**FAKE NEWS DETECTION USING NLP**

TEAM MEMBER

810021106030:HARINI V

PHASE 5: PROJECT DOCUMENTATION AND SUBMISSION

**PROJECT DEFINITION:**

The goal of this project is to develop a system that uses Natural Language Processing (NLP) techniques to

automatically identify and classify fake news articles from legitimate news sources. Fake news has become a significant problem in the age of digital media, and this project aims to contribute to the efforts in combating the spread of

misinformation.To get the accurately classified collection of news as real or fake we have to build a machine learning

model.

**DESIGN THINKING:**

## Data Collection and Preparation:

* Acquire a diverse dataset of news articles, including both genuine and fake news sources.
* Clean and preprocess the use text data, including tasks such as tokenization, stop word removal, and

stemming/lemmatization.

## Feature Extraction:

* Utilize NLP techniques to extract relevant features from the text data.
* Features may include word embeddings (e.g., Word2Vec or GloVe), TF-IDF vectors, and linguistic features (e.g.,

sentiment analysis, readability scores).

## Data Splitting:

* + Divide the dataset into training, validation, and testing sets to train and evaluate the machine learning models.

## Model Development:

* + Implement and experiment with various machine learning and deep learning models for fake news detection.

Common choices include:

* + - Logistic Regression
    - Naive Bayes
    - Random Forest
    - LSTM or Transformer-based models (e.g., BERT)
  + Fine-tune models on the training data and validate their performance using the validation set.

## Model Evaluation:

* + Assess the models’ performance using appropriate

evaluation metrics such as accuracy, precision, recall, F1- score, and ROC-AUC.

* + Employ cross-validation to ensure the model’s robustness.

## Testing and Validation:

* + Conduct thorough testing to ensure the system’s reliability and accuracy in real-world scenarios.

## Continuous Improvement:

* + Monitor the system’s performance over time and update it with new data and improvements to stay effective in

detecting evolving fake news tactics.

By implementing these modules, this project aims to provide a robust and reliable solution for fake news detection,

helping users make more informed decisions when consuming news content.

Certainly, there are several tools and technologies used in fake news detection leveraging Natural Language Processing (NLP). Some popular ones up until my last update in January 2022 include:

1. BERT (Bidirectional Encoder Representations from Transformers)

2.GPT (Generative Pre-trained Transformer)

3.LSTM (Long Short-Term Memory)

4.Random Forest and Support Vector Machines (SVM)

5.Word Embeddings (Word2Vec, GloVe, etc.)

6.TF-IDF (Term Frequency-Inverse Document Frequency)

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.

Deep learning models like LSTM, BERT, and GPT have shown remarkable success in fake news detection.

LSTM(LONG SHORT-TERM MEMORY):

LSTM can capture temporal dependencies in textual data, which is beneficial for detecting fake news since the order of words can be critical.

You can preprocess news articles as sequences of words or embeddings and use LSTM to analyze them, considering the sequential nature of language.

BERT(BIDERECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS):

* BERT pre-trained models have a deep understanding of language context and semantics, making them effective for fake news detection.
* Fine-tuning BERT on a labeled dataset specific to fake news detection tasks can significantly improve accuracy.
* Utilize BERT’s ability to encode text into contextual embeddings and feed them to a classifier for prediction.
* Incorporating these techniques may involve natural language processing (NLP) libraries like TensorFlow or PyTorch and pre- trained models like GPT, BERT, or LSTM implementations in these libraries. It’s essential to preprocess your data, fine-tune the models, and evaluate their performance rigorously to achieve improved fake news detection accuracy.

Data Preprocessing:

* + Collect a dataset of news articles, tweets, or any text data labeled as either real or fake news.
  + **Tokenize the text:** Split the text into individual words or subword tokens.
  + Create a vocabulary: Build a dictionary of unique words or tokens in your dataset.
  + Convert text to numerical data: Map words/tokens to their corresponding numerical representations (word embeddings or one-hot encoding).

Feature Engineering:

* Advanced NLP techniques involve the creation of sophisticated linguistic features, such as sentiment analysis, emotion detection, and stance analysis.
* These features can reveal the emotional tone and stance of a news article, helping to assess its credibility

Fake News Datasets:

* + The development of large, labeled fake news datasets is crucial for training and evaluating advanced models.
  + These datasets enable researchers to fine-tune models specifically for fake news detection, improving their accuracy.

Real-time Monitoring:

* + - Continuous monitoring of news sources and social media platforms is essential to detect emerging fake news stories. Advanced NLP techniques can automate this process and provide timely alerts.

