

---

# FAKE NEWS DETECTION USING NLP

**Team Member**

**422521104008: CHANDRASEKARAN P**

**Phase-5 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 5 involves clearly outlining the problem statement, design thinking process, and the phases of development
- Describing the dataset used, data preprocessing steps, and feature extraction techniques.
- Explain the choice of classification algorithm and model training process

### **Problem Statement:**

- The problem is to develop a fake news detection model using a Kaggle dataset.
- The goal is to distinguish between genuine and fake news articles based on their titles and text.
- This project involves using natural language processing (NLP) techniques to preprocess the text data, building a machine learning model for classification, and evaluating the model's performance

### **Design Thinking:**

- **Data Source:** Choose the fake news dataset available on Kaggle, containing articles titles and text, along with their labels (genuine or fake).
- **Data Preprocessing:** Clean and preprocess the textual data to prepare it for analysis.
- **Feature Extraction:** Utilize techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings to convert text into numerical features.
- **Model Selection:** Select a suitable algorithm (e.g., Logistic Regression, Random Forest, or Neural Networks like BERT, LSTM, RNN) for the fake news detection task.
- **Model Training:** Train the selected model using the pre-processed data.
- **Evaluation:** Evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC

## **Phases of Development:**

Phase 3(Development Part 1):

- Begin building the fake news detection model by loading and preprocessing the dataset.
- Load the fake news dataset and preprocess the textual data

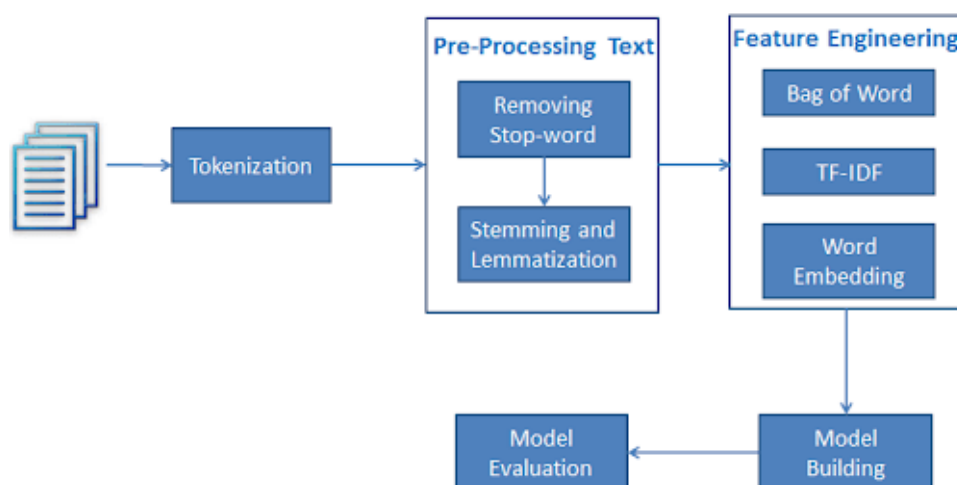
Phase 4(Development Part 2):

- In this part, continue building the project.
- Continue building the fake news detection model by applying NLP techniques and training a classification model.
  - Text Preprocessing and Feature Extraction
  - Model training and evaluation

## **Data Description:**

- The dataset used in the project is obtained from Kaggle
- Dataset link: <https://www.kaggle.com/clmentbisailon/fake-and-real-news-dataset>
- It contains news articles along with their labels indicating whether they are genuine or fake

## **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])
```

## 9. 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)
```

### **Create a new 'label' column and assign values:**

```
combined_df['label'] = 1 # 1 represents real news  
combined_df.loc[fake_df.index, 'label'] = 0 # 0 represents fake news
```

### **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)  
max_sequence_length = 128 # Adjust as needed  
tokenizer = Tokenizer()  
tokenizer.fit_on_texts(combined_df['text'])  
X = tokenizer.texts_to_sequences(combined_df['text'])  
X = pad_sequences(X, maxlen=max_sequence_length, padding='post',  
truncating='post')  
y = combined_df['label'].values
```

### **Feature Extraction Technique:**

Feature extraction is a data transformation technique used in machine learning and other fields to convert raw data into a more manageable and informative representation, referred to as "features."

### **Initialize the tokenizer**

