



SURYA GROUP OF INSTITUTIONS

NAAN MUDHALVAN

IBM ARTIFICIAL INTELLIGENCE

GOKULAKRISHNAN A

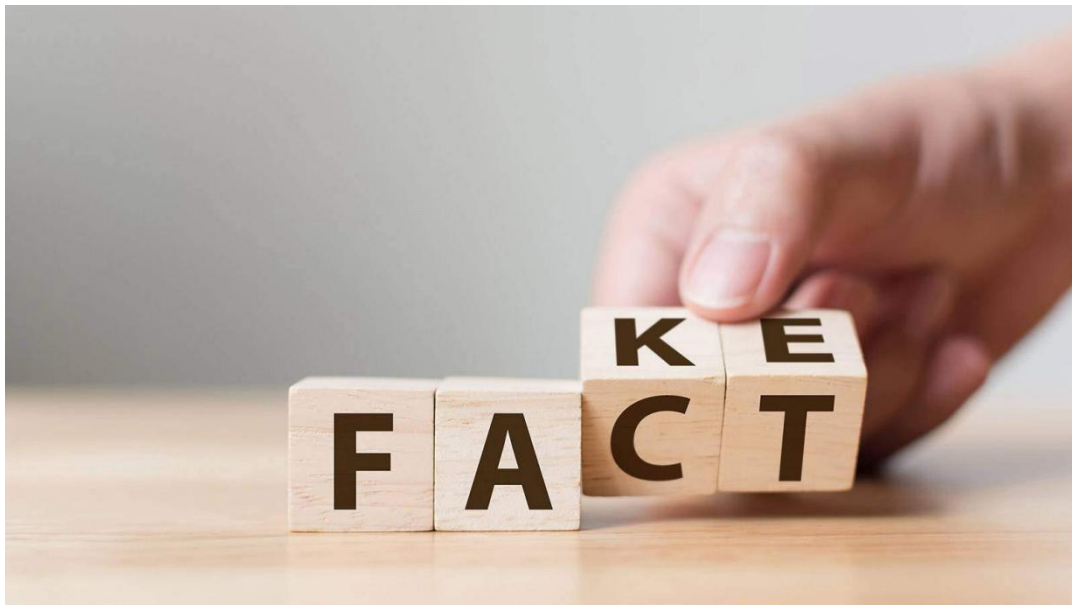
422221104013

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

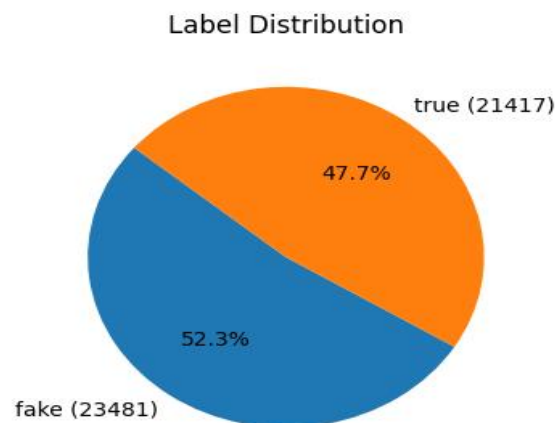
```
true_data.head()
fake_data.head()
merged_data =
merged_data.sample(frac=1).reset_index(drop=True)
merged_data.head()
merged_data.dtypes
```

```
# 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()
```

Output:



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

```

```

merged_data['clean_text'] = merged_data['text'].apply(preprocess_text)
merged_data.clean_text

```

Output:

```

0    united nations reuters general assembly wednes...
1    james keefe released blockbuster undercover vi...
2    gina loudon went cnn today discuss fallacious ...
3    must watch videohttpsyoutube5zjj2z4bu
4    washington reuters senate thursday passed legi...
...
44893 senator john mccain got outed hypocrite damnin...
44894 washington reuters senate democratic leader ch...
44895 think like knowwatch susan rice insists leaked...
44896 donald trump recently came proposal new tax pl...
44897 paris reuters french president emmanuel macron...
Name: clean_text, Length: 44898, dtype: object

```

Distribution :