**Fake news detection using BERT**

In[1]:

Import numpy as np Import pandas as pd

Import matplotlib.pyplot as plt Import os

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

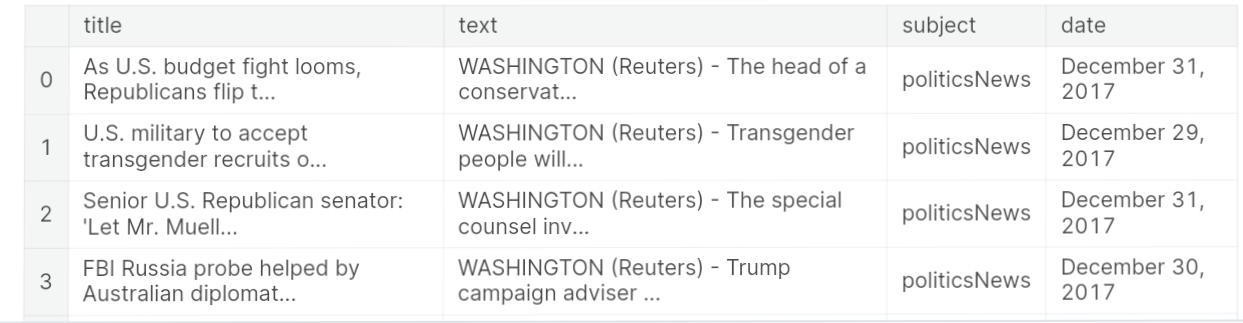
Print(os.path.join(dirname, filename)) Read Dataset

In[2]:

True\_news = pd.read\_csv(‘/kaggle/input/fake-and-real-news-dataset/True.csv’) Fake\_news = pd.read\_csv(‘/kaggle/input/fake-and-real-news-dataset/Fake.csv’) In[3]:

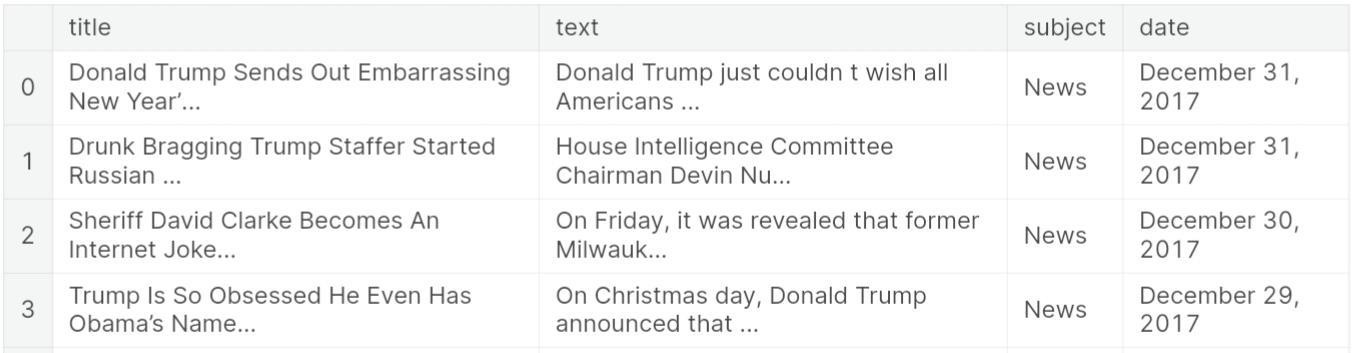
True\_news.head()

Out[3]:



In[4]:

Fake\_news.head() Out[4]:



**Make new column with labels 1 for ‘true’, 0 for ‘false’.**

**In[5]:**

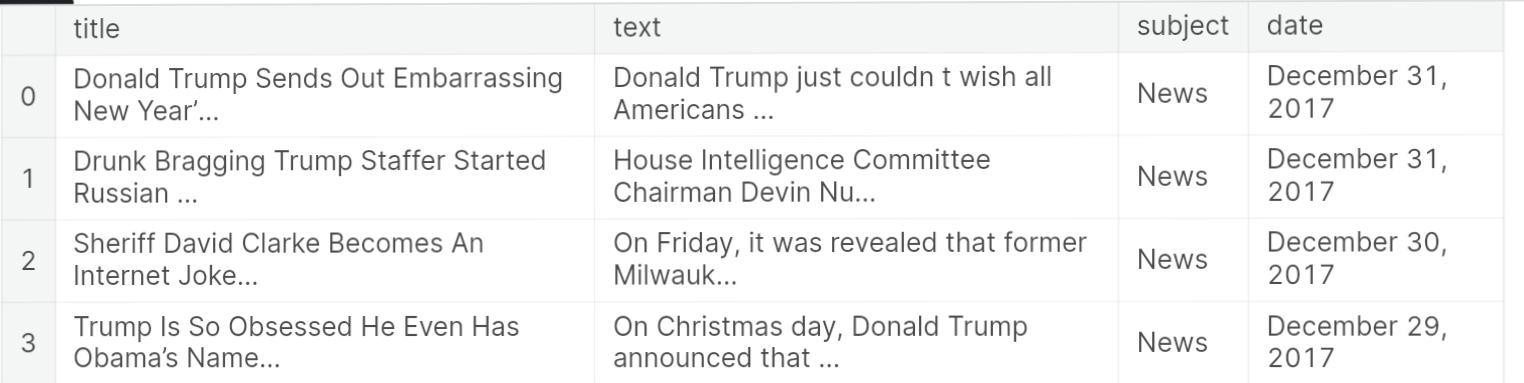
True\_news[‘target’] = [1]\*len(true\_news) Fake\_news[‘target’] = [0]\*len(fake\_news)

