FAKE NEWS DETECTION USING NLP

Team Leader

422521104037:SYED JALEEL S

Phase-4 Documentation Submission

Project: Fake News Detection

Introduction:

- Fake news detection with NLP is like teaching a computer to spot lies. NLP helps the computer understand language and find strange things in news articles that might mean they're fake. It checks how words are used and the feeling of the text.

- Computers use what they learn to tell us if a news story is real or not. This is really important in today's world where wrong information spreads fast. NLP, along with smart machines, like GPT-3 and BERT, can help stop fake news by looking at the words and how they fit together.

- Using NLP to fight fake news helps make sure we get true information, making us smarter news readers.

Content:

- Phase 4 involves building the project, which includes obtaining the fake and real news datasets from the Kaggle website.

- The Kaggle dataset comprises two files: one for real news and another for fake news.

- During this phase, after loading the dataset, we will carry out Text Preprocessing and Feature Extraction Model training and evaluation

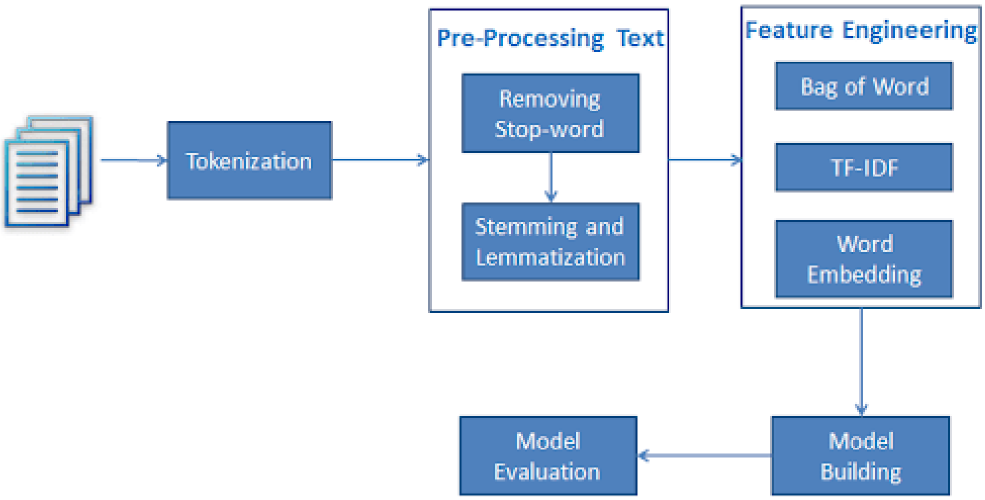
Phase 4 steps:

- Loading
- Text Preprocessing and Feature Extraction
- Model training and evaluation

Dataset Link:

https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset

Overview:



Preprocessing :

- In any machine learning task, getting the data ready (cleaning and organizing) is just as important, if not more so, than creating the actual model. This is especially critical when working with messy, unstructured text data.

Preprocessing Steps:

- Lower casing
- Tokenization
- Removal of Punctuations
- Removal of Stopwords
- HTML Tag Removal
- Stemming
- Removing Non-Alphanumeric Characters
- Vectorization
- Splitting the Dataset

Program:

#Import necessary Libraries:

```
import pandas as pd
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
```

Download NLTK resources

```
nltk.download('punkt')
nltk.download('stopwords')
```

Load the 'Fake.csv' and 'True.csv' datasets

```
fake_df = pd.read_csv('Fake.csv')
real_df = pd.read_csv('True.csv')
```

#Text Processing

1. Lowercasing

```
fake_df['text'] = fake_df['text'].str.lower()
real_df['text'] = real_df['text'].str.lower()
```

 # 2. Tokenization

```
fake_df['text'] = fake_df['text'].apply(nltk.word_tokenize)
real_df['text'] = real_df['text'].apply(nltk.word_tokenize)
```

