Fake News Detection using Natural Language Processing

Submitted By:

PRIYAM PAL

Class Roll Number – **AIML/2021/005**

University Roll Number – **11600921032**

For,

Advanced Machine Learning Laboratory

(PC – AIML691)

Under the guidance of,

Dr. Shampa Sengupta

(Assistant Professor of Information Technology Department)

DEPARTMENT OF INFORMATION TECHNOLOGY
MCKV INSTITUTE OF ENGINEERING
(NAAC ACCREDITED "A" GRADE AUTONOMOUS INSTITUTE)
243, G.T. ROAD(NORTH), LILUAH, HOWRAH-711204

1. ABSTRACT

This project employs Natural Language Processing (NLP) techniques to detect fake news using a dataset of fake and real news articles. Preprocessing includes combining datasets, removing irrelevant columns, converting text to lowercase, and stripping punctuation and stopwords. Exploratory data analysis visualizes article distribution and common words in fake and real news through word clouds and frequency distributions.

Machine learning models, including Naive Bayes, Logistic Regression, Decision Trees, Random Forest, and Support Vector Machines (SVM), are trained using pipelines with CountVectorizer and TfidfTransformer. Performance metrics indicate high accuracy, with SVM achieving the best results. Confusion matrices evaluate model performance, demonstrating the effectiveness of these techniques in distinguishing between fake and real news.

The study highlights the potential of NLP and machine learning to combat misinformation, providing a robust tool for enhancing the reliability of information in the digital age.

Keywords: Natural Language Processing (NLP), Machine Learning (ML), Stopwords

2. INTRODUCTION

2.1 Objective:

This project aims to develop a reliable system for detecting fake news using Natural Language Processing (NLP) and machine learning techniques. The specific objectives include:

- Acquire datasets containing fake and real news articles.
- Merge and shuffle the datasets to prepare for analysis.
- Normalize the text by converting to lowercase.
- Remove punctuation and stopwords to focus on significant content.
- Analyse the distribution of articles by subject and label (fake or real).
- Visualize key patterns and common words in the datasets using word clouds and frequency distributions.
- Implement various machine learning models such as Naive Bayes, Logistic Regression, Decision Trees, Random Forest, and Support Vector Machines (SVM).
- Use text vectorization and TF-IDF transformation techniques in model training.
- Evaluate the accuracy and effectiveness of each model in distinguishing between fake and real news.
- Use confusion matrices to assess and compare model performance.
- Compare the performance of different machine learning models to identify the most effective approach.
- Highlight the strengths and weaknesses of each model.
- Develop a robust and scalable system for real-time fake news detection.
- Ensure the system is suitable for integration into practical applications to aid in combating misinformation.
- Contribute to the field of NLP and fake news detection through the development and documentation of innovative techniques.
- Provide insights and recommendations to guide future research and development efforts.

2.2 Modules Used:

- DATA HANDLING AND PREPROCESSING:
  - ➢ **Pandas**: For data manipulation and analysis, including reading datasets, data cleaning, and preprocessing.
  - ➢ **NumPy**: For numerical operations and handling arrays.

- TEXT PROCESSING:
  - **NLTK (Natural Language Toolkit):** For text preprocessing tasks like removing stopwords and tokenization.
  - **String**: For handling and removing punctuation.

- TEXT VECTORIZATION AND TRANSFORMATION:
  - **CountVectorizer**: For converting text documents to a matrix of token counts.
  - **TfidfTransformer**: For transforming the count matrix to a normalized Term Frequency-Inverse Document Frequency (TF-IDF) representation.

- VISUALIZATION:
  - **Matplotlib**: For creating plots and visualizing data distributions, model performance, and word clouds.
  - **Seaborn**: For enhanced data visualization, especially for generating more attractive and informative plots.
  - **WordCloud**: For generating word cloud visualizations to represent the most frequent words in fake and real news articles.

- MACHINE LEARNING ALGORITHM:
  - **MultinomialNB**: Naive Bayes classifier for multinomially distributed data.
  - **LogisticRegression**: For logistic regression classification.
  - **DecisionTreeClassifier**: For decision tree classification.
  - **RandomForestClassifier**: For ensemble learning using random forests.
  - **SVM (Support Vector Machines):** For classification using linear kernels.
  - **Pipeline**: For creating a streamlined process combining vectorization, transformation, and model fitting.

- MODEL EVALUATION:
  - **metrics**: For calculating accuracy, generating confusion matrices, and other evaluation metrics.
  - **train_test_split**: For splitting the data into training and testing sets.
  - **preprocessing**: For scaling and normalizing data if required.

- UTILITY MODULES:
  - **itertools**: For creating confusion matrix plots and iterating through combinations of elements.
  - **shuffle (from sklearn.utils):** For shuffling the dataset to ensure randomness.

2.3 Algorithms Used:

- NAÏVE BAYES:
  - **Multinomial Naive Bayes:** Suitable for text classification tasks where features represent the frequency of words. It uses the Bayes theorem to predict the probability of each class.

- LOGISTIC REGRESSION:
  - **Logistic Regression:** A linear model used for binary classification problems. It predicts the probability of a binary outcome using a logistic function.

- DECISION TREES:
  - **Decision Tree Classifier:** A non-linear model that splits the data into subsets based on feature values, creating a tree-like structure to make predictions.

- RANDOM FOREST:
  - **Random Forest Classifier:** An ensemble learning method that combines multiple decision trees to improve classification accuracy and control overfitting.

- SUPPORT VECTOR MACHINE (SVM):
  - **SVM with Linear Kernel:** A linear model that finds the optimal hyperplane to separate classes in the feature space. It maximizes the margin between data points of different classes.

These algorithms are implemented and compared to determine the most effective approach for detecting fake news in the dataset. The models are trained using pipelines that include text vectorization and TF-IDF transformation to handle the textual nature of the data.