**Append both the data to one dataframe. In[6]:**

Dataset = true\_news.append(fake\_news).sample(frac = 1).reset\_index().drop(columns = [‘index’])

Dataset.head()

**Out[6]:**



**Visualize data to see the true/fake news ratio. In[7]:**

Label\_size = [dataset[‘target’].sum(), len(dataset[‘target’])-dataset[‘target’].sum()]

Plt.pie(label\_size,explode=[0.1,0.1],colors=[‘firebrick’,’navy’],startangle=90,shadow= True,labels=[‘Fake’,’True’],autopct=’%1.1f%%’)

**Out[7]:**

([<matplotlib.patches.Wedge at 0x7c202363e7d0>,

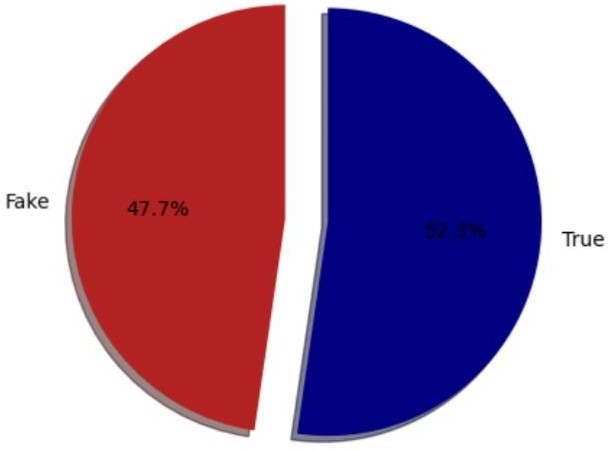
<matplotlib.patches.Wedge at 0x7c202363d090>],

[Text(-1.1968727148445069, 0.08657773651892332, ‘Fake’),

Text(1.1968727229504943, -0.08657762445961172, ‘True’)],

[Text(-0.6981757503259622, 0.050503679636038606, ’47.7%’),

Text(0.698175755054455, -0.05050361426810683, ’52.3%’)])



**In[8]:**

!pip install tensorflow-hub

**In[9]:**

**!pip install -q tf-models-official In[10]:**

From sklearn.model\_selection import train\_test\_split Import tensorflow as tf

Import tensorflow\_hub as hub Import tensorflow\_text as text

From official.nlp import optimization

Tfhub\_handle\_preprocess = ‘https://tfhub.dev/tensorflow/bert\_en\_uncased\_preprocess/3’

Tfhub\_handle\_encoder = ‘https://tfhub.dev/tensorflow/small\_bert/bert\_en\_uncased\_L-4\_H-512\_A-8/1’

**Split data to train, validation, test set. In[11]:**

X\_train, x\_test, y\_train, y\_test = train\_test\_split(dataset[‘title’],dataset[‘target’],test\_size = 0.3,

random\_state = 42)

X\_val, xval\_test, y\_val, yval\_test = train\_test\_split(x\_test,y\_test, test\_size

=0.5,random\_state = 42)

**This is the classifier model that uses BERT model. This code block is taken from here.**

**In[12]:**

Def build\_classifier\_model():

Text\_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name=’text’)

Preprocessing\_layer = hub.KerasLayer(tfhub\_handle\_preprocess, name=’preprocessing’)

Encoder\_inputs = preprocessing\_layer(text\_input)

Encoder = hub.KerasLayer(tfhub\_handle\_encoder, trainable=True, name=’BERT\_encoder’)

Outputs = encoder(encoder\_inputs) Net = outputs[‘pooled\_output’]

Net = tf.keras.layers.Dropout(0.1)(net)

Net = tf.keras.layers.Dense(1, activation=None, name=’classifier’)(net) Return tf.keras.Model(text\_input, net)

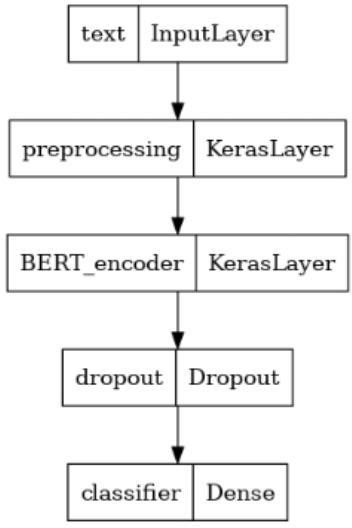
**In[13]:**

Classifier\_model = build\_classifier\_model()