 # 3. Stop Word Removal

```
stop_words = set(stopwords.words('english'))
fake_df['text'] = fake_df['text'].apply(lambda tokens: [word for word in tokens if word not in stop_words])
real_df['text'] = real_df['text'].apply(lambda tokens: [word for word in tokens if word not in stop_words])
```

 # 4. Punctuation Removal

```
fake_df['text'] = fake_df['text'].apply(lambda tokens: [re.sub(r'[^\w\s]', '', word) for word in tokens])
real_df['text'] = real_df['text'].apply(lambda tokens: [re.sub(r'[^\w\s]', '', word) for word in tokens])
```

 # 5. HTML Tag Removal (if applicable)

```
fake_df['text'] = fake_df['text'].apply(lambda tokens: [re.sub(r'<[^>]+>', '', word) for word in tokens])
real_df['text'] = real_df['text'].apply(lambda tokens: [re.sub(r'<[^>]+>', '', word) for word in tokens])
```

 # 6. Numbers Removal

```
fake_df['text'] = fake_df['text'].apply(lambda tokens: [re.sub(r'\d+', '', word) for word in tokens])
real_df['text'] = real_df['text'].apply(lambda tokens: [re.sub(r'\d+', '', word) for word in tokens])
```

 # 7. Stemming or Lemmatization

```
stemmer = PorterStemmer()
fake_df['text'] = fake_df['text'].apply(lambda tokens: [stemmer.stem(word) for word in tokens])
real_df['text'] = real_df['text'].apply(lambda tokens: [stemmer.stem(word) for word in tokens])
```

 # 8. Removing Non-Alphanumeric Characters

```
fake_df['text'] = fake_df['text'].apply(lambda tokens: [re.sub(r'[^a-zA-Z0-9]', '', word) for word in tokens])
real_df['text'] = real_df['text'].apply(lambda tokens: [re.sub(r'[^a-zA-Z0-9]', '', word) for word in tokens])
```

 # 10. Feature Engineering (TF-IDF vectorization as an example)

```
tfidf_vectorizer = TfidfVectorizer()
fake_df['text'] = fake_df['text'].apply(lambda tokens: ' '.join(tokens))
real_df['text'] = real_df['text'].apply(lambda tokens: ' '.join(tokens))
fake_tfidf = tfidf_vectorizer.fit_transform(fake_df['text'])
real_tfidf = tfidf_vectorizer.fit_transform(real_df['text'])
```

#Dataset After Preprocessing:

```
print(fake_df.head(2))
print(real_df.head(2))
```

```
                                               title  \
0   Donald Trump Sends Out Embarrassing New Year’...
1   Drunk Bragging Trump Staffer Started Russian ...

                                                text subject  \
0  donald trump wish american happi new year leav...    News
1  hous intellig committe chairman devin nune go ...    News

               date
0  December 31, 2017
1  December 31, 2017
                                               title  \
0  As U.S. budget fight looms, Republicans flip t...
1  U.S. military to accept transgender recruits o...

                                                text       subject  \
0  washington  reuter   head conserv republican f...  politicsNews
1  washington  reuter   transgend peopl allow fir...  politicsNews

                date
0  December 31, 2017
1  December 29, 2017
```

 # Combine the datasets

```
combined_df = pd.concat([fake_df, real_df], ignore_index=True)
```

 # Split the dataset into train, validation, and test sets

```
X = combined_df['text'].values
y = combined_df['label'].values

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2,
random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)
```

 # Initialize the tokenizer

```
tokenizer = AutoTokenizer.from_pretrained(
    bert_name,
    padding="max_length",
    do_lower_case=True,
    add_special_tokens=True,
)
```