```

# Calculate label distribution

```

```

label_distribution = merged_data['label'].value_counts()

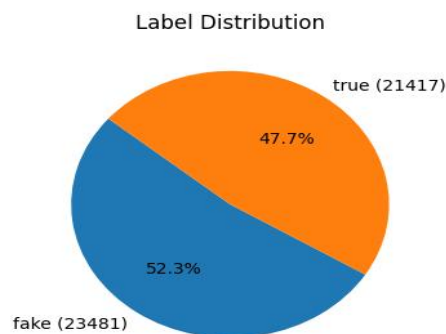
# Extracting labels and counts for pie chart
labels = [f' {label} ( {count})' for label, count in
zip(label_distribution.index, label_distribution.values)]

# 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()

```

Output:



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]`

Output:

James Keefe released blockbuster undercover video yesterday saying going commit acts terror trump supporters attend deploraball washington seen hereone organizers mike chernovich conservative author activist active twitter excerpt admit committing act domestic terrorism buy tickets overt criminal conspiracy definitely picked wrong group try terrorize jeff sessions gon attorney general new department justice thought dealing obama would let say people cares well charge anymore eric holder loretta.

`total_fake_news[0:500]`

Output:

James Keefe released blockbuster undercover video yesterday saying going commit acts terror trump supporters attend deploraball washington seen hereone organizers mike chernovich conservative author activist active twitter excerpt admit committing act domestic terrorism buy tickets overt criminal conspiracy definitely picked wrong group try terrorize jeff sessions gon attorney general new department justice thought dealing obama would let say people cares well charge anymore eric holder loretta.

```
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')
```

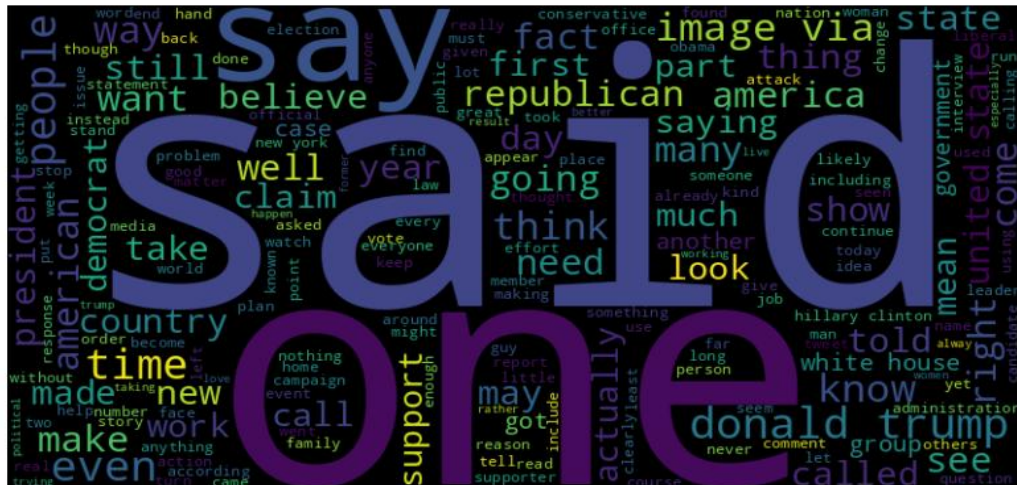
```
plt.show()
```

Output:



plt.show()

Output:



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)
```

Output:

Total Tokens: 5831432

```
# Apply post padding
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()

```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---------------------------|-----------------|---------|
| embedding (Embedding) | (None, 20, 4) | 810784 |
| simple_rnn (SimpleRNN) | (None, 20, 128) | 17024 |
| simple_rnn_1 (SimpleRNN) | (None, 20, 64) | 12352 |
| simple_rnn_2 (SimpleRNN) | (None, 32) | 3104 |
| dense (Dense) | (None, 1) | 33 |
| Total params: 843,297 | | |
| Trainable params: 843,297 | | |
| Non-trainable params: 0 | | |

```

# Train the model