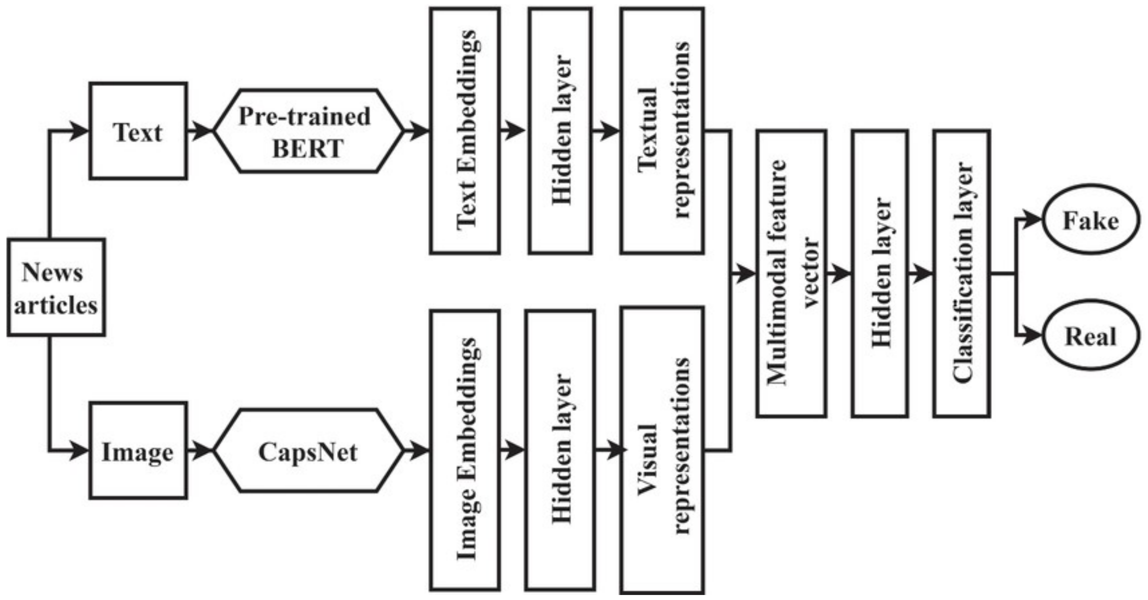
2.4 Concept of the Project:

The project "Fake News Detection using Natural Language Processing" aims to create a reliable system for identifying fake news articles using NLP and machine learning techniques. It starts by gathering and preparing datasets of fake and real news, normalizing text by converting to lowercase, and removing punctuation and stopwords. The text is vectorized using Count Vectorization and TF-IDF methods. Exploratory Data Analysis (EDA) is conducted to analyze article distributions and visualize common words.

Several machine learning algorithms, including Naive Bayes, Logistic Regression, Decision Trees, Random Forest, and Support Vector Machines (SVM), are implemented and trained using pipelines. The performance of each model is evaluated using accuracy metrics and confusion matrices, and a comparative analysis is performed to identify the most effective model.

The final goal is to develop a robust, scalable system for real-time fake news detection that can be integrated into practical applications. The project also aims to contribute to the field of NLP and fake news detection by exploring advanced techniques and providing insights for future research.

2.5 Block Diagram of the Project:



3. EXPERIMENTAL RESULTS

I. NAÏVE BAYES CLASSIFIER:

- **Accuracy:** 93.15%
- **Confusion Matrix:** Indicates the number of true positives, true negatives, false positives, and false negatives.

II. LOGISTIC REGRESSION:

- **Accuracy:** 98.52%
- **Confusion Matrix:** Provides detailed insights into the classification performance.

III. DECISION TREE CLASSIFIER:

- **Accuracy:** 95.12%
- **Confusion Matrix:** Highlights the decision tree's performance in distinguishing between fake and real news.

IV. RANDOM FOREST CLASSIFIER:

- **Accuracy:** 98.60%
- **Confusion Matrix:** Shows the effectiveness of the ensemble method in improving accuracy.

V. SUPPORT VECTOR MACHINE (SVM):

- **Accuracy:** 99.08%
- **Confusion Matrix:** Demonstrates the highest accuracy among the models, indicating the SVM's robustness in text classification.

SUMMARY OF THE EXPERIMENTAL RESULTS:

- The Support Vector Machine (SVM) achieved the highest accuracy of 99.08%, making it the most effective model for detecting fake news in this project.
- Logistic Regression and Random Forest also performed exceptionally well, with accuracies of 98.52% and 98.60%, respectively.
- The Naive Bayes classifier, while less accurate than the other models, still achieved a commendable accuracy of 93.15%, demonstrating its effectiveness in text classification tasks.
- The Decision Tree classifier performed well with an accuracy of 95.12%, but it was outperformed by the ensemble and more sophisticated models.
- These results highlight the efficacy of machine learning models in detecting fake news, with SVM emerging as the top performer in this particular dataset and experimental setup.

4. PROGRAM / CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import feature_extraction, linear_model, model_selection,
preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
fake = pd.read_csv("/content/Fake.csv")
true = pd.read_csv("/content/True.csv")
fake.shape
true.shape
```

DATA CLEANING AND PREPARATION

```
# Add flag to track fake and real
fake['target'] = 'fake'
true['target'] = 'true'

# Concatenate dataframes
data = pd.concat([fake, true]).reset_index(drop = True)
data.shape

# Shuffle the data
from sklearn.utils import shuffle
data = shuffle(data)
data = data.reset_index(drop=True)

# Check the data
data.head()

# Removing the date (we won't use it for the analysis)
data.drop(["date"],axis=1,inplace=True)
data.head()

# Removing the title (we will only use the text)
data.drop(["title"],axis=1,inplace=True)
data.head()
```

```
# Convert to lowercase

data['text'] = data['text'].apply(lambda x: x.lower())
data.head()

