SURYA GROUP OF INSTITUTIONS

NAAN MUDHALVAN

IBM ARTIFICAL INTELLIGENCE

PRADAP B

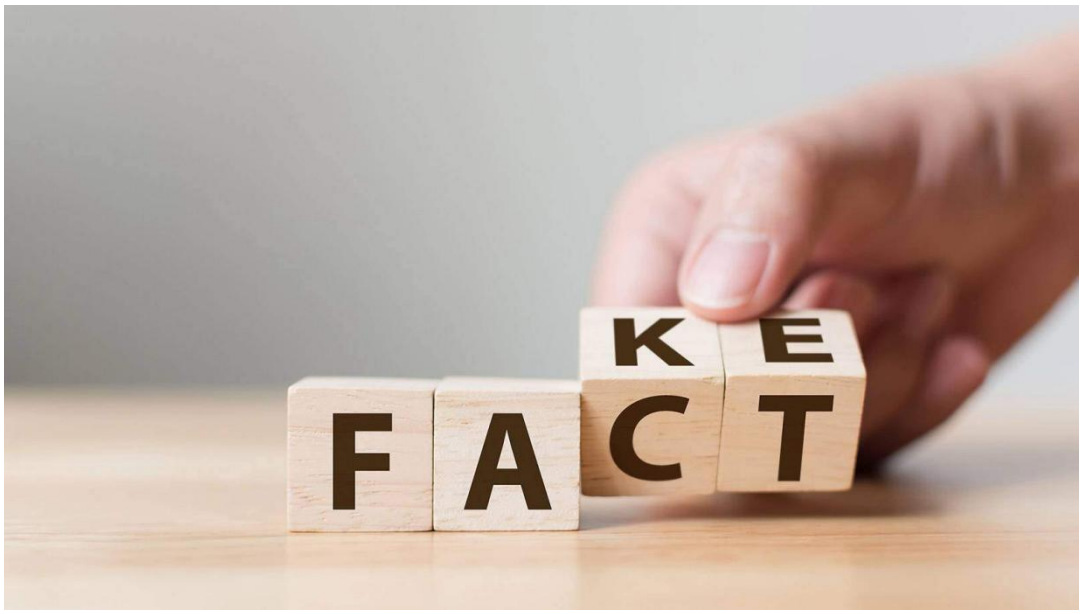
422221104026

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')
```

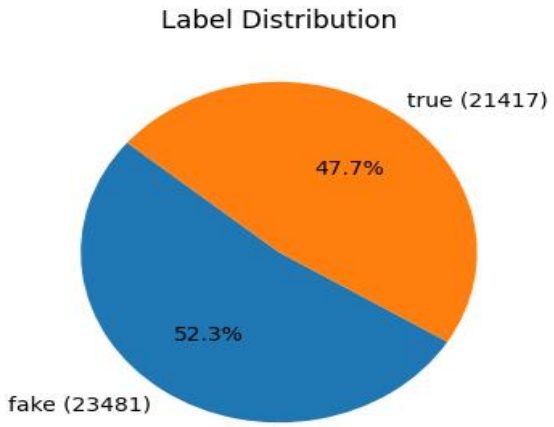
Step 3 : *Adding Truth Value Labels*

```
# 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(dro
p=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()
```