```

```

model.fit(X_train_pad, y_train.map({'fake': 1, 'true': 0}), epochs=20,
validation_data=(X_test_pad, y_test.map({'fake': 1, 'true': 0})))

```

Output:

Epoch 1/20

1123/1123 [=====] - 29s 23ms/step - loss: 0.2698 - accuracy: 0.8684 - auc: 0.9525 - val_loss: 0.1548 - val_accuracy: 0.9415 - val_auc: 0.9849

Epoch 2/20

1123/1123 [=====] - 26s 23ms/step - loss: 0.0796 - accuracy: 0.9729 - auc: 0.9951 - val_loss: 0.1550 - val_accuracy: 0.9457 - val_auc: 0.9862

Epoch 3/20

1123/1123 [=====] - 26s 23ms/step - loss: 0.0327 - accuracy: 0.9901 - auc: 0.9986 - val_loss: 0.1744 - val_accuracy: 0.9508 - val_auc: 0.9841

Epoch 4/20

1123/1123 [=====] - 25s 22ms/step - loss: 0.0168 - accuracy: 0.9948 - auc: 0.9994 - val_loss: 0.1844 - val_accuracy: 0.9497 - val_auc: 0.9822

Epoch 5/20

1123/1123 [=====] - 26s 23ms/step - loss: 0.0102 - accuracy: 0.9967 - auc: 0.9997 - val_loss: 0.2891 - val_accuracy: 0.9416 - val_auc: 0.9675

Epoch 6/20

1123/1123 [=====] - 26s 23ms/step - loss: 0.0074 - accuracy: 0.9979 - auc: 0.9997 - val_loss: 0.2298 - val_accuracy: 0.9516 - val_auc: 0.9756

Epoch 7/20

1123/1123 [=====] - 25s 22ms/step - loss: 0.0057 - accuracy: 0.9981 - auc: 0.9998 - val_loss: 0.2585 - val_accuracy: 0.9506 - val_auc: 0.9722

Epoch 8/20

1123/1123 [=====] - 25s 22ms/step - loss: 0.0061 - accuracy: 0.9982 - auc: 0.9998 - val_loss: 0.2368 - val_accuracy: 0.9537 - val_auc: 0.9755

Epoch 9/20

1123/1123 [=====] - 25s 22ms/step - loss: 0.0061 - accuracy: 0.9981 - auc: 0.9998 - val_loss: 0.2821 - val_accuracy: 0.9491 - val_auc: 0.9705

Epoch 10/20

1123/1123 [=====] - 26s 23ms/step - loss: 0.0031 - accuracy: 0.9992 - auc: 0.9999 - val_loss: 0.2865 - val_accuracy: 0.9518 - val_auc: 0.9704

Epoch 11/20

1123/1123 [=====] - 25s 22ms/step - loss: 0.0044 - accuracy: 0.9986 - auc: 0.9999 - val_loss: 0.3036 - val_accuracy: 0.9483 - val_auc: 0.9681

Epoch 12/20

1123/1123 [=====] - 25s 22ms/step - loss: 0.0049 - accuracy: 0.9986 - auc: 0.9998 - val_loss: 0.4864 - val_accuracy: 0.9156 - val_auc: 0.9427

Epoch 13/20

1123/1123 [=====] - 25s 22ms/step - loss: 0.0037 - accuracy: 0.9989 - auc: 0.9999 - val_loss: 0.2758 - val_accuracy: 0.9531 - val_auc: 0.9708

Epoch 14/20

1123/1123 [=====] - 26s 23ms/step - loss: 0.0020 - accuracy: 0.9996 - auc: 0.9999 - val_loss: 0.3219 - val_accuracy: 0.9496 - val_auc: 0.9671

