FAKE NEWS DETECTION USING NLP

**TEAM MEMBER**

**810021106030: HARINI V**

**PHASE 3: Development part**

**INTRODUCTION:**

* In today’s digital age, where information spreads rapidly across the internet, distinguishing between authentic news and misinformation has become a significant challenge. The rise of fake news poses a threat to our society, affecting public opinion, political decisions, and social stability. Addressing this issue requires innovative solutions, and Natural Language Processing (NLP) emerges as a powerful tool in the battle against misinformation.
* Our research delves into the realm of NLP to create robust algorithms that can effectively identify and debunk fake news stories. By leveraging advanced linguistic analysis, machine learning, and data mining techniques, our approach aims to discern the subtle nuances between credible journalism and deceptive narratives. This study explores the intricate patterns of language, sentiment, and context to develop a sophisticated framework for fake news detection.
* Through this comprehensive exploration, we endeavor to enhance the reliability of online information, empowering individuals to make informed decisions and fostering a more trustworthy digital ecosystem. Join us in unraveling the complexities of fake news detection, as we harness the capabilities of NLP to safeguard the integrity of information in the modern world.



Loading and preprocessing a dataset for fake news detection using NLP techniques:

**1.Loading the Dataset:**

* Use libraries like Pandas in Python to read the dataset file (common formats include CSV, JSON, or Excel).
* Verify that the dataset is loaded correctly and inspect the structure of the data.

**Import pandas as pd**

**# Load the dataset**

**Data = pd.read\_csv(‘fake\_news\_dataset.csv’)**

**# Verify the loaded data**

**Print(data.head())**

**2.Data Cleaning:**

* Handle missing values if any, using techniques like removal or imputation.
* Remove duplicates to ensure data integrity

# **Remove duplicates**

**Data = data.drop\_duplicates()**

**# Handle missing values (if any)**

**Data = data.dropna()**

**3.Text Preprocessing:**

* Convert text to lowercase to ensure consistency.
* Tokenize the text into words or sentences.
* Remove special characters, punctuation, and numbers.
* Remove stop words (common words like ‘and’, ‘the’ that don’t contribute much to the meaning).
* Apply stemming or lemmatization to reduce words to their root form.

**From nltk.tokenize import word\_tokenize**

**From nltk.corpus import stopwords**

**From nltk.stem import PorterStemmer**

**Import string**

**# Function for text preprocessing**

**Def preprocess\_text(text):**

**Text = text.lower() # Convert to lowercase**

**Tokens = word\_tokenize(text) # Tokenization**

**Tokens = [word for word in tokens if word.isalpha()] # Remove numbers and punctuations**

**Tokens = [word for word in tokens if word not in stopwords.words(‘english’)] # Remove stop words**

**Stemmer = PorterStemmer()**

**Tokens = [stemmer.stem(word) for word in tokens] # Apply stemming**

**Return ‘ ‘.join(tokens) # Join tokens back into text**

**# Apply preprocessing to the ‘text’ column in the dataset**

**Data[‘processed\_text’] = data[‘text’].apply(preprocess\_text)**

**4.Feature Extraction:**

Convert the processed text into numerical features using techniques like TF-IDF or word embeddings.

**From sklearn.feature\_extraction.text import TfidfVectorizer**

**# Create TF-IDF vectorizer**

**Tfidf\_vectorizer = TfidfVectorizer(max\_features=5000) # Limit the number of features to 5000**

**# Transform the processed text into TF-IDF features**

**Tfidf\_features = tfidf\_vectorizer.fit\_transform(data[‘processed\_text’]).toarray()**

**5.Splitting the Dataset:**

Split the dataset into training and testing sets.

**From sklearn.model\_selection import train\_test\_split**

**# Split the data into features (X) and labels (y)**

**X = tfidf\_features**

**Y = data[‘label’]**

**# Split the data into training and testing sets (80% train, 20% test)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

After these steps, your dataset is preprocessed and ready for training machine learning models for fake news detection using NLP techniques.