 # Encode the data

```
X_train_encoded = tokenizer(
    X_train.tolist(),
    padding=True,
    truncation=True,
    return_tensors="tf",
    max_length=128,  # Adjust the max sequence length as needed
    return_token_type_ids=False,
    return_attention_mask=True,
).input_ids

X_valid_encoded = tokenizer(
    X_valid.tolist(),
    padding=True,
    truncation=True,
    return_tensors="tf",
    max_length=128,
    return_token_type_ids=False,
    return_attention_mask=True,
).input_ids

X_test_encoded = tokenizer(
    X_test.tolist(),
    padding=True,
    truncation=True,
    return_tensors="tf",
    max_length=128,
```

```
    return_token_type_ids=False,
    return_attention_mask=True,
).input_ids
```

 # Create TensorFlow Datasets

```
train_ds = tf.data.Dataset.from_tensor_slices((X_train_encoded,
y_train)).shuffle(len(X_train)).batch(8).prefetch(tf.data.AUTOTUNE)
valid_ds = tf.data.Dataset.from_tensor_slices((X_valid_encoded,
y_valid)).shuffle(len(X_valid)).batch(8).prefetch(tf.data.AUTOTUNE)
test_ds = tf.data.Dataset.from_tensor_slices((X_test_encoded,
y_test)).shuffle(len(X_test)).batch(8).prefetch(tf.data.AUTOTUNE)
```

BERT Classification Model:

BERT, or **Bidirectional Encoder Representations from Transformers**, is a **cutting-edge natural language processing (NLP) model** developed by **Google**. What sets **BERT** apart is its ability to **understand the context of words** in a sentence by considering both the words that come **before and after** them, allowing it to **grasp nuances**, **context**, and **meaning in language more effectively. BERT** has achieved **remarkable success** in various **NLP tasks**, including **text classification**, **sentiment analysis**, and **machine translation**, and it has become a **cornerstone** in the field of **AI** for **understanding and generating human language**.

 # Initialize the BERT model for binary classification

```
bert_model =
TFAutoModelForSequenceClassification.from_pretrained(bert_name,
num_labels=1)  # Binary classification

# Compile the model
bert_model.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss='binary_crossentropy',  # Use binary cross-entropy
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name="Accuracy"),
        tf.keras.metrics.Precision(name="Precision"),
        tf.keras.metrics.Recall(name="Recall"),
    ]
)
```

