content tagdiv-type").
 ↪find_all('p')
                tempContent = [info.text.strip() for info in contentContainer]
                news.append([title, formatedDate.strftime(date_format), " ".
 ↪join(tempContent)])
                tempContent = []
```

```
            else:
                reached = True
                break

        print(f"Total News as of now: {len(news)}")
        if reached:
            print("Reached the given date limit.")
            break

    if reached:
        break  # No need to continue looping once target date is reached

print("Scraping completed.")

# Now you can process the 'news' list as needed, such as writing it to a CSV␣
 ↪file.
```

[ ]:
```
news = []
date_format = "%A, %d %B %Y - %H:%M"
target_date_str = "Monday, 18 July 2022 - 00:00"
target_date = datetime.strptime(target_date_str, date_format)
reached = False

print(target_date)

for i in range(1, 1500):
    print(f"Page Number: {i}")
    url = f"https://www.hirunews.lk/english/local-news.php?pageID={i}"
    response = requests.get(url).content
    bs = BeautifulSoup(response, 'lxml')

    listOfNews = bs.find('div', class_="trending-section")
    articles = listOfNews.find_all('div', class_="row")

    for article in articles:
        title = article.find('div', class_="column middle").find("a").text
        dateString = article.find('div', class_="middle-tittle-time").text.
 ↪strip()
        formatedDate = datetime.strptime(dateString, date_format)

        if target_date < formatedDate:
            contentURL = article.find('div', class_="column middle").
 ↪find("a")['href']
            moreInfo = requests.get(contentURL).content
            bs2 = BeautifulSoup(moreInfo, 'lxml')
            content = bs2.find('div', class_="main-article-section").
 ↪find('div', id="article-phara2").text.strip()
```

```
            news.append([title, formatedDate.strftime("%B %d, %Y"), content])
        else:
            reached = True
            break

    print(f"Total News as of now: {len(news)}")
    if reached:
        print("Reached the given date limit.")
        break

print("Scraping completed.")

# Now you can process the 'news' list as needed, such as writing it to a CSV␣
 ↪file.
```

[ ]:
```
# Island news Scraping
# Create an empty list to store the news data
news_data = []

for page in range(1, 401):  # Start from page 1
    news_website_url = f"https://island.lk/category/news/page/{page}/"
    html_text = requests.get(news_website_url).text
    soup = BeautifulSoup(html_text, 'lxml')
    reports = soup.find_all('li', class_='mvp-blog-story-wrap left relative␣
 ↪infinite-post')

    for report in reports:
        news_title = report.find('h2').text
        news_url = report.find('a')['href']
        individual_news = requests.get(news_url).text
        soup2 = BeautifulSoup(individual_news, 'lxml')
        news_date = soup2.find('span', class_='mvp-post-date updated').text
        news_content = soup2.find('div', id='mvp-content-main').text.strip()

        # Append the news data as a dictionary to the list
        news_data.append({
            'Title': news_title,
            'Date': news_date,
            'URL': news_url,
            'Content': news_content
        })

    print(f"Scraped page {page}")

# Specify the CSV file path
csv_file_path = 'island_data.csv'
```

```
# Write the news data to the CSV file
with open(csv_file_path, 'w', newline='', encoding='utf-8') as csvfile:
    fieldnames = ['Title', 'Date', 'URL', 'Content']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    # Write the header
    writer.writeheader()

    # Write each news item as a row
    for news_item in news_data:
        writer.writerow(news_item)

print("CSV file has been created successfully.")
```

[ ]:
```
# dailynews Scraping
# Create an empty list to store the news data
news_data = []

for page in range(1, 401):  # Start from page 1
    news_website_url = f"https://dailynews.lk/category/news/page/{page}/"
    html_text = requests.get(news_website_url).text
    soup = BeautifulSoup(html_text, 'lxml')
    reports = soup.find_all('li', class_='mvp-blog-story-wrap left relative␣
 ↪infinite-post')

    for report in reports:
        news_title = report.find('h2').text
        news_url = report.find('a')['href']
        individual_news = requests.get(news_url).text
        soup2 = BeautifulSoup(individual_news, 'lxml')
        news_date = soup2.find('span', class_='mvp-post-date updated').text
        news_content = soup2.find('div', id='mvp-content-main').text.strip()