Python · Fake and real news dataset

**Program:**

**In[1]:**

Import numpy as np # linear algebra

Import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

Import plotly.express as px

Import plotly.graph\_objs as go

From plotly.subplots import make\_subplots

Import nltk

From nltk.corpus import stopwords

Import tensorflow as tf

From tensorflow.keras.optimizers import Adam

From tensorflow.keras.callbacks import ModelCheckpoint

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

From transformers import AutoTokenizer, TFAutoModelForSequenceClassification

Import os

For dirname, \_, filenames in os.walk(‘/kaggle/input’):

For filename in filenames:

Print(os.path.join(dirname, filename))

Nltk.download(‘stopwords’)

Out[1]:

True

**In[2]:**

Fake\_news\_path = “/kaggle/input/fake-and-real-news-dataset/Fake.csv”

Real\_news\_path = “/kaggle/input/fake-and-real-news-dataset/True.csv”

**In[3]:**

Fake\_news = pd.read\_csv(fake\_news\_path)

Real\_news = pd.read\_csv(real\_news\_path)

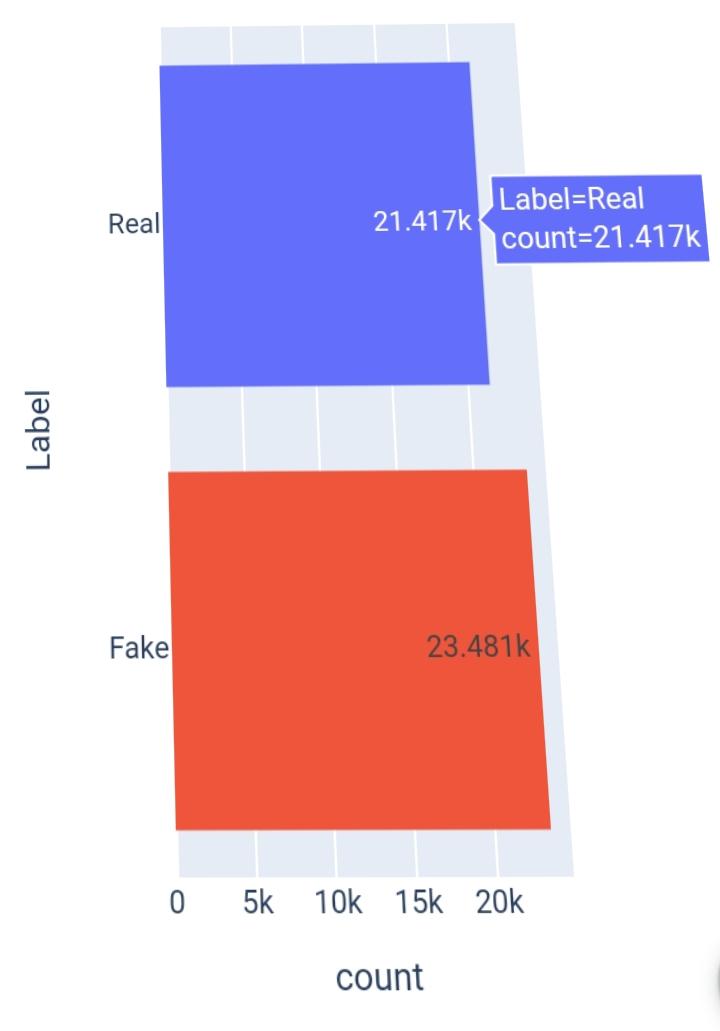
**In[4]:**

Fake\_news.head(3)