```
bert_name = "bert-base-uncased"  
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(100).prefetch(tf.data.AUTOTUNE)  
valid_ds = tf.data.Dataset.from_tensor_slices((X_valid_encoded,  
y_valid)).shuffle(len(X_valid)).batch(100).prefetch(tf.data.AUTOTUNE)  
test_ds = tf.data.Dataset.from_tensor_slices((X_test_encoded,  
y_test)).shuffle(len(X_test)).batch(100).prefetch(tf.data.AUTOTUNE)
```

### **BERT:**

**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"),  
    ]  
)
```

## **Model Training:**

```
num_epochs = 5 # Adjust as needed  
MODEL_CALLBACKS = [] # Add any callbacks you need  
batch_per_epoch=100  
  
model_history = bert_model.fit(  
    train_ds,  
    validation_data=valid_ds,  
    epochs=num_epochs,  
    steps_per_epoch=batch_per_epoch,  
    batch_size=16,  
    callbacks=MODEL_CALLBACKS  
)
```

```
Epoch 1/5  
100/100 [=====] - 1882s 19s/step - loss: 0.5402 - Accuracy: 0.7468 - Precision: 0.7809 - Recall: 0.6658 - val_loss: 0.3871 - val  
_Accuracy: 0.8094 - val_Precision: 1.0000 - val_Recall: 0.6004  
Epoch 2/5  
100/100 [=====] - 1770s 18s/step - loss: 0.1387 - Accuracy: 0.9548 - Precision: 0.9866 - Recall: 0.9183 - val_loss: 0.0302 - val  
_Accuracy: 0.9947 - val_Precision: 0.9894 - val_Recall: 0.9995  
Epoch 3/5  
100/100 [=====] - 1767s 18s/step - loss: 0.0326 - Accuracy: 0.9964 - Precision: 0.9983 - Recall: 0.9941 - val_loss: 0.0314 - val  
_Accuracy: 0.9958 - val_Precision: 0.9912 - val_Recall: 1.0000  
Epoch 4/5  
100/100 [=====] - 1765s 18s/step - loss: 0.0178 - Accuracy: 0.9972 - Precision: 0.9966 - Recall: 0.9974 - val_loss: 0.0200 - val  
_Accuracy: 0.9978 - val_Precision: 0.9954 - val_Recall: 1.0000  
Epoch 5/5  
100/100 [=====] - 1818s 18s/step - loss: 0.0228 - Accuracy: 0.9976 - Precision: 0.9976 - Recall: 0.9976 - val_loss: 0.0144 - val  
_Accuracy: 0.9982 - val_Precision: 0.9967 - val_Recall: 0.9995
```

### **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}")
```

```
Test Loss      : 0.02491799183189869  
Test Accuracy  : 0.9968819618225098  
Test Precision : 0.9947941303253174  
Test Recall    : 0.9985747933387756
```

### **Model training process:**

- The model is compiled with settings:
  - Optimizer: Adam with a learning rate of  $1e-5$  (0.00001).
  - Loss function: Binary cross-entropy, suitable for binary classification.
  - Metrics: Binary accuracy, precision, and recall are monitored during training
- The training loop runs for a specified number of epochs (controlled by the `num_epochs` variable, which is set to 5).
- Training data is fed to the model in batches.
- The model updates its weights and biases based on the loss calculated for each batch.
- The training process also uses callbacks for various purposes
- After training, the model is evaluated on a separate test dataset (`test_ds`).
- Metrics such as test loss, test accuracy, test precision, and test recall are calculated and displayed.

### **The Model's training history:**