**In[14]:**

Tf.keras.utils.plot\_model(classifier\_model)

**Out[14]:**



**In[15]:**

Loss = tf.keras.losses.BinaryCrossentropy(from\_logits=True) Metrics = tf.metrics.BinaryAccuracy()

**In[16]:**

Epochs = 8

Steps\_per\_epoch = 10

Num\_train\_steps = steps\_per\_epoch \* epochs Num\_warmup\_steps = int(0.1\*num\_train\_steps) Init\_lr = 1e-5

Optimizer = optimization.create\_optimizer(init\_lr=init\_lr,

Num\_train\_steps=num\_train\_steps, Num\_warmup\_steps=num\_warmup\_steps, Optimizer\_type=’adam w’)

**In[17]:**

Classifier\_model.compile(optimizer=optimizer, Loss=loss,

Metrics=metrics)

**In[18]:**

Print(f’Training model with {tfhub\_handle\_encoder}’) History = classifier\_model.fit(x=x\_train,y=y\_train,

Validation\_data=(x\_val,y\_val), Epochs=epochs)

**In[19]:**

Loss, accuracy = classifier\_model.evaluate(x\_test) Print(f’Loss: {loss}’)

Print(f’Accuracy: {accuracy}’)

**In[20]:**

History\_dict = history.history Print(history\_dict.keys())

Acc = history\_dict[‘binary\_accuracy’] Val\_acc = history\_dict[‘val\_binary\_accuracy’] Loss = history\_dict[‘loss’]

Val\_loss = history\_dict[‘val\_loss’]

Epochs = range(1, len(acc) + 1) Fig = plt.figure(figsize=(10, 6)) Fig.tight\_layout()

Plt.subplot(2, 1, 1)

# r is for “solid red line”

Plt.plot(epochs, loss, ‘r’, label=’Training loss’) # b is for “solid blue line”

Plt.plot(epochs, val\_loss, ‘b’, label=’Validation loss’) Plt.title(‘Training and validation loss’) Plt.xlabel(‘Epochs’)

Plt.ylabel(‘Loss’) Plt.legend()

Plt.subplot(2, 1, 2)

Plt.plot(epochs, acc, ‘r’, label=’Training acc’) Plt.plot(epochs, val\_acc, ‘b’, label=’Validation acc’)

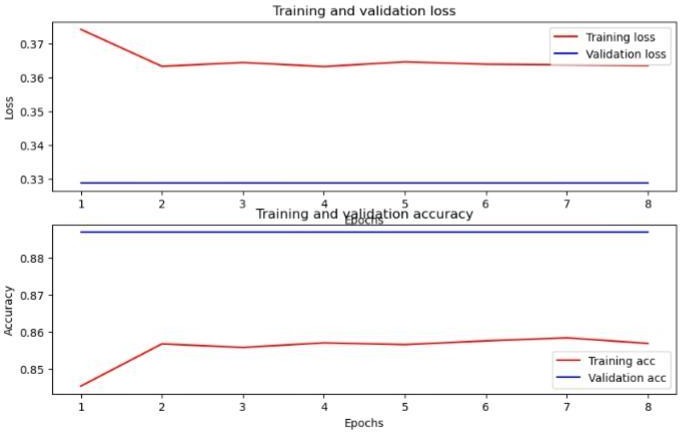
Plt.title(‘Training and validation accuracy’) Plt.xlabel(‘Epochs’)

Plt.ylabel(‘Accuracy’) Plt.legend(loc=’lower right’)

Dict\_keys([‘loss’, ‘binary\_accuracy’, ‘val\_loss’, ‘val\_binary\_accuracy’])

**Out[20]:**

<matplotlib.legend.Legend at 0x7c1f70136b90>



Fake news detection using the BERT (Bidirectional Encoder Representations from Transformers) method represents a significant advancement in the field of Natural

Language Processing (NLP). BERT, with its deep understanding of contextual language, has revolutionized the way we discern deceptive information from factual content.

.



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()

1. **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)

1. **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()