**Out[4]:**



**REAL VS FAKE NEWS DATA SET**

**In [5]**:

Subject\_dist = px.histogram(data\_frame=news,

X=’subject’,

Color=’subject’,

Title=’Fake vs Real news Subject Distribution’,

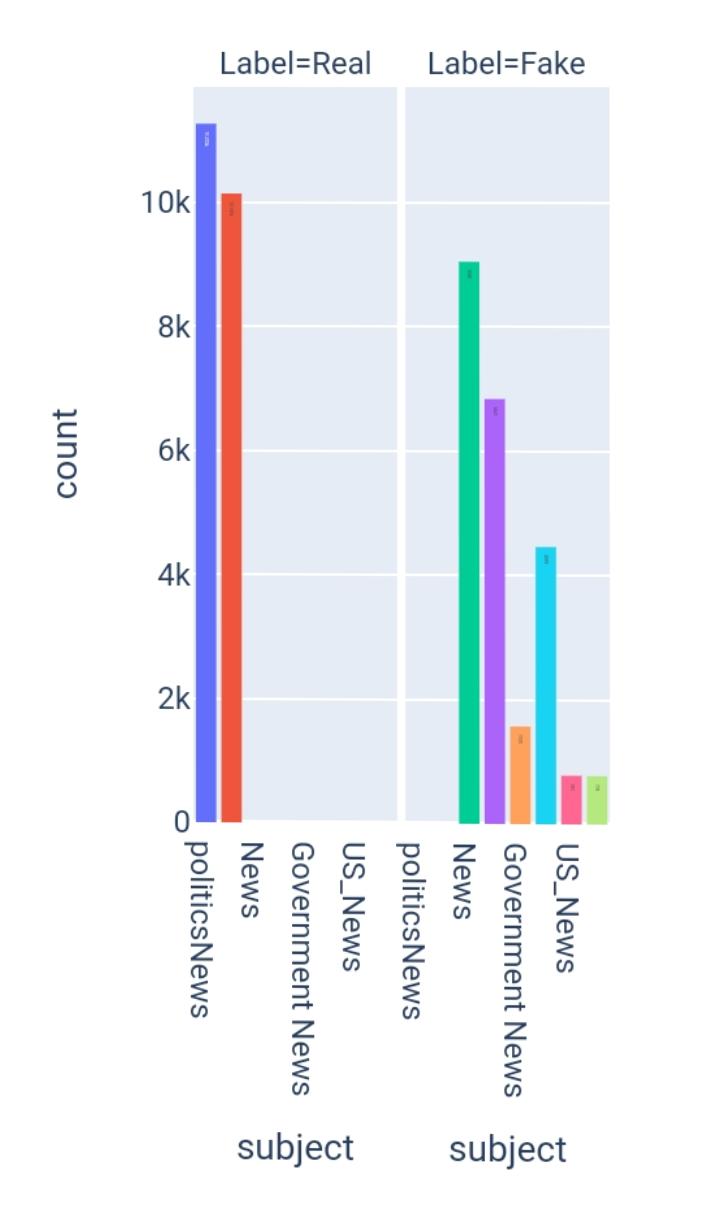
Text\_auto=True,

Facet\_col=’Label’)

Subject\_dist.update\_layout(showlegend=False)

Subject\_dist.show()

