**PROJECT 8 : FAKE NEWS DETECTION USING (NLP)**

**Phase – 4 submission**

**INTRODUCTION :**

Text preprocessing, feature extraction, model training, and evaluation are integral components of natural language processing (NLP), a field that empowers machines to understand and work with human language. In this process, raw text data is refined and converted into a format that can be digested by machine learning models, ultimately enabling these models to perform tasks like text classification, sentiment analysis, language translation, and more. This introductory explanation provides an overview of these fundamental stages in NLP, setting the stage for a deeper exploration of each step's importance and intricacies.

**TECHNIQUES USED IN (NLP) :**

* **Text Preprocessing**
* **Feature Extraction**
* **Model Training**
* **Model Evaluation**

**TEXT PREPROCESSING :**

* **Lowercasing:** Converting all text to lowercase to ensure uniformity.
* **Tokenization:** Splitting text into individual words or tokens.
* **Stop Word Removal:** Eliminating common words (e.g., "the," "and") that don't carry much information.
* **Stemming and Lemmatization:** Reducing words to their base or root form.
* **Removing Special Characters and Punctuation:** Getting rid of non-alphanumeric characters.
* **Handling Missing Data:** Dealing with missing values in the text.
* **Spell Checking and Correction**: Correcting common spelling errors.

**FEATURE EXTRACTION :**

* **Bag of Words (BoW):** Representing text as a matrix of word frequencies in the document.
* **TF-IDF (Term Frequency-Inverse Document Frequency):** Assigning weights to words based on their importance in the document and the entire corpus.
* **Word Embeddings** (e.g., Word2Vec, GloVe): Capturing semantic relationships between words by mapping them to dense vectors.
* **N-grams:** Capturing sequences of words (bigrams, trigrams, etc.).
* **Document Embeddings** (e.g., Doc2Vec): Creating vector representations of entire documents.
* **Feature Engineering:** Creating custom features that are relevant to the specific NLP task.

**MODEL TRAINING :**

* **Naive Bayes:** A probabilistic model used for text classification tasks.
* **Support Vector Machines (SVM):** Effective for text classification and sentiment analysis.
* **Recurrent Neural Networks (RNN):** Suitable for sequential data like text.
* **Convolutional Neural Networks (CNN):** Useful for tasks like text classification and text generation.
* **Transformer Models** (e.g., BERT, GPT): State-of-the-art for various NLP tasks, including language understanding and generation.

**MODEL EVALUATION :**

* **Accuracy:** Measures the proportion of correctly classified instances.
* **Precision and Recall:** Useful metrics for binary classification tasks.
* **F1-score:** Combines precision and recall to balance trade-offs.
* **Mean Squared Error (MSE) or Root Mean Squared Error (RMSE):** For regression tasks.
* **Cross-Validation:** Splitting data into multiple subsets for robust evaluation.
* **Confusion Matrix**: Provides a detailed breakdown of model performance.

**ALGORITHM :**

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

import nltk

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

import matplotlib.pyplot as plt

# Load the true and false datasets

true\_data = pd.read\_excel("true.xlsx")

false\_data = pd.read\_excel("false.xlsx")

# Combine the datasets into one

true\_data['label'] = 1

false\_data['label'] = 0

data = pd.concat([true\_data, false\_data], ignore\_index=True)

# Text Preprocessing

nltk.download('stopwords')

nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

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

def preprocess\_text(text):

words = text.split()

words = [word for word in words if word not in stop\_words]

words = [lemmatizer.lemmatize(word) for word in words]

return ' '.join(words)

data['text'] = data['text'].str.lower()

data['text'] = data['text'].apply(preprocess\_text)

# Feature Extraction: TF-IDF vectorization

tfidf\_vectorizer = TfidfVectorizer(max\_features=5000) # You can adjust max\_features as needed

X = tfidf\_vectorizer.fit\_transform(data['text'])

y = data['label']

# Split the data into training and testing sets

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

# Model Training: Random Forest Classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_classifier.fit(X\_train, y\_train)

# Model Evaluation

y\_pred = rf\_classifier.predict(X\_test)

# Calculate accuracy and print classification report

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Classification Report:\n", report)

# Create a simple accuracy plot

plt.bar(['Accuracy'], [accuracy], color='blue')

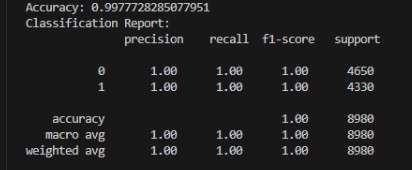
plt.ylim(0, 1) # Set the y-axis limit to show accuracy between 0 and 1

plt.ylabel('Accuracy')

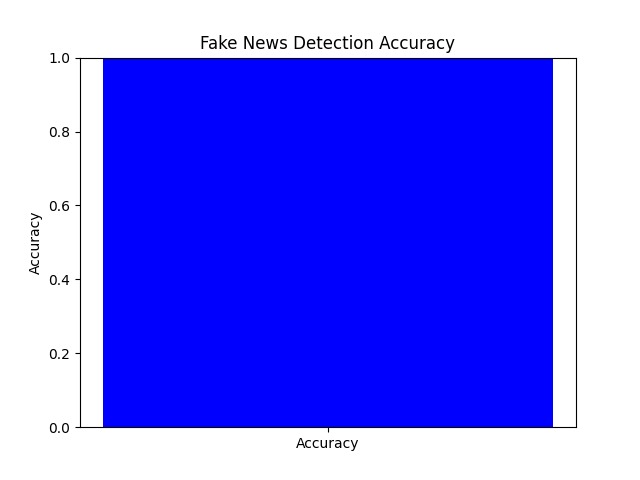
plt.title('Fake News Detection Accuracy')

plt.show()

**OUTPUT :**

****

**GRAPHICAL REPRESENTATION :**

****

**KEYS IN (NLP) :**

* Data Quality Matters
* Evaluation Metrics
* Fine -Tunning
* Continuous Monitoring
* Interpretability

**CONCLUSION :**

Fake news detection involves collecting and preprocessing data, extracting features from text, training a machine learning model, and evaluating its performance. The choice of quality data, effective text preprocessing, and the right model are critical for accurate detection. Continuous monitoring and transparency are also important in the dynamic landscape of fake news.