1. **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

**In[2]:**

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[3]:**

**In[4]:**

**Out[4]:**

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

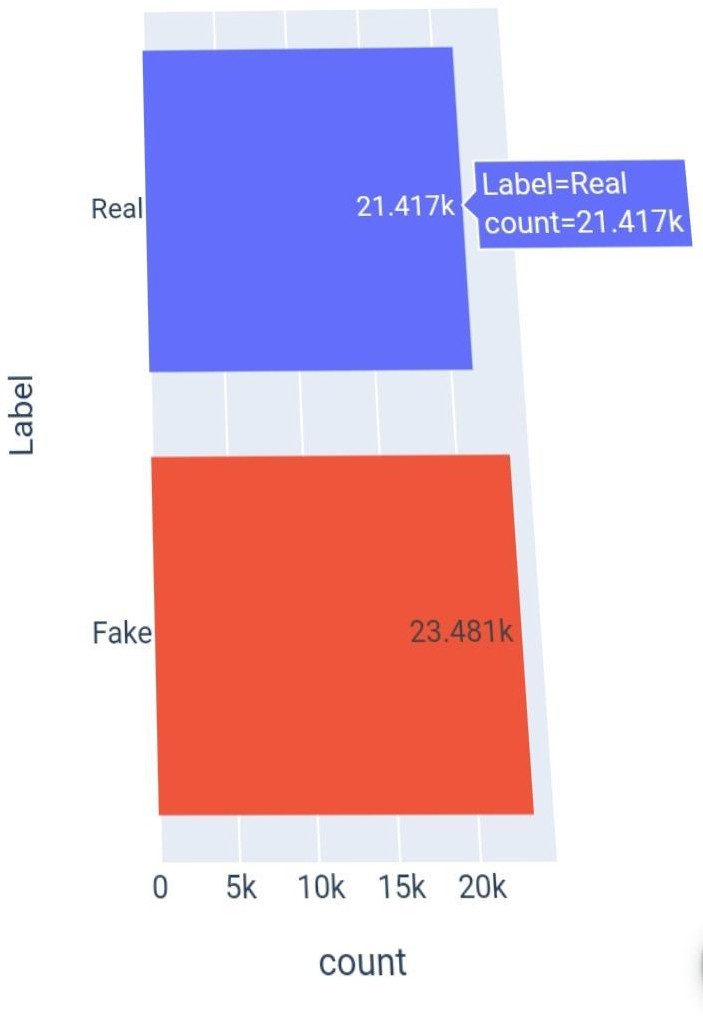
Real\_news\_path = “/kaggle/input/fake-and-real- news-dataset/True.csv”

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

Fake\_news.head(3)



### REAL VS FAKE NEWS DATA SET



**In [5]**:

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

Color=’subject’,

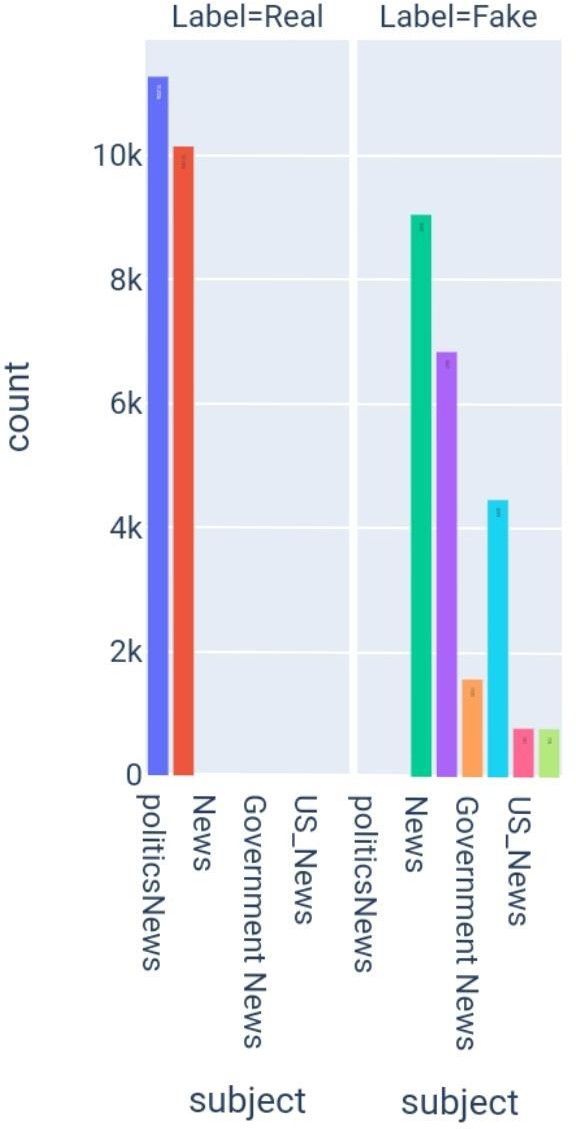
Title=’Fake vs Real news Subject

Distribution’,

Text\_auto=True, Facet\_col=’Label’)

