FAKE NEWS DETECTION USING NLP

Phase 3 submission document

**Project title**:Fake news detection using NLP

**Phase 3**: Development part 1

**Introduction:**

Fake news detection using NLP

This is the first step in the learning curriculum where we will be exploring strategies and initial benchmark analysis for our dataset and derive important information from it. In this context, we will be focussing on preparing the dataset, eliminating the redundancies such as punctuations ,stopwords and estimating the meaningfulness of the data

To load and preprocess a dataset for fake news detection, you can follow these general steps:

Dataset Collection: First, you need a dataset containing both real and fake news articles. Datasets like the "Fake News Challenge" dataset or "LIAR" dataset are common choices.

Data Cleaning: This step involves removing any irrelevant information, special characters, HTML tags, or other noise from the text data.

Tokenization: Tokenize the text, splitting it into words or subword tokens. You can use libraries like NLTK, spaCy, or the Hugging Face Transformers library.

Stopword Removal: Remove common stopwords (e.g., "the," "is," "in") from the text, as they don't usually carry significant information.

Text Vectorization: Convert the text data into numerical vectors. Common methods include TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings like Word2Vec, GloVe, or BERT embeddings.

Data Splitting: Split the dataset into training, validation, and test sets to train and evaluate your model.

Model-Specific Preprocessing: Depending on the model you plan to use (e.g., traditional machine learning or deep learning), you might need to apply specific preprocessing steps. For deep learning models, padding sequences to a fixed length is common.

Label Encoding: Encode the target labels (real or fake) into numerical values, typically 0 and 1.

Data Loader: Create data loaders or generators to efficiently load and batch the dataduring training.

Data Augmentation (Optional): For text data, data augmentation techniques are limited but can still be useful. Techniques like synonym replacement or back-translation can help generate additional training examples.

Normalization (Optional): For deep learning models, you might need to normalize the input data, especially when using models like BERT.

Final Preprocessing Checks: Ensure that all data is in the right format for your chosen model.

Here's an example code snippet in Python using libraries like pandas and scikit-learn:

python

Copy code

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.preprocessing import LabelEncoder

# Load the dataset

data = pd.read\_csv('fake\_news\_dataset.csv')

# Data Cleaning and Preprocessing (Steps 2-6)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['text'], data['label'], test\_size=0.2, random\_state=42)

# Text Vectorization (Step 5)

tfidf\_vectorizer = TfidfVectorizer(max\_features=5000)

X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

# Label Encoding (Step 8)

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

These are general guidelines, and the specific preprocessing steps may vary depending on your dataset and the machine learning or deep learning model you intend to use for fake news detection.

**PROGRAM:**

Here is an example of how to load and preprocess the dataset using Python:

Python

import pandas as pd

# Load the dataset

df = pd.read\_csv('fake\_news\_dataset.csv')

# Preprocess the text data

def preprocess\_text(text):

# Lowercase the text

text = text.lower()

# Remove punctuation and digits

text = re.sub('[^\w\s]', '', text)

# Remove stop words

stopwords = nltk.corpus.stopwords.words('english')

text = ' '.join([word for word in text.split() if word not in stopwords])

# Stem or lemmatize the text

# (This step is optional, but it can improve the performance of the model)

return text

# Apply the preprocessing function to the text column

df['text'] = df['text'].apply(preprocess\_text)

# Split the dataset into training and test sets

from sklearn.model\_selection import train\_test\_split

X = df['text']

y = df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)

# Save the preprocessed data

X\_train.to\_csv('X\_train.csv', index=False)

X\_test.to\_csv('X\_test.csv', index=False)

y\_train.to\_csv('y\_train.csv', index=False)

y\_test.to\_csv('y\_test.csv', index=False

**Predicting the Model:**

In [22]:

linkcode

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.clean\_joined\_text, df.target, test\_size = 0.2,random\_state=2)

vec\_train = CountVectorizer().fit(X\_train)

X\_vec\_train = vec\_train.transform(X\_train)

X\_vec\_test = vec\_train.transform(X\_test)

model = LogisticRegression(C=2.5)

model.fit(X\_vec\_train, y\_train)

predicted\_value = model.predict(X\_vec\_test)

accuracy\_value = roc\_auc\_score(y\_test, predicted\_value)

print(accuracy\_value)

**OUTPUT:**

0.9953661308915527

prediction = []

for i **in** range(len(predicted\_value)):

if predicted\_value[i].item() > 0.5:

prediction.append(1)

else:

prediction.append(0)