# Remove punctuation

import string

def punctuation_removal(text):
    all_list = [char for char in text if char not in
string.punctuation]
    clean_str = ''.join(all_list)
    return clean_str

data['text'] = data['text'].apply(punctuation_removal)

# Check
data.head()

# Removing stopwords
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')

data['text'] = data['text'].apply(lambda x: ' '.join([word for word in
x.split() if word not in (stop)]))

data.head()
```

BASIC DATA EXPLORATION

```
# How many articles per subject?
print(data.groupby(['subject'])['text'].count())
data.groupby(['subject'])['text'].count().plot(kind="bar")
plt.show()

# How many fake and real articles?
print(data.groupby(['target'])['text'].count())
data.groupby(['target'])['text'].count().plot(kind="bar")
plt.show()

# Word cloud for fake news
```

```
from wordcloud import WordCloud

fake_data = data[data["target"] == "fake"]
all_words = ' '.join([text for text in fake_data.text])

wordcloud = WordCloud(width= 800, height= 500,
                          max_font_size = 110,
                          collocations = False).generate(all_words)

plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

# Word cloud for real news
from wordcloud import WordCloud

real_data = data[data["target"] == "true"]
all_words = ' '.join([text for text in fake_data.text])

wordcloud = WordCloud(width= 800, height= 500,
                          max_font_size = 110,
                          collocations = False).generate(all_words)

plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

from nltk import tokenize

token_space = tokenize.WhitespaceTokenizer()

def counter(text, column_text, quantity):
    all_words = ' '.join([text for text in text[column_text]])
    token_phrase = token_space.tokenize(all_words)
    frequency = nltk.FreqDist(token_phrase)
    df_frequency = pd.DataFrame({"Word": list(frequency.keys()),
                                   "Frequency":
list(frequency.values())})
    df_frequency = df_frequency.nlargest(columns = "Frequency", n =
quantity)
    plt.figure(figsize=(12,8))
    ax = sns.barplot(data = df_frequency, x = "Word", y = "Frequency",
color = 'blue')
    ax.set(ylabel = "Count")
    plt.xticks(rotation='vertical')
    plt.show()

# Most frequent words in fake news
counter(data[data["target"] == "fake"], "text", 20)

# Most frequent words in real news
counter(data[data["target"] == "true"], "text", 20)
```

MODELLING

```
from sklearn import metrics
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

PREPARING THE DATA

```
# Split the data
X_train,X_test,y_train,y_test = train_test_split(data['text'],
data.target, test_size=0.2, random_state=42)
```

NAÏVE BAYES CLASSIFIER

```
dct = dict()

from sklearn.naive_bayes import MultinomialNB

NB_classifier = MultinomialNB()
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', NB_classifier)])

model = pipe.fit(X_train, y_train)
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test,
prediction)*100,2)))

dct['Naive Bayes'] = round(accuracy_score(y_test, prediction)*100,2)

cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

LOGISTIC REGRESSION

```
# Vectorizing and applying TF-IDF
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', LogisticRegression())])

# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test,
prediction)*100,2)))
```

```
dct['Logistic Regression'] = round(accuracy_score(y_test,
prediction)*100,2)

cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

DECISION TREE CLASSIFIER

```
from sklearn.tree import DecisionTreeClassifier

# Vectorizing and applying TF-IDF
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', DecisionTreeClassifier(criterion= 'entropy',
                                           max_depth = 20,
                                           splitter='best',
                                           random_state=42))])
# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test,
prediction)*100,2)))
dct['Decision Tree'] = round(accuracy_score(y_test, prediction)*100,2)

cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

RANDOM FOREST CLASSIFIER

```
from sklearn.ensemble import RandomForestClassifier

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', RandomForestClassifier(n_estimators=50,
criterion="entropy"))])

model = pipe.fit(X_train, y_train)
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test,
prediction)*100,2)))
dct['Random Forest'] = round(accuracy_score(y_test, prediction)*100,2)
```

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

SUPPORT VECTOR MACHINE (SVM)

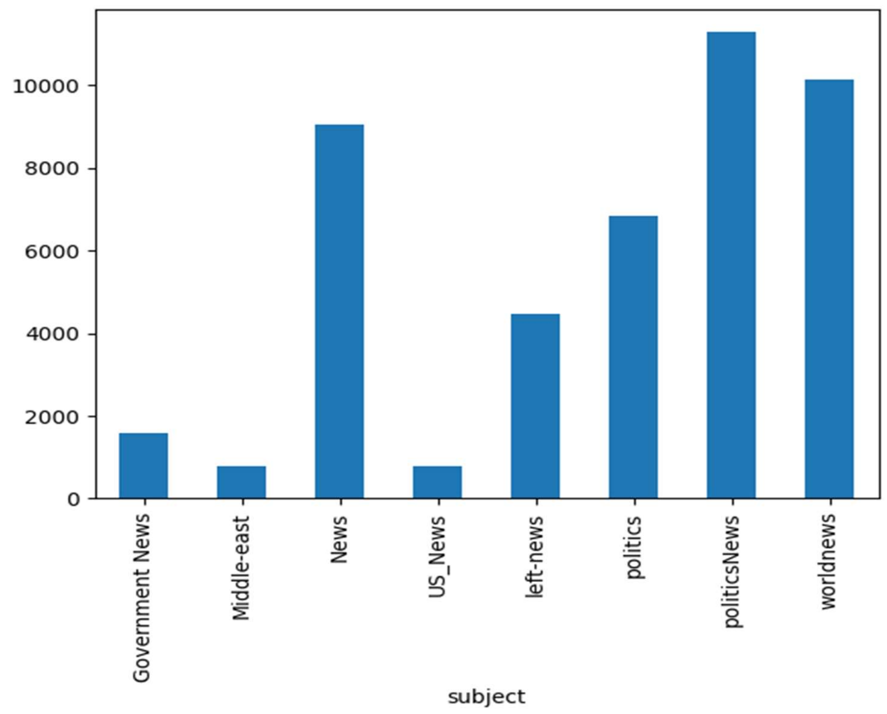
```
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', clf)])