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

FAKE VS REAL NEWS SUBJECT DISTRIBUTION



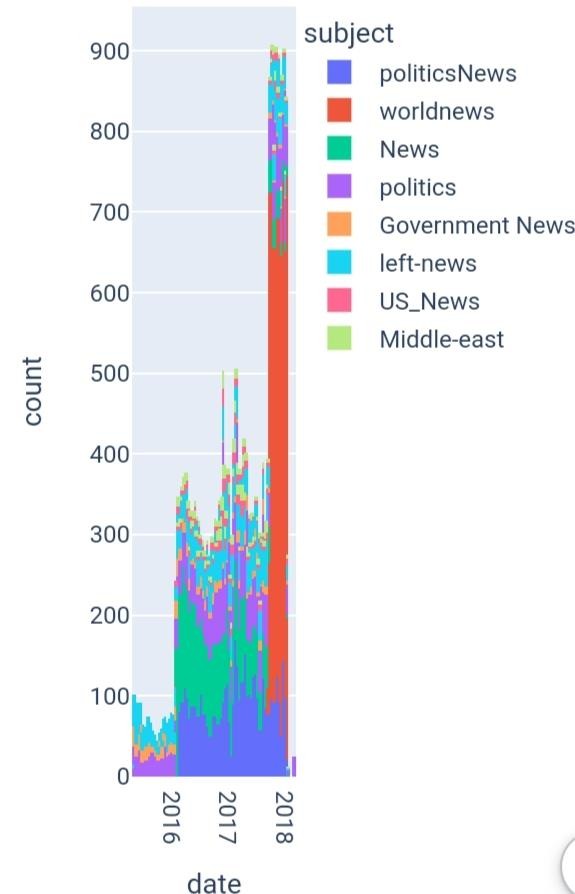
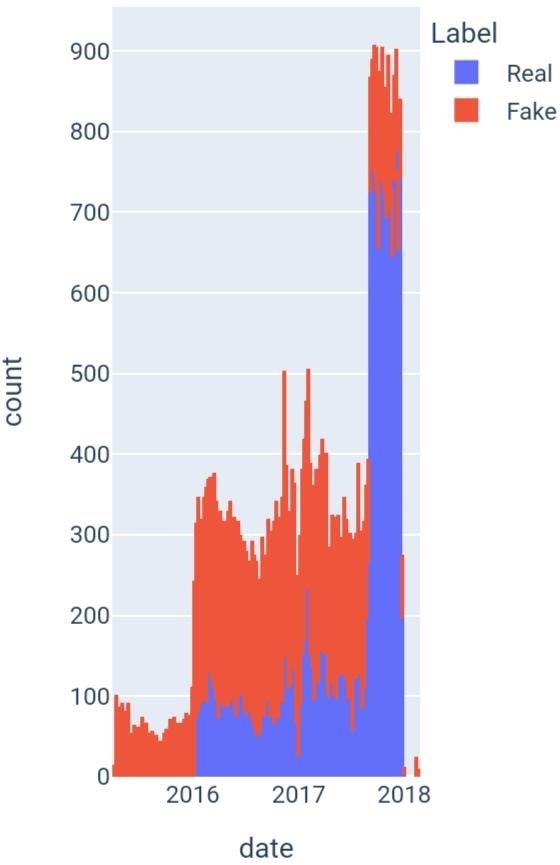
**In[6]:**

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.]:**



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.

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.

In the realm of Natural Language Processing (NLP), the task of fake news detection is of paramount importance in the age of information overload and misinformation. Detecting fake news relies on a multi-step process that encompasses text preprocessing, feature extraction, model training, and rigorous evaluation. This introductory guide provides an overview of these essential stages in a fake news detection project



# Text Preprocessing:

Text preprocessing serves as the foundation for effective NLP tasks. In the context of fake news detection, it involves cleaning and transforming raw text data to make it suitable for

machine learning models. Key preprocessing steps include:

# Data Collection:

Gathering a dataset that includes labeled examples of real and fake news articles.

# Text Cleaning:

Removing irrelevant information, HTML tags, special characters, and noise from the text.

# Tokenization:

Dividing the text into individual words or tokens to facilitate analysis.

# Lowercasing:

Ensuring uniformity by converting all text to lowercase.

# Stopword Removal:

Eliminating common, non- informative words like “the,” “and,” and “is.”

# Stemming or Lemmatization:

Reducing words to their root forms for normalization.

# Vectorization:

Converting textual data into numerical form using techniques like TF-IDF or Word Embeddings.

# Feature Extraction:

Feature extraction is the process of transforming textual data into structured representations that can be used by machine learning

models. Key considerations in this phase include:

# Feature Selection:

Choosing the most relevant features for your model, often based on metrics like chi-squared or information gain.

Word Embeddings: Leveraging pre- trained word embeddings to capture semantic information and relationships between words.

# N-grams:

Utilizing n-grams (groups of adjacent words) to capture contextual information.

# Topic Modeling:

Applying techniques like Latent Dirichlet Allocation (LDA) to identify latent topics within the text.

# Model Training and Evaluation:

The core of fake news detection lies in developing, training, and assessing machine learning models.

This stage includes the following components:

# Data Split:

Dividing the dataset into training and testing subsets to ensure model evaluation.

# Selecting Algorithms:

Choosing appropriate machine learning algorithms for text classification, including Multinomial Naïve Bayes, Logistic Regression,

Random Forest, or deep learning models like LSTM or Transformers.

# Model Training:

Training the chosen model on the training data, which involves learning the patterns in real and fake news articles.