Epoch 15/20

1123/1123 [=====] - 26s 23ms/step - loss: 4.6594e-04 - accuracy: 0.9999 - auc: 1.0000 - val_loss: 0.3571 - val_accuracy: 0.9491 - val_auc: 0.9634

Epoch 16/20

1123/1123 [=====] - 25s 22ms/step - loss: 3.2795e-04 - accuracy: 1.0000 - auc: 1.0000 - val_loss: 0.3868 - val_accuracy: 0.9480 - val_auc: 0.9610

Epoch 17/20

1123/1123 [=====] - 25s 22ms/step - loss: 3.2790e-04 - accuracy: 1.0000 - auc: 1.0000 - val_loss: 0.3979 - val_accuracy: 0.9480 - val_auc: 0.9595

Epoch 18/20

```
1123/1123 [=====] - 26s 23ms/step - loss: 3.2813e-04 - accurac
y: 1.0000 - auc: 1.0000 - val_loss: 0.4101 - val_accuracy: 0.9469 - val_auc: 0.9588
Epoch 19/20
1123/1123 [=====] - 25s 22ms/step - loss: 3.2916e-04 - accurac
y: 1.0000 - auc: 1.0000 - val_loss: 0.4161 - val_accuracy: 0.9464 - val_auc: 0.9580
Epoch 20/20
1123/1123 [=====] - 25s 22ms/step - loss: 3.2679e-04 - accurac
y: 1.0000 - auc: 1.0000 - val_loss: 0.4322 - val_accuracy: 0.9463 - val_auc: 0.9562
```

Step 10 : Making Predictions

```
train_preds = model.predict(X_train_pad)
```

```
test_preds = model.predict(X_test_pad)
```

Output:

```
1123/1123 [=====] - 7s 6ms/step
281/281 [=====] - 2s 6ms/step
```

Step 11 : Examining Results

```
# Calculate log loss, ROC-AUC score, and confusion matrix for
training set
```

```
train_loss = log_loss(y_train.map({'fake': 1, 'true': 0}), train_preds)
```

```
train_auc = roc_auc_score(y_train.map({'fake': 1, 'true': 0}),
train_preds)
```

```
train_confusion = confusion_matrix(y_train.map({'fake': 1, 'true': 0}),
train_preds > 0.5)
```

```
# Calculate log loss, ROC-AUC score, and confusion matrix for
testing set
```

```
test_loss = log_loss(y_test.map({'fake': 1, 'true': 0}), test_preds)
```

```
test_auc = roc_auc_score(y_test.map({'fake': 1, 'true': 0}), test_preds)
test_confusion = confusion_matrix(y_test.map({'fake': 1, 'true': 0}),
test_preds > 0.5)

# print("Training Log Loss:", train_loss)
# print("Training ROC-AUC Score:", train_auc)
# print("Training Confusion Matrix:")
# print(train_confusion)
# print("Testing Log Loss:", test_loss)
# print("Testing ROC-AUC Score:", test_auc)
# print("Testing Confusion Matrix:")
# print(test_confusion)
```

from tabulate import tabulate

```
# ... Your previous code for metrics ...
```

```
# Tabulate metrics
```

```
table_data = [
    ["Training Log Loss", train_loss],
    ["Training ROC-AUC Score", train_auc],
#    ["Training Confusion Matrix", train_confusion],
    ["Testing Log Loss", test_loss],
    ["Testing ROC-AUC Score", test_auc],
#    ["Testing Confusion Matrix", test_confusion]
```

```
]
# Print metrics as a table
print(tabulate(table_data, headers=["Metric", "Value"],
tablefmt="pretty"))
```

Output:

| Metric | Value |
|------------------------|-----------------------|
| Training Log Loss | 0.0003151029772055638 |
| Training ROC-AUC Score | 0.9999877151892584 |
| Testing Log Loss | 0.43219265655119493 |
| Testing ROC-AUC Score | 0.9857079578396398 |