        # Append the news data as a dictionary to the list
        news_data.append({
            'Title': news_title,
            'Date': news_date,
            'URL': news_url,
            'Content': news_content
        })

    print(f"Scraped page {page}")

# Specify the CSV file path
csv_file_path = 'island_data.csv'

# Write the news data to the CSV file
```

```
with open(csv_file_path, 'w', newline='', encoding='utf-8') as csvfile:
    fieldnames = ['Title', 'Date', 'URL', 'Content']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    # Write the header
    writer.writeheader()

    # Write each news item as a row
    for news_item in news_data:
        writer.writerow(news_item)

print("CSV file has been created successfully.")
```

[ ]:
```
news = []
date_format = "%B %d, %Y"
target_date_str = "July 17, 2022"
target_date = datetime.strptime(target_date_str, date_format)
reached = False

print(target_date)

for i in range(1,800):
  print(f"Page Number: {i}")
  url = f"https://lankanewsweb.net/archives/category/news/page/{i}/"
  response = requests.get(url).content
  bs = BeautifulSoup(response, 'lxml')

  listOfNews = bs.find('div', class_="td_block_wrap tdb_loop tdi_89␣
 ↪tdb-numbered-pagination td_with_ajax_pagination td-pb-border-top␣
 ↪td_block_template_8 tdb-category-loop-posts")
  articles = listOfNews.find_all('div', class_="tdb_module_loop td_module_wrap␣
 ↪td-animation-stack")

  for article in articles:
    title = article.find('div', class_="td-module-meta-info").find('p').text
    contentURL = article.find('div', class_="td-module-meta-info").find('p').
 ↪find('a').attrs['href']

    moreInfo = requests.get(contentURL).content
    bs2 = BeautifulSoup(moreInfo, 'lxml')
    dateString = bs2.find('div', class_="td-theme-wrap").find('div',␣
 ↪class_="td_block_wrap tdb_single_date tdi_97 td-pb-border-top␣
 ↪td_block_template_1 tdb-post-meta").find('div', class_="tdb-block-inner␣
 ↪td-fix-index").find('time').text.strip()
    formatedDate = datetime.strptime(dateString, date_format)
    print(formatedDate)
    if(target_date < formatedDate):
```

```
      content = bs2.find('div', class_="td-theme-wrap").find('div',␣
↪class_="vc_column_inner tdi_95 wpb_column vc_column_container␣
↪tdc-inner-column td-pb-span9").find('div', class_="vc_column-inner").
↪find('div', class_="wpb_wrapper").find('div', class_="td_block_wrap␣
↪tdb_single_content tdi_100 td-pb-border-top td_block_template_1␣
↪td-post-content tagdiv-type").find('div', class_="tdb-block-inner␣
↪td-fix-index").text.strip()
      news.append([title, formatedDate.strftime("%B %d, %Y"), content])
    else:
      reached = True

  print(f"Total News as of now: {len(news)}")
  if(reached):
    print("Reached the give date limit.")
    break
```

[ ]:
```
news = []
date_format = "%Y-%m-%dT%H:%M:%S%z"
target_date_str = "2022-07-18T00:00:00+00:00"
target_date = datetime.strptime(target_date_str, date_format)
reached = False

for i in range(1,400):
  print(f"Page Number: {i}")
  url = f"https://www.newswire.lk/category/news/page/{i}/"
  response = requests.get(url).content
  bs = BeautifulSoup(response, 'lxml')

  listOfNews = bs.find('div', id="content-wrap")
  articles = listOfNews.find_all('article')

  for article in articles:
    container = article.find('div', class_="entry-grid-content")
    time = container.find('div', class_="entry-byline").find('time').
↪attrs['datetime']
    datetime_object = datetime.strptime(time, date_format)
    dateStr = datetime_object.strftime("%B %d, %Y")
    # print(f"Published Date: {dateStr}")

    if(target_date < datetime_object):
      titleTag = container.find('header', class_="entry-header").find('h2').
↪find('a')
      title = titleTag.text
      moreURL = titleTag.attrs['href']
      moreInfo = requests.get(moreURL).content
      bs2 = BeautifulSoup(moreInfo, 'lxml')
```