# Model Evaluation:

Evaluating the model’s performance using critical metrics such as accuracy, precision, recall, F1-score, and ROC AUC. It also includes creating a confusion matrix to visualize model performance.

# Hyperparameter Tuning:

Fine-tuning the model’s hyperparameters to optimize its performance.

# Cross-validation:

Applying cross-validation techniques to ensure the model generalizes well to unseen data.

# Interpretability:

Exploring model interpretability techniques to understand the decision-making process of the model.

Let’s consider naive bayes algorithm

# FAKE NEWS WITH NAIVE BAYES USING NLP

**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 re Import string Import nltk

From nltk.corpus import stopwords From nltk.stem import PorterStemmer From nltk.tokenize import word\_tokenize Import matplotlib.pyplot as plt

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

From sklearn.feature\_extraction.text import TfidfVectorizer

From keras.layers import TextVectorization From keras.utils import pad\_sequences From xgboost import XGBClassifier

From scipy.sparse import hstack Import random

### DATA LOADING: In[2]:

Real\_df = pd.read\_csv(‘/kaggle/input/fake- and-real-news-dataset/True.csv’)

Fake\_df = pd.read\_csv(‘/kaggle/input/fake- and-real-news-dataset/Fake.csv’)

### In[3]:

Real\_df.info()

### In[4]:

Fake\_df.info()

### In[5]:

List(real\_df.sample(5).title)

### Out[5]:

[‘Renegade colonel surrenders in eastern Congo after clashes, seven dead’,

‘Factbox: Trump meetings include rapper Kanye West, Microsoft founder Bill Gates’,

“At under $5 each, Trump’s votes came cheap”,

“Israeli air strike hits near Syria’s Homs”,

“Trump calls storm over Russia hacking ‘political witchhunt’: NYT”]

### DATA VISUALISATION

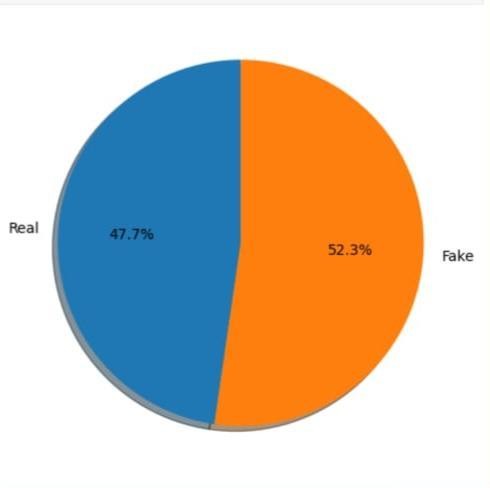
**IN[6]:**

fig = plt.figure(figsize=(5, 5)) labels = ‘Real’, ‘Fake’

sizes = [len(real\_df), len(fake\_df)]

plt.pie(sizes, labels=labels, autopct=’%1.1f%%’, Shadow=True, startangle=90)

plt.axis(‘equal’) plt.show()



### TITLES

We start by classifying the titles. First we put them in a numpy array as it is easier for us.

### In[7]:

Positive\_titles = np.array(real\_df.title) Negative\_titles = np.array(fake\_df.title)

### In[8]:

Def process\_news(news): “””Process news function. Input:

News: a string containing a news’ text or

title

Output:

Newss\_clean: a list of words containing the processed news’ text or title

“””

Stemmer = PorterStemmer() Stopwords\_english =

stopwords.words(‘english’)

# remove hyperlinks

News = re.sub(r’https?://[^\s\n\r]+’, ‘’, news)

# tokenize news

#tokenizer = word\_tokenize News\_tokens = word\_tokenize(news)

News\_clean = []

for word in news\_tokens:

if (word not in stopwords\_english and # remove stopwords

Word not in string.punctuation): # remove punctuation

Stem\_word = stemmer.stem(word) # stemming word

News\_clean.append(stem\_word) Return news\_clean

### In[9]:

rand\_id = random.randint(0,len(positive\_titles))

print(positive\_titles[rand\_id],process\_news(posit ive\_titles[rand\_id]))

### In[10]:

news\_title = np.concatenate((positive\_titles, negative\_titles), axis=0)

### In[11]:

positive\_y = np.array(real\_df.Fake\_news) Negative\_y = np.array(fake\_df.Fake\_news)

Y = np.concatenate((positive\_y, negative\_y), axis=0)

### In[12]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(news\_title, y, test\_size=0.20, random\_state=0)

### Building of the words frequency dictionary

This function builds a dictionary of the different words contained in the training set (title of text) and assign the number of times it was seen in a positive news or a negative news.

### In[13]:

Def build\_freqs(news, ys): “””Build frequencies.