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
```

```
# Plot Training Confusion Matrix
```

```
sns.heatmap(train_confusion, annot=True, fmt='d', cmap='Blues',
ax=axes[0])
```

```
axes[0].set_title('Training Confusion Matrix')
```

```
# Plot Testing Confusion Matrix
```

```
sns.heatmap(test_confusion, annot=True, fmt='d', cmap='Blues',
ax=axes[1])
```

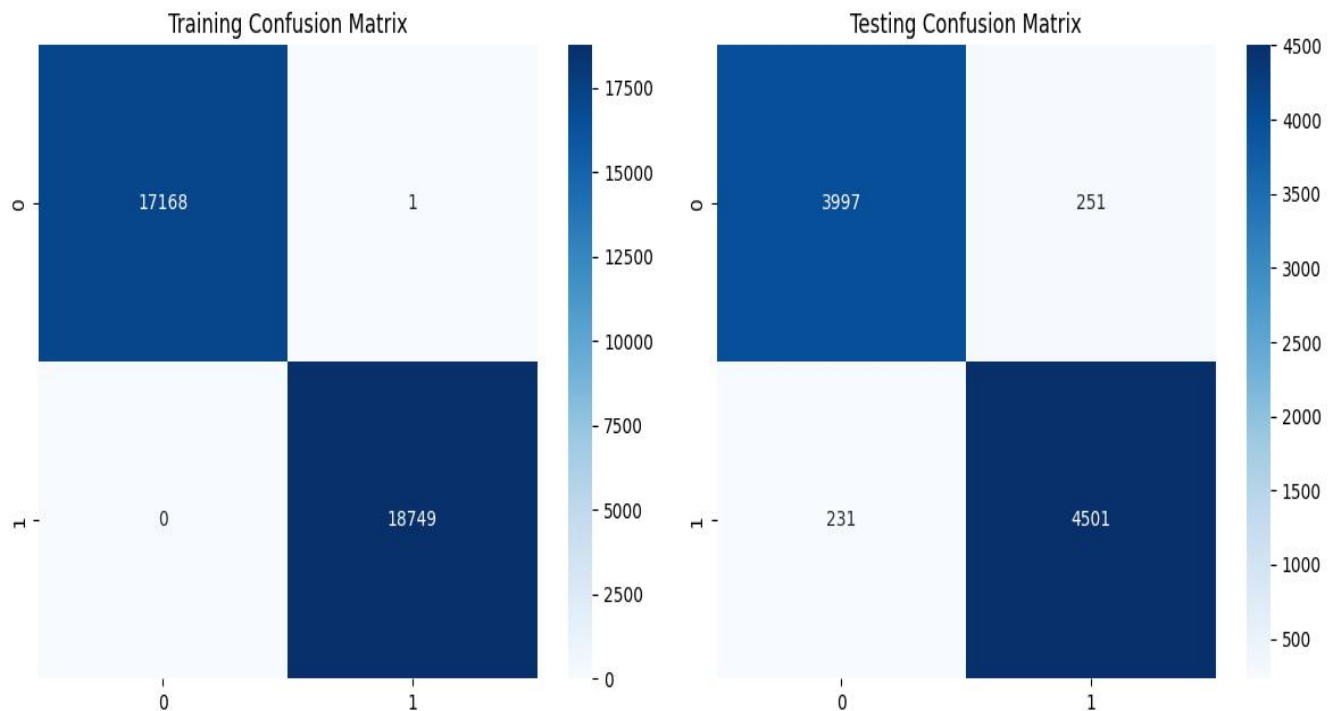
```
axes[1].set_title('Testing Confusion Matrix')
```

```
# Adjust layout
```

```
plt.tight_layout()
```

```
plt.show()
```

Output:



Conclusion:

We have classified our news data using three classification models. We have analysed the performance of the models using accuracy and confusion matrix. But this is only a beginning point for the problem. There are advanced techniques like BERT, GloVe and ELMo which are popularly used in the field of NLP. If you are interested in NLP, you can work forward with these techniques.