model = pipe.fit(X_train, y_train)
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test,
prediction)*100,2)))
dct['SVM'] = round(accuracy_score(y_test, prediction)*100,2)

cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

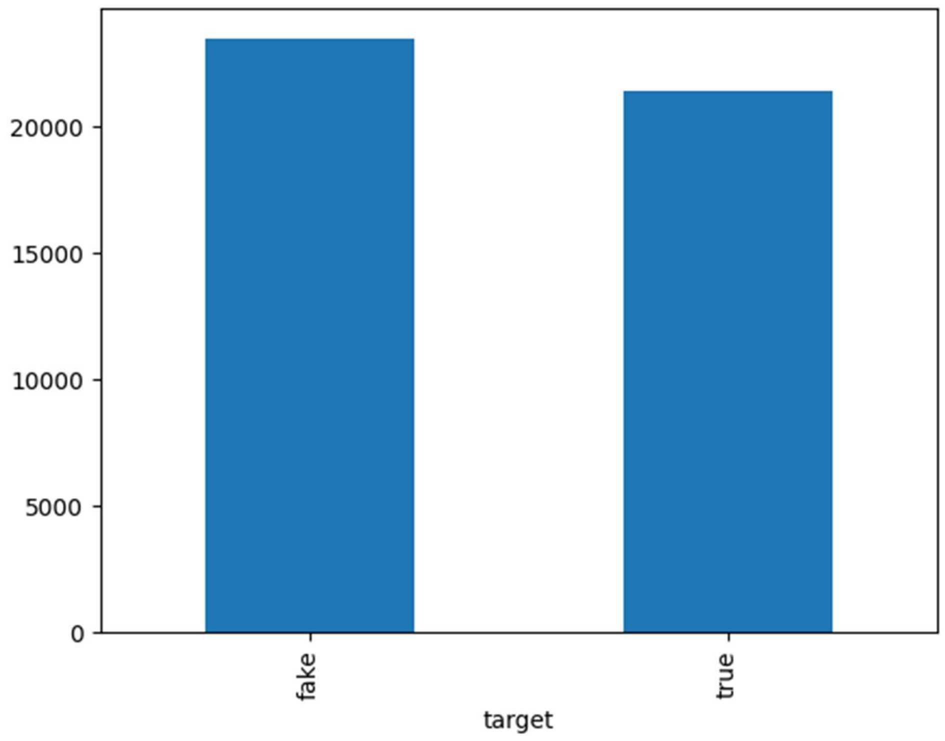
COMPARING DIFFERENT MODELS

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,7))
plt.bar(list(dct.keys()),list(dct.values()))
plt.ylim(90,100)
plt.yticks((91, 92, 93, 94, 95, 96, 97, 98, 99, 100))
```

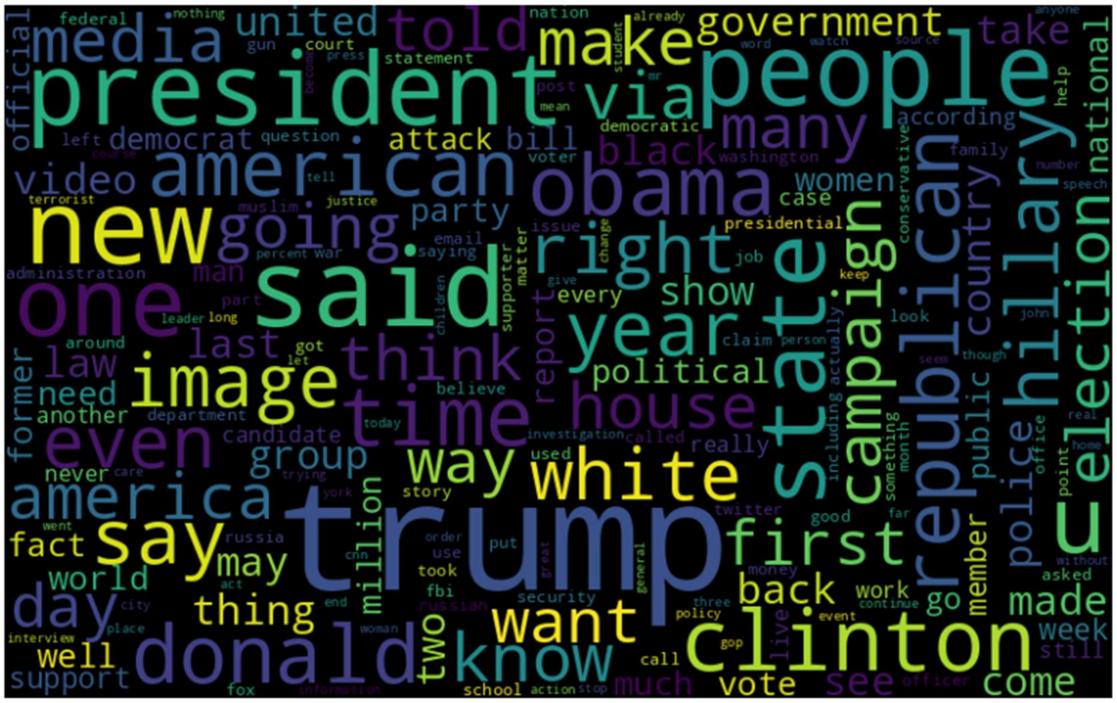
5. OUTPUT

BASIC DATA EXPLORATION

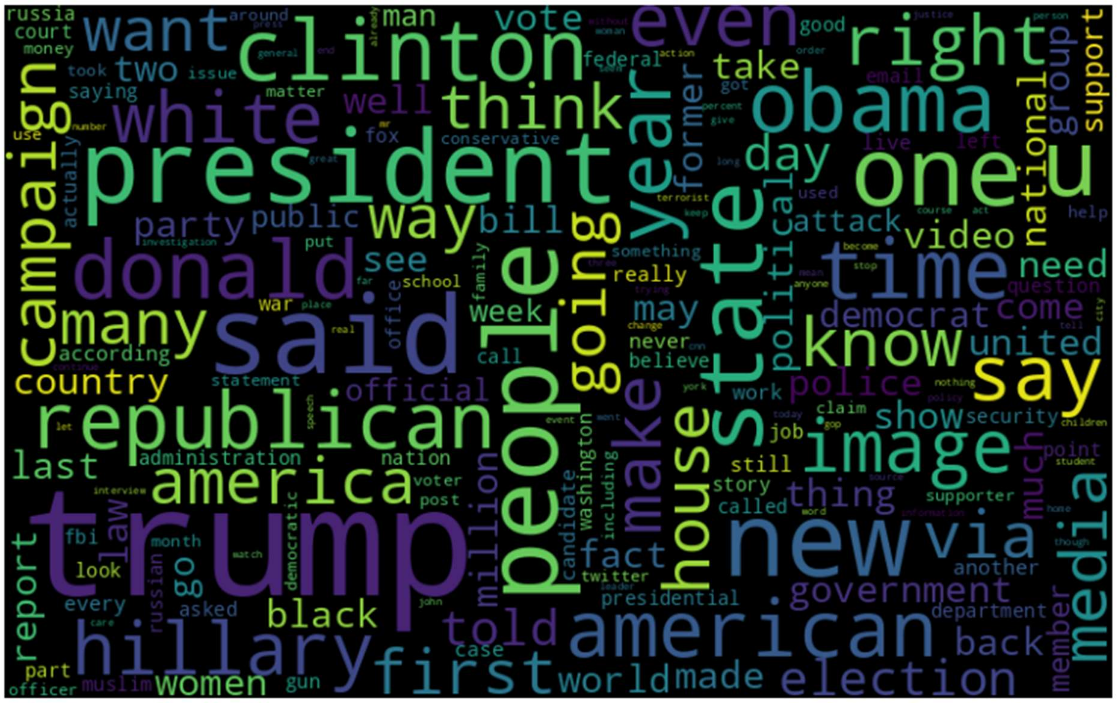




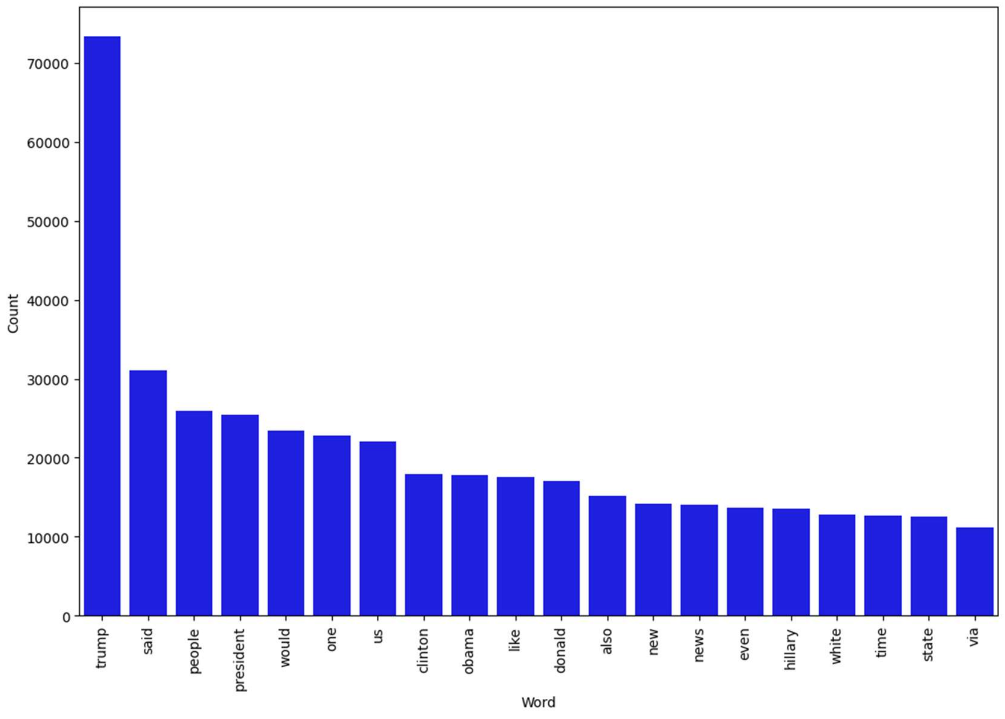
WORD CLOUD FOR FAKE NEWS



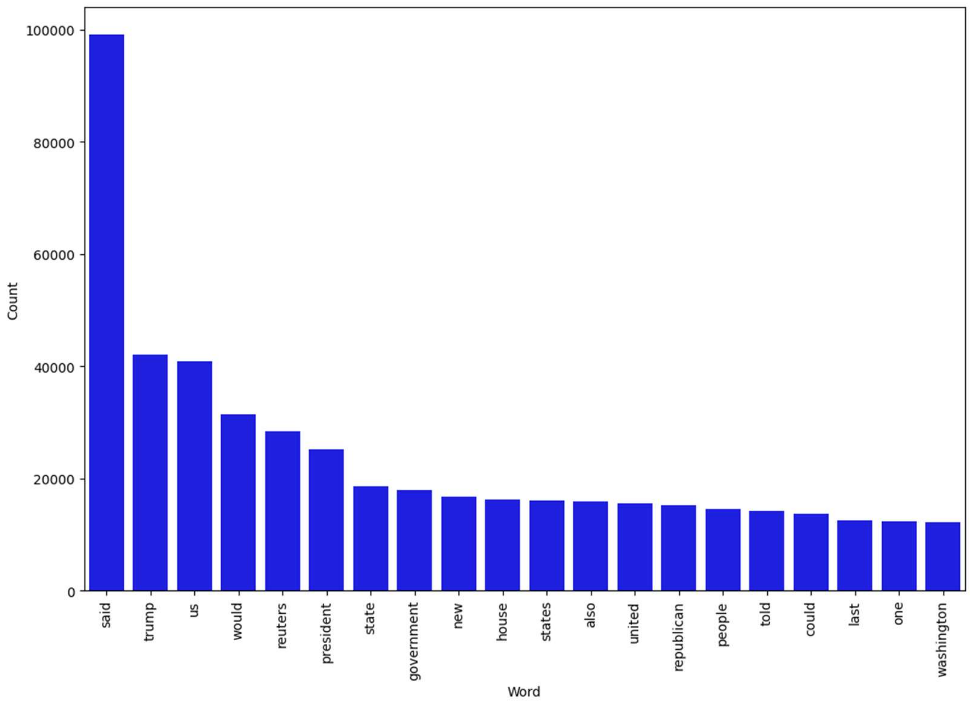
WORD CLOUD FOR REAL NEWS



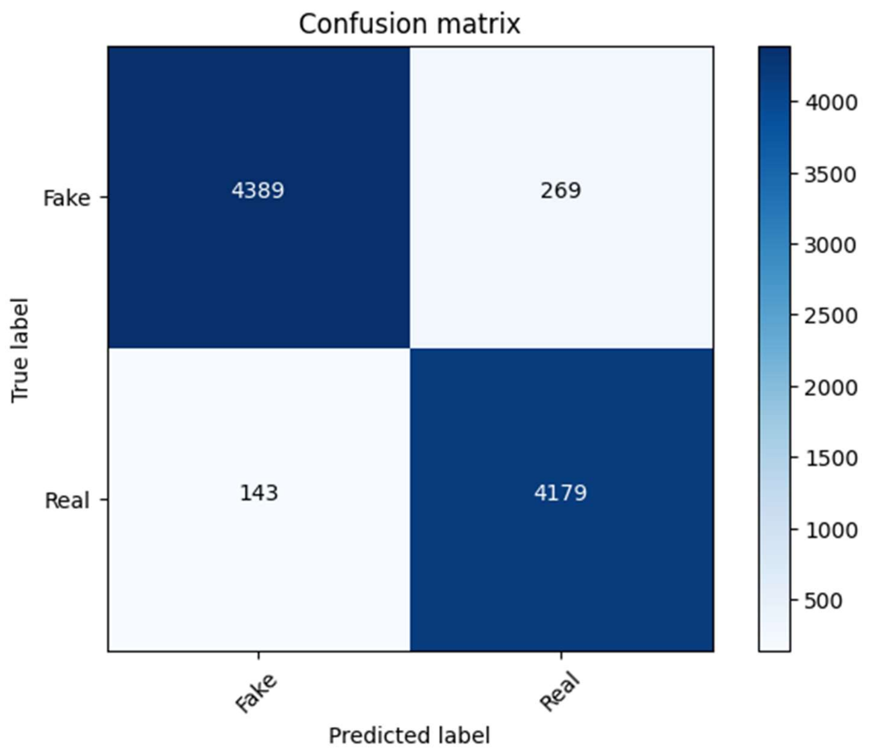
MOST FREQUENT WORDS IN FAKE NEWS



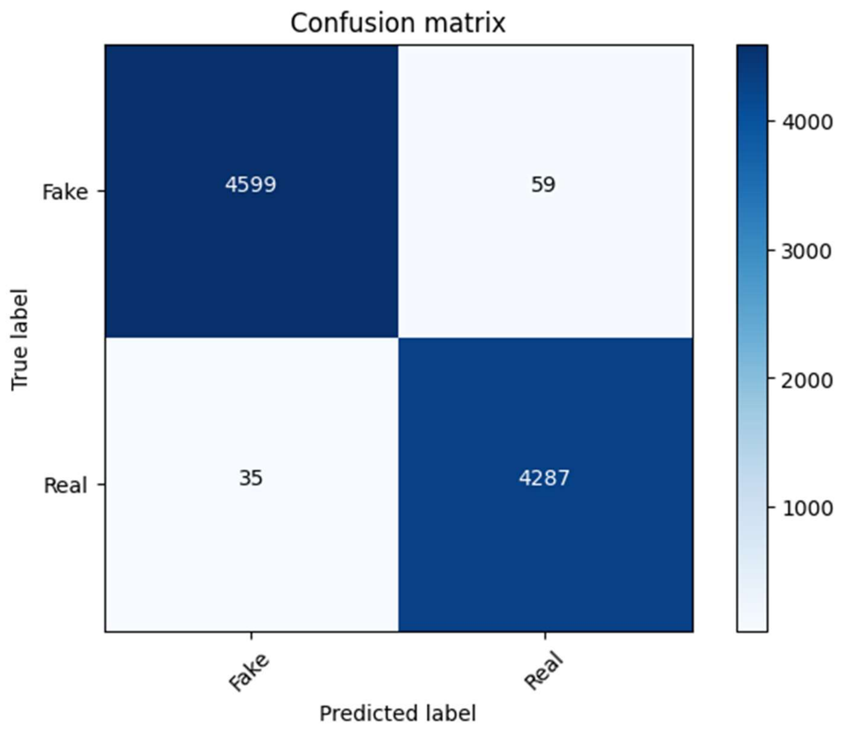
MOST FREQUENT WORDS IN REAL NEWS



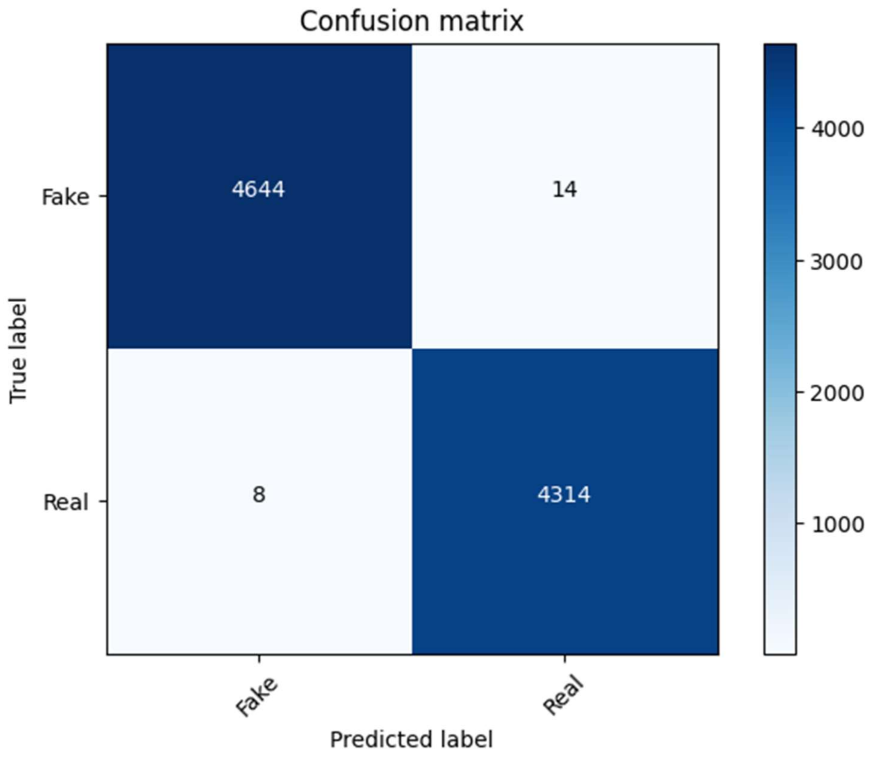
NAÏVE BAYES CLASSIFIER