Input:

News: a list of news title or texts

Ys: an m x 1 array with the real/fake label of each title/news

(either 0 or 1) Output:

Freqs: a dictionary mapping each (word, real/fake) pair to its

Frequency

“””

Freqs = {}

for y, new in zip(yslist, news):

for word in process\_news(new): Pair = (word, y)

if pair in freqs:

Freqs[pair] += 1 else:

Freqs[pair] = 1 return freqs

### Training of the model In[14]:

Freqs = build\_freqs(news\_title, y)

This function train the model and return the logprior and loglikelihood parameters that will allow us to make futur predictions. For each words this will assign a value “loglikelihood[word]” that will indicate if a word is more likely to be attributed to the fake label (>0) or the real label (<0). The log prior is a regularization parameter that takes a non 0

value if the ratio of fake and real news is not equal to 1.

### In[15]:

Def train\_naive\_bayes(freqs, train\_x, train\_y): ‘’’

Input:

Freqs: dictionary from (word, label) to how often the word appears

Train\_x: a list of news

Train\_y: a list of labels correponding to the news (0,1)

Output:

Logprior: the log prior. (equation 3 above) Loglikelihood: the log likelihood of you

Naïve bayes equation. (equation 6 above) ‘’’

Loglikelihood = {} Logprior = 0

Vocab = set([pair[0] for pair in freqs.keys()])

V = len(vocab)

N\_pos = N\_neg = 0

for pair in freqs.keys():

if pair[1] > 0: N\_pos+=freqs[pair]

else:

N\_neg += freqs[pair] D = len(train\_y)

D\_pos = np.sum((train\_y == 1) D\_neg = np.sum((train\_y == 0))

Logprior = np.log(D\_pos) – np.log(D\_neg) for word in vocab:

Freq\_pos = freqs.get((word,1),0) Freq\_neg = freqs.get((word,0),0) P\_w\_pos = (freq\_pos + 1)/(N\_pos +V) P\_w\_neg = (freq\_neg + 1)/(N\_neg +V)

Loglikelihood[word] = np.log(p\_w\_pos) – np.log(p\_w\_neg)

return logprior, loglikelihood

### In[16]:

Logprior, loglikelihood = train\_naive\_bayes(freqs, X\_train, y\_train)

### Prediction and accuracy of our model

**In[17]:**

Def naive\_bayes\_predict(news, logprior, loglikelihood):

‘’’

Input:

News: a string Logprior: a number

Loglikelihood: a dictionary of words mapping to numbers

Output:

P: the sum of all the logliklihoods of each word in the news (if found in the

dictionary) + logprior (a number)word\_l = process\_news(news)

# initialize probability to zero P = 0

# add the logprior P += logprior

for word in word\_l:

if word in loglikelihood:

P += loglikelihood[word] return p

To test our model we look at the difference between each predicted values of the test set and each assigned value of the test set.

### In[18]:

Def test\_naive\_bayes(test\_x, test\_y, logprior, loglikelihood, naïve\_bayes\_predict=naïve\_bayes\_predict):

“””

Input:

Test\_x: A list of news title/text Test\_y: the corresponding labels for

the list of news title/text Logprior: the logprior

Loglikelihood: a dictionary with the loglikelihoods for each word

Output:

Accuracy: (# of news title/text classified correctly)/(total # of news title/text)

“””

Accuracy = 0 # return this properly

y\_hats = []

for new in test\_x:

if naïve\_bayes\_predict(new, logprior, loglikelihood) > 0:

y\_hat\_i = 1 else:

y\_hat\_i = 0

y\_hats.append(y\_hat\_i) error = np.sum((y\_hats !=

test\_y))/len(test\_y) accuracy = 1 – error return accuracy

### In[19]:

Test\_naive\_bayes(X\_test, y\_test, logprior, loglikelihood)

### Out[19]:

0.9891982182628062

An accuracy score of almost 99% is very good for a model this simple and fast.

In this last cell you can look at the predicted value and assigned value of random news titles.

### In[20]:

Rand\_id = random.randint(0,len(X\_test))

Value\_predict = ‘Fake’ if naïve\_bayes\_predict(X\_test[rand\_id], logprior, loglikelihood) > 0 else ‘Real’

Assigned\_value = ‘Fake’ if y\_test[rand\_id] == 1 else ‘Real’

Print(X\_test[rand\_id], ‘Predicted: ‘, Value\_predict, ‘; Assigned value:’, assigned\_value)

COMEDIAN DAVE CHAPPELLE Stuns NY

Audience: SLAMS Hillary…Compares TRUMP To The Terminator:”Most Gangsta

Candidate Ever” Predicted: Fake ; Assigned value: Fake

CONCLUSION:

In the fight against fake news, Naïve Bayes serves as a fundamental tool that can provide a reliable initial assessment of the authenticity of news articles. However, given the evolving nature of misinformation and the ever-increasing volume of data, continuous research and innovation are required to stay ahead in the battle to ensure accurate and trustworthy information in our society. When using Naïve Bayes for fake news detection, it’s essential to view it as one piece of a broader strategy that may include more complex models and ethical considerations to address the multifaceted challenges presented by fake news in the digital age.