```
      content = bs2.find('div', class_="content-wrap theiaStickySidebar").
↪find('article').find("div", class_="entry-the-content").text
      news.append([title, dateStr, content])
      # print(f"Total news until page {i} is {len(news)}")
    else:
      reached = True

  print(f"Total News as of now: {len(news)}")
  if(reached):
    print("Reached the give date limit.")
    break
```

[ ]:
```
news = []
date_format = "%d %B %Y"
target_date_str = "01 June 2022"
target_date = datetime.strptime(target_date_str, date_format)
reached = False

print(target_date)

for i in range(0,200):
  print(f"Page Number: {i}")
  url = f"https://www.tamilguardian.com/news-region/tamil-affairs?page={i}"
  response = requests.get(url).content
  bs = BeautifulSoup(response, 'lxml')

  listOfNews = bs.find('div', class_="view-content")
  articles = listOfNews.find_all('div', class_="views-row")

  for article in articles:
    title = article.find('div', class_="title-body").find('div',␣
↪class_="post-title").find('h2')
    contentURL = title.find('a').attrs['href']
    moreInfo = requests.get(f"https://www.tamilguardian.com{contentURL}").
↪content
    bs2 = BeautifulSoup(moreInfo, 'lxml')
    dateString = bs2.find('div', class_="content-first").find('div',␣
↪class_="post-date-1").text.strip()
    formatedDate = datetime.strptime(dateString, date_format)
    print(formatedDate)

    if(target_date <= formatedDate):
      content = bs2.find('div', class_="content-first").find('div',␣
↪class_="content").find('div', class_="field-item even").text.strip()
      news.append([title.text.strip(), formatedDate.strftime("%B %d, %Y"),␣
↪content])
```

```
    else:
      reached = True
      break

  print(f"Total News as of now: {len(news)}")
  if(reached):
    print("Reached the give date limit.")
    break
```

[ ]:
```
base_url = "https://www.dailymirror.lk"
categories = {
    "featured": "/features/131",
    "news": "/news/209",
    "financial": "/financial-news/265",
    "other": "/Other/117",
    "sports": "/sports",
    "expose": "/expose/333",
    "hardtalk": "/hard-talk/334",
    "business": "/business-news/273",
}

complete_article_list = []

# loop through the categories
for category_name, category_url in categories.items():
    print(f"Running for: {category_name} : {base_url}{category_url}")
    limit_reached = False
    loop_count = 0
    page_size = 30 # number of articles per page, this is fixed
    article_list = []

    while not limit_reached:
        url = f"{base_url}{category_url}/{loop_count * page_size}"
        print(f"Scraping page #{loop_count + 1} from {url}")
        time.sleep(1)
        response = requests.get(url)
        soup = BeautifulSoup(response.content, "html.parser")

        if(category_name != "sports"):
            main_div =  soup.find('div', id='breakingnewsads')
            articles = main_div.find_all('div', class_='lineg')
        else:
            main_div =  soup.find('div', class_='inleft')
            # Find all the div elements with class "lineg" within the main div
            articles = main_div.find_all('div', class_='row')
```

```
        # Check if the page has no articles
        if len(articles) == 0:
            limit_reached = True
            break

        for article in articles:
            date_time_str = article.find('span', class_='gtime').text.strip()
            date_time = datetime.strptime(date_time_str, '%d %b %Y')
            date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")
            if date_time <= datetime(2022, 6, 1):
                limit_reached = True
                break  # Break out of the loop

            title = article.find('h3', class_='cat-hd-tx').text.strip()
            excerpt = article.find_all('p')[1].text.strip()
            url = article.select_one("a")["href"]

            time.sleep(1)
            # ignore article if url has "https://www.dailymirror.lk/
↪infographics"
            if "https://www.dailymirror.lk/infographics" in url:
                continue

            try:
                print(f"Scraping article #{url}")
                # get the full article content from the article url
                article_response = requests.get(url)
                article_soup = BeautifulSoup(article_response.content, "html.
↪parser")
                article_content = article_soup.find('header',␣
↪class_='inner-content').text.strip()
            except Exception as e:
                print(f"Error scraping article #{url}: {str(e)}")
                continue

            article_dict = {
                "title": title,
                "excerpt": excerpt,
                "date_time": date_time_iso,
                "url": url,
                "content": article_content,
                "category": category_name,
                "source": "daily_mirror"
            }
            article_list.append(article_dict)
```