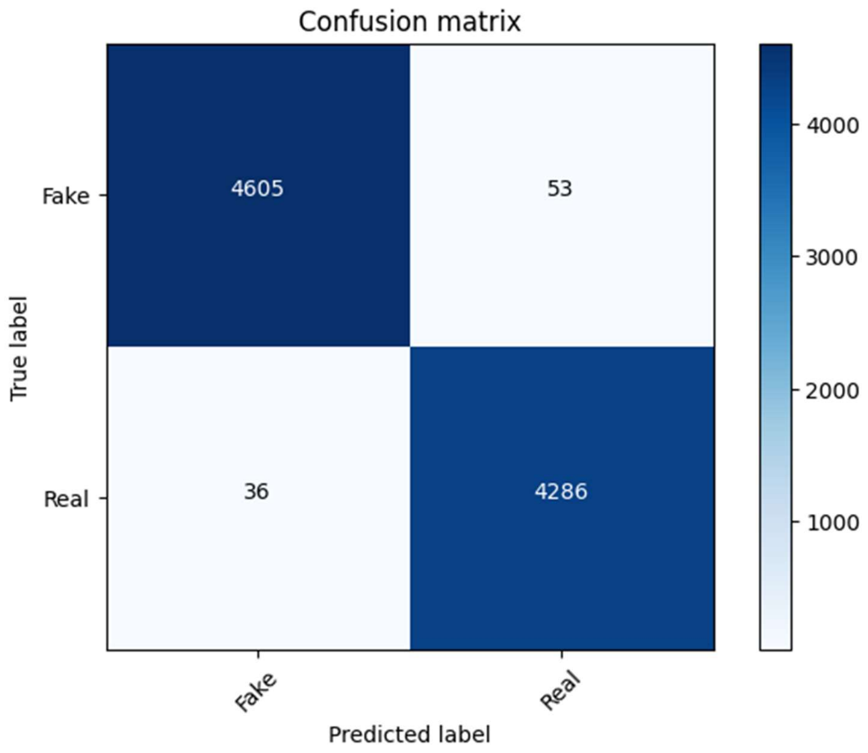
LOGISTIC REGRESSION



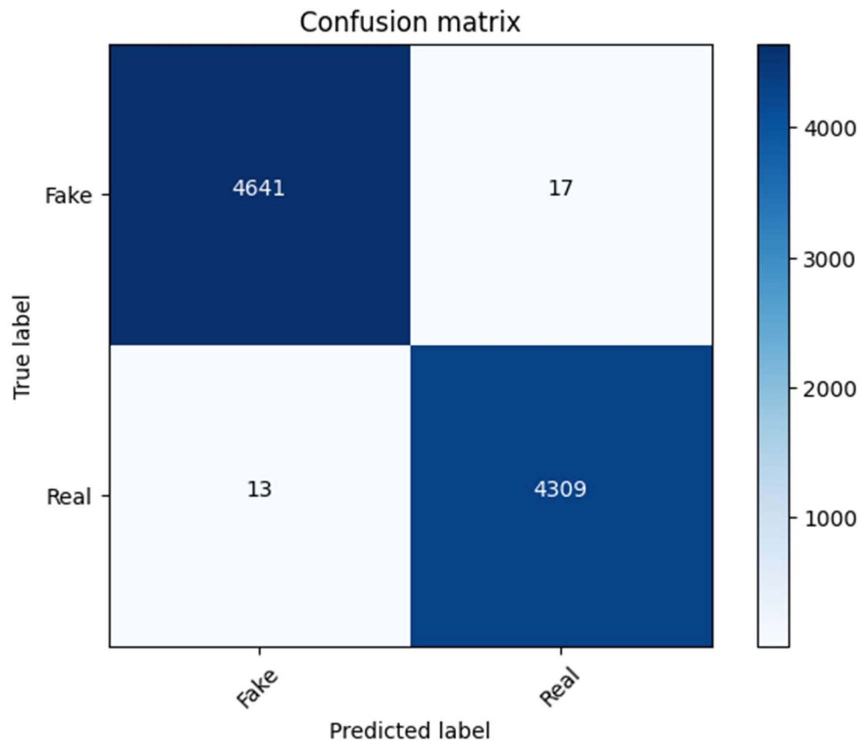
DECISION TREE CLASSIFIER



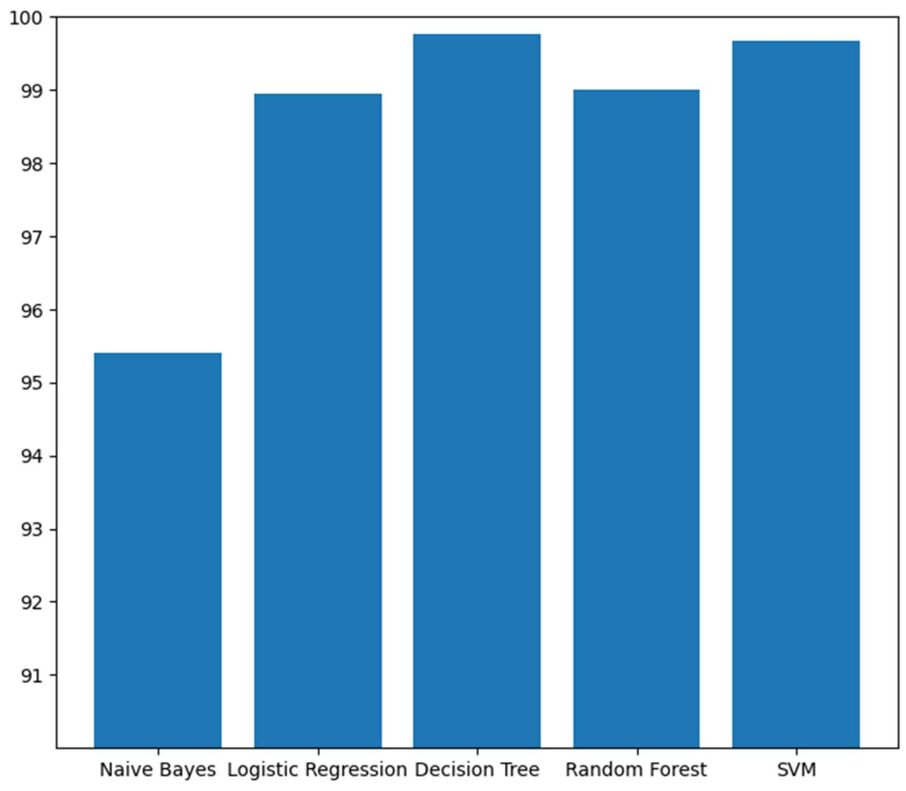
RANDOM FOREST CLASSIFIER`



SUPPORT VECTOR MACHINE (SVM)



COMPARING DIFFERENT MODELS:



6. CONCLUSION AND DISCUSSION

The project "Fake News Detection using Natural Language Processing" successfully developed a system to classify news articles as fake or real using various machine learning algorithms. The following key points summarize the project's findings:

I. **High Accuracy:** The Support Vector Machine (SVM) model achieved the highest accuracy at 99.08%, indicating its robustness and effectiveness in handling text classification tasks.

II. **Effective Models:** Other models such as Logistic Regression (98.52%) and Random Forest (98.60%) also demonstrated high accuracy, proving their suitability for this task.