```
model_history = pd.DataFrame(model_history.history)
print(model_history)
```

	loss	Accuracy	Precision	Recall	val_loss	val_Accuracy	\
0	0.540212	0.7468	0.780888	0.665844	0.387105	0.809354	
1	0.138698	0.9548	0.986559	0.918265	0.030188	0.994655	
2	0.032578	0.9964	0.998312	0.994118	0.031439	0.995768	
3	0.017791	0.9972	0.996567	0.997423	0.019986	0.997773	
4	0.022822	0.9976	0.997565	0.997565	0.014443	0.998218	

	val_Precision	val_Recall
0	1.000000	0.600374
1	0.989372	0.999533
2	0.991208	1.000000
3	0.995353	1.000000
4	0.996741	0.999533

### **Extract training history data**

```
train_loss = model_history['loss']
val_loss = model_history['val_loss']
train_accuracy = model_history['Accuracy']
val_accuracy = model_history['val_Accuracy']
train_precision = model_history['Precision']
val_precision = model_history['val_Precision']
train_recall = model_history['Recall']
val_recall = model_history['val_Recall']
```

### **Plot training history**

```
epochs = range(1, len(train_loss) + 1)
```

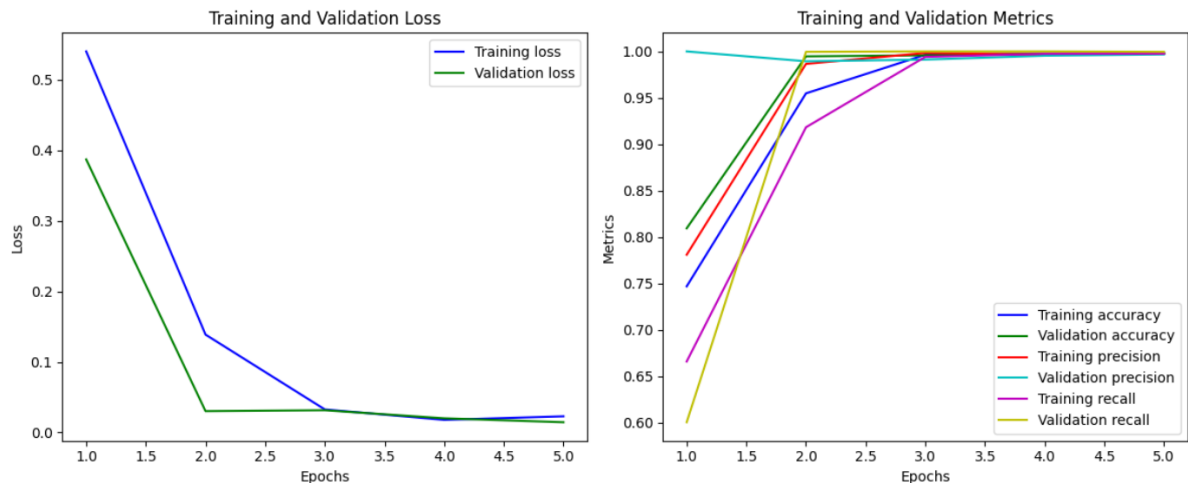
```
# Loss plot
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

### **Metrics plots**

```
plt.subplot(1, 2, 2)
plt.plot(epochs, train_accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'g', label='Validation accuracy')
plt.plot(epochs, train_precision, 'r', label='Training precision')
plt.plot(epochs, val_precision, 'c', label='Validation precision')
plt.plot(epochs, train_recall, 'm', label='Training recall')
```

```
plt.plot(epochs, val_recall, 'y', label='Validation recall')
plt.title('Training and Validation Metrics')
plt.xlabel('Epochs')
plt.ylabel('Metrics')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```



## **Conclusion:**

- The documentation covers the key steps involved in building a fake news detection model using NLP and BERT for classification. The document also mentions further evaluation using a classification report.
- BERT's understanding of contextual language and its fine-tuning capability make it a suitable candidate for fake news detection. The model has been optimized with specific settings for the task
- The project demonstrates a well-structured approach to model training, including batch processing and callbacks. Evaluation metrics such as accuracy, precision, recall, and F1-score are used to assess the model's performance on a test dataset
- As fake news detection involves potentially sensitive information and content, it's crucial to address ethical concerns, including data privacy, bias, and responsible use of the technology