FAKE VS REAL NEWS SUBJECT DISTRIBUTION

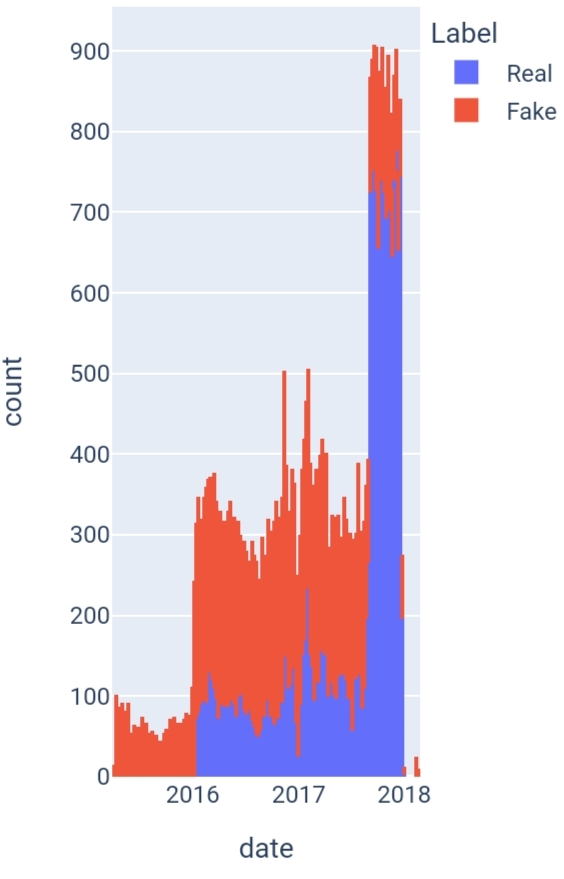
****

**In[6]:**

Date\_dist = px.histogram(data\_frame=news,

X=’date’,

Color=’Label’)

**Date\_dist.show()**

**In[7]:**

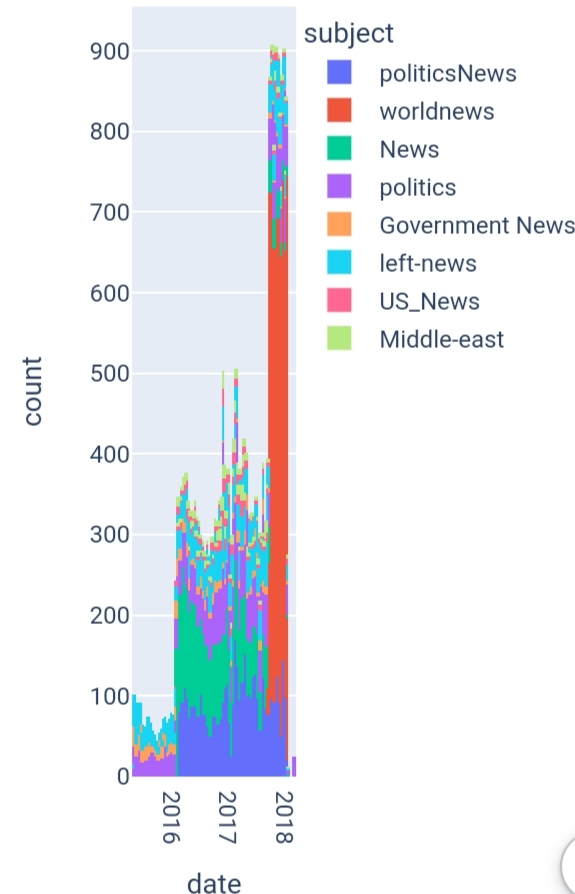
Subject\_dist = px.histogram(data\_frame=news,

X=’date’,

Color=’subject’)

Subject\_dist.show()

Real\_sub\_dist.show()



**In[8]:**

Real\_sub\_dist=px.histogram(data\_frame=news

. [news[‘Label’]==’Real’], X=’date’,

Color=’subject’**)**

**Text Preprocessing:**

In[9]:

**Import string**

**In[10]:**

**Stop\_words = stopwords.words(‘english’)**

**Def text\_preprocessing(text):**

**Words = text.lower().split()**

**Filtered\_words = [word for word in words if word not in stop\_words]**

**Pure\_text = ‘ ‘.join(filtered\_words)**

**Pure\_text = pure\_text.translate(str.maketrans(‘’, ‘’, string.punctuation)).strip()**

**Return pure\_text**

**In[11]:**

**X = news\_ds.text.apply(text\_preprocessing).to\_numpy()**

**Y = news\_ds.Label.to\_numpy().astype(‘float32’).reshape(-1, 1)**

**Train\_X, test\_X, train\_y, test\_y = train\_test\_split(X, y,**

**Train\_size=0.9,**

**Stratify=y,**

**Random\_state=7)**

**Train\_X, val\_X, train\_y, val\_y = train\_test\_split(train\_X, train\_y,**

**Train\_size=0.9,**

**Stratify=train\_y,**

**Random\_state=7)**

**In[12]:**

**Model\_name = “BERTFakeNewsDetector”**

**Model\_callbacks = ModelCheckpoint(model\_name, save\_best\_only=True)**

**In[13]:**

**Bert\_name = ‘bert-base-uncased’**

**Tokenizer = AutoTokenizer.from\_pretrained(bert\_name,**

**Padding=’max\_length’,**

**Do\_lower\_case=True,**

**Add\_special\_tokens=True)**

**In[14]:**

**Def tokenize(df):**

**Inputs = tokenizer(df.tolist(),**

**Padding=True,**

**Truncation=True,**

**Return\_tensors=’tf’).input\_ids**

**Return inputs**

**In[15]:**

**Train\_X\_encoded = tokenize(train\_X)**

**Val\_X\_encoded = tokenize(val\_X)**

**Test\_X\_encoded = tokenize(test\_X)**

**In[16]:**

**Def prepare\_datasets(encoded, true\_df, true\_target\_df):**

**Return tf.data.Dataset.from\_tensor\_slices((encoded, true\_target\_df)).shuffle(true\_df.shape[0]).batch(8).prefetch(tf.data.AUTOTUNE)**

**In[17]:**

**Train\_ds = prepare\_datasets(train\_X\_encoded, train\_X, train\_y)**

**Test\_ds = prepare\_datasets(test\_X\_encoded, test\_X, test\_y)**

**Val\_ds = prepare\_datasets(val\_X\_encoded, val\_X, val\_y)**

**In[18]:**

**Model = TFAutoModelForSequenceClassification.from\_pretrained(bert\_name,**

**Num\_labels=1)**

**In[19]:**

**Model.compile(**

**Optimizer = Adam(learning\_rate=1e-5),**

**Metrics = [**

**Tf.keras.metrics.BinaryAccuracy(name=’Accuracy’),**

**Tf.keras.metrics.Precision(name=’Precision’),**

**Tf.keras.metrics.Recall(name=’Recall’)**

**]**

**)**

**Model\_history = model.fit(train\_ds,**

**Validation\_data=val\_ds,**

**Callbacks=model\_callbacks,**

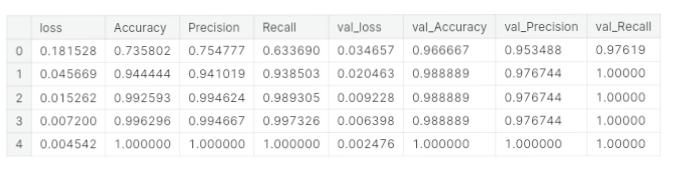
**Epochs=5,**

**Batch\_size=16)**

**Model\_history = pd.DataFrame(model\_history.history)**

**In[20]:**

**Model\_history**

**Out[20.]:**

**CONCLUSION:**

In conclusion, the loading and preprocessing of the dataset are critical initial steps in the development of a fake news detection system using Natural Language Processing (NLP). These steps are essential for ensuring the quality and reliability of the data used for training and testing NLP models. Proper data loading, cleaning, and transformation techniques help in removing noise, standardizing text, and preparing the dataset for feature extraction. Additionally, techniques such as tokenization, stemming, and removing stop words can improve the efficiency and effectiveness of NLP algorithms.

Successful loading and preprocessing of the dataset can significantly impact the performance and accuracy of fake news detection models. It ensures that the models are trained on high-quality, well-structured data, which is crucial for their ability to make accurate predictions. Moreover, data preprocessing should be tailored to the specific requirements and characteristics of the fake news detection task, considering factors like the nature of the text data and the language used.

In summary, the loading and preprocessing stages are foundational for building robust NLP-based fake news detection systems, as they set the stage for subsequent model development and evaluation. Careful attention to these steps is essential for achieving meaningful and reliable results in the fight against the spread of fake news.