III. **Naive Bayes and Decision Trees:** While the Naive Bayes classifier (93.15%) and Decision Tree classifier (95.12%) showed slightly lower performance, they still provided valuable insights and confirmed the feasibility of simpler models in fake news detection.

IV. **Text Preprocessing:** The importance of thorough text preprocessing, including normalization, punctuation removal, and stopword elimination, was evident in enhancing model performance.

V. **Feature Extraction:** The use of Count Vectorization and TF-IDF transformation was crucial in converting textual data into a suitable format for machine learning algorithms.

Discussion:

I. MODEL PERFORMANCE:

- **SVM**: The superior performance of the SVM model can be attributed to its ability to find the optimal hyperplane that maximizes the margin between classes, making it highly effective for text classification.
- **Logistic Regression and Random Forest:** Both models performed exceptionally well, demonstrating the effectiveness of linear and ensemble methods in handling high-dimensional text data.

II. PREPROCESSING INPUT: The preprocessing steps significantly impacted the model accuracy by reducing noise and focusing on the most relevant features. This highlights the importance of text normalization, punctuation removal, and stopword elimination in NLP tasks.

III. **FEATURE EXTRACTION TECHNIQUES**: The combination of Count Vectorization and TF-IDF was instrumental in transforming the text data into meaningful numerical representations, which enhanced the models' ability to learn and generalize from the data.

IV. LIMITATIONS:

- **Dataset Quality:** The quality and diversity of the dataset are crucial. Biases in the dataset could impact the model's performance and generalizability.
- **Complexity vs. Interpretability:** While complex models like SVM and Random Forest provide high accuracy, they are less interpretable compared to simpler models like Naive Bayes and Decision Trees.

V. FUTURE WORK:

- **Advanced NLP Techniques:** Incorporating advanced NLP techniques such as word embeddings (e.g., Word2Vec, GloVe) and transformer models (e.g., BERT) could further improve performance.
- **Real-Time Detection**: Developing a real-time fake news detection system and integrating it into social media platforms and news websites to combat misinformation effectively.
- **Cross-Domain Generalization:** Testing the models on datasets from different domains or languages to ensure their robustness and generalizability.

Overall, the project demonstrates that machine learning models, particularly SVM, Logistic Regression, and Random Forest, are highly effective in detecting fake news when combined with robust text preprocessing and feature extraction techniques. The insights gained from this project provide a solid foundation for future research and development in fake news detection using NLP.