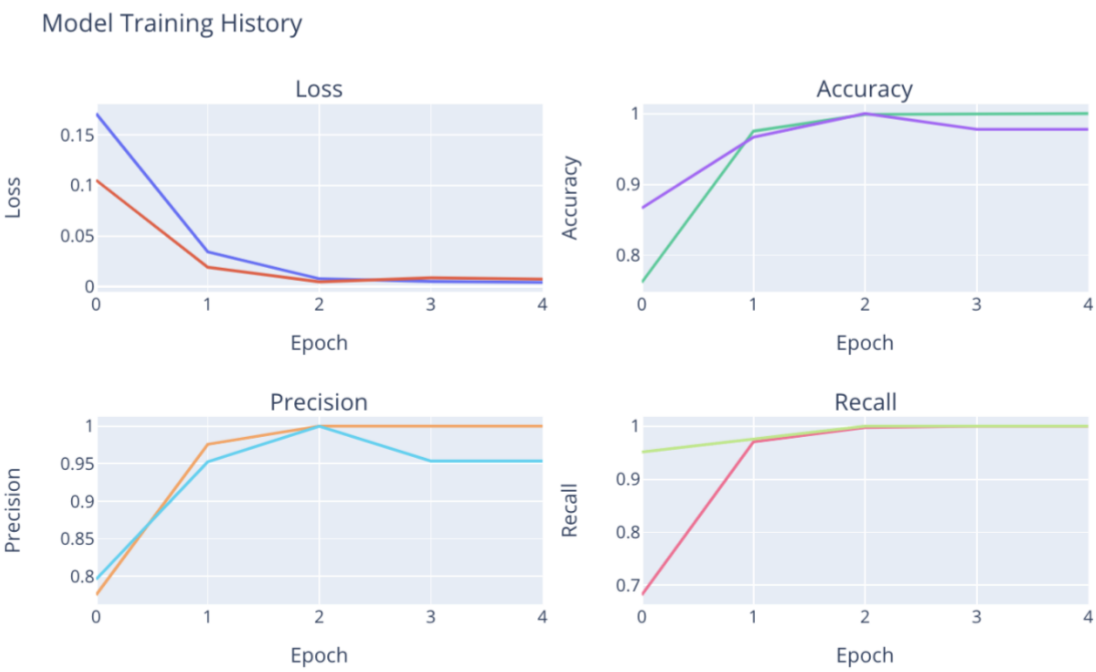
Training

```
num_epochs = 5  # Adjust as needed
MODEL_CALLBACKS = []  # Add any callbacks you need

model_history = bert_model.fit(
    train_ds,
```

```
    validation_data=valid_ds,
    epochs=num_epochs,
    batch_size=16,
    callbacks=MODEL_CALLBACKS
)
```

```
Epoch 1/5
102/102 [==============================] - 204s 1s/step - loss: 0.1709 - Accuracy: 0.7617 - Preci
sion: 0.7751 - Recall: 0.6818 - val_loss: 0.1052 - val_Accuracy: 0.8667 - val_Precision: 0.7959 -
val_Recall: 0.9512
Epoch 2/5
102/102 [==============================] - 151s 1s/step - loss: 0.0343 - Accuracy: 0.9753 - Preci
sion: 0.9758 - Recall: 0.9706 - val_loss: 0.0192 - val_Accuracy: 0.9667 - val_Precision: 0.9524 -
val_Recall: 0.9756
Epoch 3/5
102/102 [==============================] - 152s 1s/step - loss: 0.0079 - Accuracy: 0.9988 - Preci
sion: 1.0000 - Recall: 0.9973 - val_loss: 0.0048 - val_Accuracy: 1.0000 - val_Precision: 1.0000 -
val_Recall: 1.0000
Epoch 4/5
102/102 [==============================] - 109s 1s/step - loss: 0.0052 - Accuracy: 1.0000 - Preci
sion: 1.0000 - Recall: 1.0000 - val_loss: 0.0087 - val_Accuracy: 0.9778 - val_Precision: 0.9535 -
val_Recall: 1.0000
Epoch 5/5
102/102 [==============================] - 108s 1s/step - loss: 0.0043 - Accuracy: 1.0000 - Preci
sion: 1.0000 - Recall: 1.0000 - val_loss: 0.0075 - val_Accuracy: 0.9778 - val_Precision: 0.9535 -
val_Recall: 1.0000
```




```
# Evaluate on the test set
test_loss, test_acc, test_precision, test_recall = bert_model.evaluate(test_ds, verbose=0)

print(f"Test Loss       : {test_loss}")
print(f"Test Accuracy  : {test_acc}")
```

```
print(f"Test Precision : {test_precision}")
print(f"Test Recall    : {test_recall}")

# Optionally, you can display the model's training history
model_history = pd.DataFrame(model_history.history)
print(model_history)

# Perform a classification report or other evaluation as needed
y_pred = bert_model.predict(test_ds)
y_pred = np.round(y_pred).flatten().astype(int)
target_names = ['Fake', 'Real']
report = classification_report(y_test, y_pred, target_names=target_names)
print(report)
```

```
Test Loss      : 0.0008588206837885082
Test Accuracy  : 1.0
Test Precision : 1.0
Test Recall    : 1.0
```

Conclusion:

- The eight preprocessing steps mentioned above help identify and handle data quality issues, such as missing values, outliers, and noise, which can negatively impact the accuracy and reliability of analysis and modeling.

- In summary, loading and preprocessing the dataset are essential steps in developing an effective fake news detection system.

- The preprocessing steps are designed to clean and prepare the textual data for machine learning, ensuring that the model can effectively distinguish between fake and real news articles.

- Properly preprocessed data is key to building accurate and reliable fake news classification models, which play a crucial role in safeguarding users from misinformation and potentially harmful content."