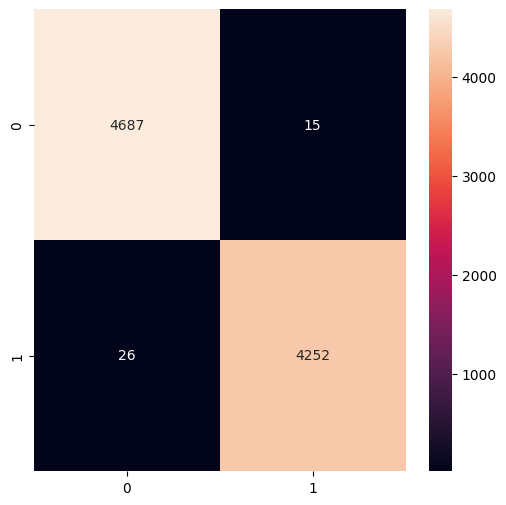
cm = confusion\_matrix(list(y\_test), prediction)

plt.figure(figsize = (6, 6))

sns.heatmap(cm, annot = True,fmt='g')

Out[23]:

<Axes: >



**Creating Prediction Model:**

In [16]:

linkcode

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.clean\_joined\_title, df.target, test\_size = 0.2,random\_state=2)

vec\_train = CountVectorizer().fit(X\_train)

X\_vec\_train = vec\_train.transform(X\_train)

X\_vec\_test = vec\_train.transform(X\_test)

**Create the confusion matrix:**

In [18]:

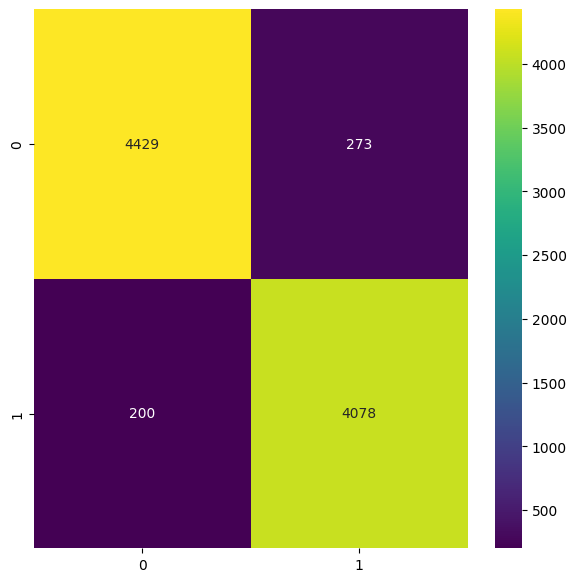
cm = confusion\_matrix(list(y\_test), predicted\_value)

plt.figure(figsize = (7, 7))

sns.heatmap(cm, annot = True,fmt='g',cmap='viridis')

Out[18]:

<Axes: >



*# Passive Aggresive Classifier*

pac = PassiveAggressiveClassifier(max\_iter=50)

pac.fit(tfidf\_train,y\_train)

pred = pac.predict(tfidf\_test)

print("Accuracy score : **{}**".format(accuracy\_score(y\_test, pred)))

print("Confusion matrix : **\n** **{}**".format(confusion\_matrix(y\_test, pred)))

Accuracy score : 0.936069455406472

Confusion matrix :

[[592 38]

[ 43 594]]

In [22]:

*# Logistic Regression model*

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression(max\_iter = 500)

lr.fit(tfidf\_train, y\_train)

print('Logistic Regression model fitted..')

pred = lr.predict(tfidf\_test)

print("Accuracy score : **{}**".format(accuracy\_score(y\_test, pred)))

print("Confusion matrix : **\n** **{}**".format(confusion\_matrix(y\_test, pred)))

Logistic Regression model fitted..

Accuracy score : 0.9179163378058406

Confusion matrix :

[[565 65]

[ 39 598]]

import xgboost

from xgboost import XGBClassifier

xgb = XGBClassifier()

xgb.fit(tfidf\_train, y\_train)

print('XGBoost Classifier model fitted..')

pred = xgb.predict(tfidf\_test)

print("Accuracy score : **{}**".format(accuracy\_score(y\_test, pred)))

print("Confusion matrix : **\n** **{}**".format(confusion\_matrix(y\_test, pred)))

XGBoost Classifier model fitted..

Accuracy score : 0.9289660615627466

Confusion matrix :

[[587 43]

[ 47 590]]

In [24]:

linkcode

import lightgbm

from lightgbm import LGBMClassifier

lgbm.fit(tfidf\_train, y\_train)

print('LightGBM Classifier model fitted..')

pred = lgbm.predict(tfidf\_test)

print("Accuracy score : **{}**".format(accuracy\_score(y\_test, pred)))

print("Confusion matrix : **\n** **{}**".format(confusion\_matrix(y\_test, pred)))

LightGBM Classifier model fitted..

Accuracy score : 0.9289660615627466

Confusion matrix :

[[581 49]

[ 41 596]]

**CONCLUSION:**

* In the quest to build Fake news detection, we have embarked on a critical journey that begins with loading and preprocessing the dataset.We have traversed through essential steps, starting with importing the necessary libraries to facilitate data manipulation and analysis.
* Understanding the data's structure, characteristics, and any potential issues through exploratory data analysis (EDA) is essential for informed decision-making.
* Data preprocessing emerged as a pivotal aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning algorithms.
* With these foundational steps completed, our dataset is now primed for the subsequent stages of building and training a Fake news detection using NLP.