

SURYA GROUP OF INSTITUTIONS

NAAN MUDHALVAN

|  |
| --- |
| IBM ARTIFICAL INTELLIGENCE  PRADAP B  422221104026  TEAM - 07 |

Fake News Detection Using NLP

Fake news detection using Natural Language Processing (NLP) is a crucial application of AI and NLP techniques to combat the spread of misinformation. In this example, I'll provide a simplified Python program that uses NLP and machine learning to classify news articles as either real or fake. Note that real-world applications of fake news detection are more complex and require large datasets and more sophisticated models.

Here's a step-by-step guide and a basic Python program:



Step 1: Import

import pandas as pd

import re

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from wordcloud import WordCloud

import matplotlib.pyplot as plt

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

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, SimpleRNN, Dense

from sklearn.metrics import log\_loss, roc\_auc\_score, confusion\_matrix

import seaborn as sns

Step 2 : Import Dataset

true\_data = pd.read\_csv('/kaggle/input/fake-and-real-news-dataset/True.csv')

fake\_data = pd.read\_csv('/kaggle/input/fake-and-real-news-dataset/Fake.csv')

## Step 3 : Adding Truth Value Labels

# Add labels and merge the data

fake\_data['label'] = 'fake'

true\_data['label'] = 'true'

merged\_data = pd.concat([fake\_data, true\_data])

## Step 4 : EDA

true\_data.head()

fake\_data.head()  
merged\_data = merged\_data.sample(frac=1).reset\_index(drop=True)

merged\_data.head()

merged\_data.dtypes

# Calculate label distribution

label\_distribution = merged\_data['label'].value\_counts()

# Extracting labels and counts for pie chart

labels = [f"{label} ({count})" for label, count in zip(label\_distribution.index, label\_distribution.values)]

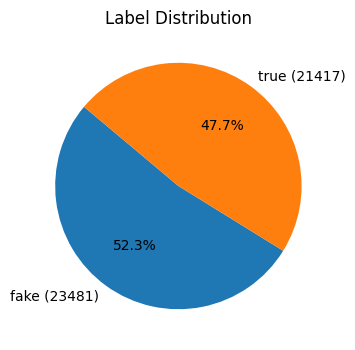
# Plotting the pie chart

plt.figure(figsize=(4, 4))

plt.pie(label\_distribution, labels=labels, autopct='%1.1f%%', startangle=140)

plt.title('Label Distribution')

plt.show()



## Step 5 : Preprocessing the Text

def preprocess\_text(text):

# Convert text to lowercase

text = text.lower()

# Remove punctuations

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

# Tokenize the text

words = word\_tokenize(text)

# Remove stopwords and words with length <= 2

stop\_words = set(stopwords.words('english'))

words = [word for word in words if word not in stop\_words and len(word) > 2]

# Remove repeated words

words = list(dict.fromkeys(words))

# Join the words back into text

text = ' '.join(words)

return text

Distribution :

# Calculate label distribution

label\_distribution = merged\_data['label'].value\_counts()

# Extracting labels and counts for pie chart

labels = [f"{label} ({count})" for label, count in zip(label\_distribution.index, label\_distribution.values)]

# Plotting the pie chart

plt.figure(figsize=(4, 4))

plt.pie(label\_distribution, labels=labels, autopct='%1.1f%%', startangle=140)

plt.title('Label Distribution')

plt.show()

## Step 6 :Checking Fake Political News and Fake News Buzzwords

fake\_politics\_data = ' '.join(merged\_data[(merged\_data['subject'] == 'politics') & (merged\_data['label'] == 'fake')]['clean\_text'])

total\_fake\_news = ' '.join(merged\_data[merged\_data['label'] == 'fake']['clean\_text'])

## fake\_politics\_data[0:500]

## total\_fake\_news[0:500]

## wordcloud = WordCloud(width=800, height=400).generate(fake\_politics\_data)

## plt.figure(figsize=(10, 5))

## plt.imshow(wordcloud, interpolation='bilinear')

## plt.axis('off')

## plt.title('Word Cloud for Fake Politics News')

## plt.show()

wordcloud = WordCloud(width=800, height=400).generate(total\_fake\_news)

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Word Cloud for Fake News')

plt.show()

## Step 7 : Splitting the Dataset

X = merged\_data['clean\_text']

y = merged\_data['label']

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

## Step 8: Performing Tokenization

# Tokenize text

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(X\_train)

X\_train\_tokens = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_tokens = tokenizer.texts\_to\_sequences(X\_test)

# print(f"Total tokens: {len(tokenizer.word\_index)}")

# Calculate total tokens

total\_tokens = sum([len(tokens) for tokens in X\_train\_tokens])

print("Total Tokens:", total\_tokens)

maxlen = 20

X\_train\_pad = pad\_sequences(X\_train\_tokens, maxlen=maxlen, padding='post')

X\_test\_pad = pad\_sequences(X\_test\_tokens, maxlen=maxlen, padding='post')

## Step 9 : RNN Model

# Build the RNN model

model = Sequential()

model.add(Embedding(input\_dim=len(tokenizer.word\_index) + 1, output\_dim=4, input\_length=maxlen))

model.add(SimpleRNN(units=128, return\_sequences=True))

model.add(SimpleRNN(units=64, return\_sequences=True))

model.add(SimpleRNN(units=32))

model.add(Dense(units=1, activation='sigmoid'))

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy', 'AUC'])

model.summary()