```
            complete_article_list.append(article_dict)
        loop_count += 1
    print(f"Total articles scraped for {category_name}: {len(article_list)}")
    with open(f"datasets/daily_mirror/{category_name}.json", "w") as file:
        json.dump(article_list, file)


print(f"Total articles scraped: {len(article_list)}")

# Save the articles as a JSON array
with open("datasets/daily_mirror.json", "w") as file:
    json.dump(complete_article_list, file)

print("Data collection completed for all categories.")

# read datasets/daily_mirror.json and print number of articles collected
with open(f'datasets/daily_mirror.json', 'r') as f:
    daily_mirror_data = json.load(f);

print(f'Number of news items collected from daily mirror:␣
 ↪{len(daily_mirror_data)}')
```

[ ]:
```
base_url = "https://adaderana.lk/news.php?nid={id}"
start_id = 82795
batch_size = 100
page = 1
articles = []


def parse_datetime(datetime_str):
    # Parse the date and time string into the desired format: "2022-06-01T07:42:
 ↪04.000Z"
    datetime_obj = datetime.strptime(datetime_str, "%B %d, %Y   %I:%M %p")
    datetime_obj -= timedelta(hours=5, minutes=30)  # Convert to Sri Lankan␣
 ↪time (GMT +5:30)
    formatted_datetime = datetime_obj.strftime("%Y-%m-%dT%H:%M:%S.000Z")
    return formatted_datetime


def scrape_article(article_id):
    url = base_url.format(id=article_id)
    # print(f"Scraping article #{article_id} from {url}")
    headers = {
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
 ↪avif,image/webp,image/apng,/;q=0.8,application/signed-exchange;v=b3;q=0.7",
        "Accept-Encoding": "gzip, deflate, br",
        "Accept-Language": "en-GB,en-US;q=0.9,en;q=0.8",
```

```
        "Cache-Control": "max-age=0",
        "Cookie": "",
        "Sec-Ch-Ua": '"Not.A/Brand";v="8", "Chromium";v="114", "Google Chrome";
↪v="114"',
        "Sec-Ch-Ua-Mobile": "?0",
        "Sec-Ch-Ua-Platform": '"macOS"',
        "Sec-Fetch-Dest": "document",
        "Sec-Fetch-Mode": "navigate",
        "Sec-Fetch-Site": "none",
        "Sec-Fetch-User": "?1",
        "Upgrade-Insecure-Requests": "1",
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)␣
↪AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36"
    }
    try:
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            soup = BeautifulSoup(response.content, "html.parser")
            title = soup.select_one("h1").text.strip()
            datetime_str = soup.select_one(".news-datestamp").text.strip()
            content_element = soup.select_one(".news-content")
            content_html = str(content_element)
            if(content_html == "<div class=\"news-content\">\n</div>"):
                return False
            parsed_datetime = parse_datetime(datetime_str)
            article = {
                "id": article_id,
                "title": title,
                "datetime": parsed_datetime,
                "content": content_html,
                "url": url,
                "source": "adaderana"
            }
            articles.append(article)
            return True
    except Exception as e:
        print(f"Error scraping article #{article_id}: {str(e)}")
    return False

while True:
    if not scrape_article(start_id):
        print(f"Could not scrape article #{start_id}. Stopping.")
        print(f"Total articles scraped: {len(articles)}")
        break
    start_id += 1
    # Delay for 1 seconds
    time.sleep(1)
```

```
# Save the articles as a JSON array
with open("datasets/adaderana.json", "w") as file:
    json.dump(articles, file)


print(f"Scraped {len(articles)} articles from Ada Derana.")
```