Fake News Detection using Machine Learning

Introduction

In today's digital age, the rapid spread of information has made it easier for people to access news and updates. However, this accessibility has also increased the proliferation of fake news, which can mislead and misinform the public. Identifying fake news effectively is crucial to maintain the integrity of information. This project addresses this challenge by utilizing machine learning techniques to classify news articles as either real or fake.

Code Explanation

The code for the Fake News Detection project follows a systematic approach, starting from data preprocessing to model training and evaluation. Below is a breakdown of the main components:

1. Data Loading and Exploration

The dataset is loaded using the Pandas library, and the initial exploration provides insights into its shape and structure. The dataset contains 44,919 samples and includes columns such as `text`, `title`, `subject`, `date`, and `class` (where 1 indicates real news and 0 indicates fake news). Unnecessary columns like `title`, `subject`, and `date` are dropped to focus on the text data.

2. Data Preprocessing

Preprocessing is essential for text data. The steps include:

- **Removing Punctuation:** All non-word characters are removed.
- **Tokenization:** The text is split into individual words using NLTK's `word_tokenize`.
- **Stop Words Removal:** Common English words (e.g., "and", "the") are filtered out.
- **Stemming:** The words are reduced to their base form using Porter Stemmer, making the text simpler and more uniform.

A helper function `preprocess_text` is defined to apply these steps to the entire dataset.

3. Data Visualization

A Word Cloud is generated to visualize the most frequent words in both real and fake news articles. This helps understand the common terms used in each category. Additionally, a bar chart shows the top 20 most common words in the dataset.

4. Feature Extraction

The text data is vectorized using **TF-IDF Vectorizer**, which converts the text into numerical features. This representation captures the importance of words while reducing the impact of commonly occurring words.

5. Model Building and Training

Several machine learning models are tested:

- **Logistic Regression**: A simple yet effective model that achieves high accuracy.
- **Decision Tree Classifier**: This model shows even better performance with an accuracy of over 99%.

The models are trained using the `train_test_split` function, which divides the data into training and testing sets (75% training, 25% testing).

6. Model Evaluation

The models are evaluated based on their accuracy scores. Both training and testing accuracies are reported, showing the effectiveness of the Decision Tree Classifier. The confusion matrix further illustrates the model's performance in distinguishing real and fake news.

7. Prediction Function

A function `predict_news` is defined to take a new text input, preprocess it, vectorize it using TF-IDF, and make a prediction using the trained Decision Tree model. The function returns whether the input news is predicted as "Real" or "Fake."

8. User Interaction

The code includes an interactive component where the user can input a news article, and the model will classify it in real time. This demonstrates the practical application of the project in a real-world scenario.

Conclusion

The Fake News Detection project demonstrates the effective use of machine learning techniques in addressing a real-world problem. By leveraging text preprocessing, feature extraction with TF-IDF, and various classification models, the project successfully identifies fake news with high accuracy. Among the models tested, the Decision Tree Classifier provided the best results, achieving an accuracy of over 99%. This highlights the potential of machine learning in automating the detection of misinformation, which is crucial in today’s digital landscape.

While the results are promising, future improvements could include exploring more advanced models like neural networks and incorporating additional features such as metadata from the news articles. Overall, this project showcases a practical application of machine learning in natural language processing, contributing to the fight against the spread of fake news.

CODE

[2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[4]:
```
data = pd.read_csv('News.csv',index_col=0)
data.head(50)
data.shape
```

[4]:
```
(44919, 5)
```

[5]:
```
data.shape
```

[5]:
```
(44919, 5)
```

[6]:
```
data = data.drop(["title", "subject","date"], axis = 1)
```

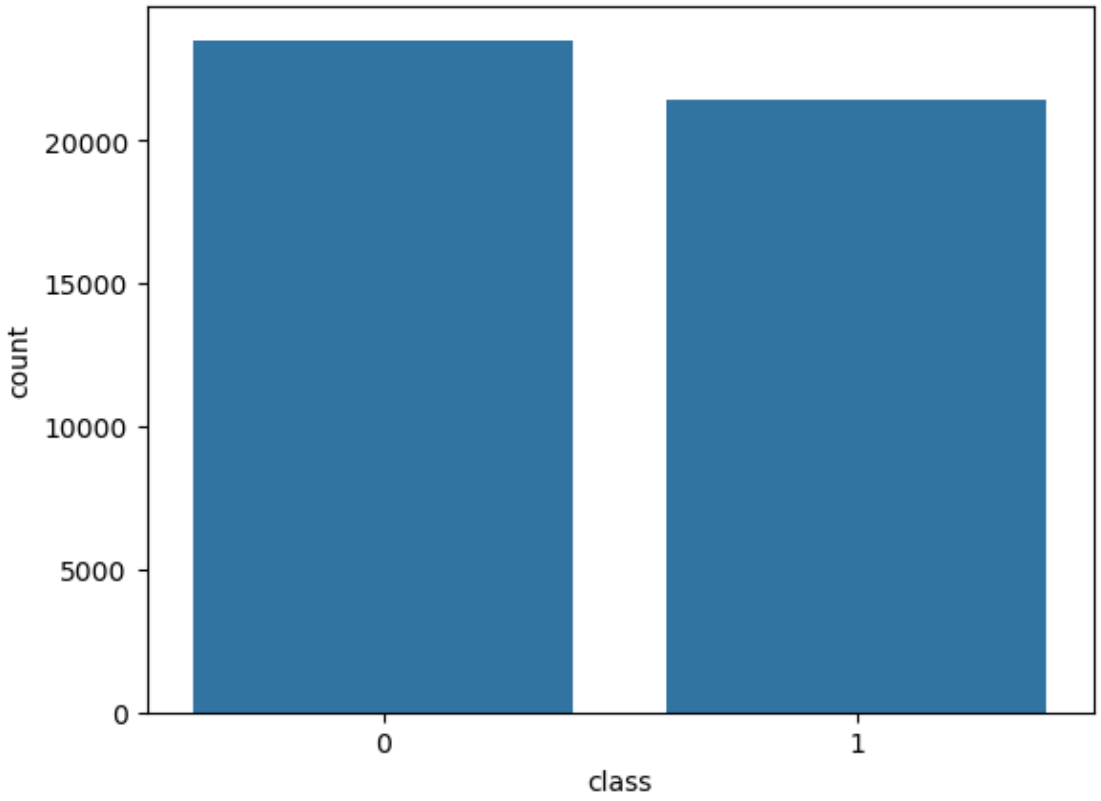
[7]:
```
data.isnull().sum()
```

[7]:
```
text     0
class    0
dtype: int64
```

[8]:
```
# Shuffling
data = data.sample(frac=1)
data.reset_index(inplace=True)
data.drop(["index"], axis=1, inplace=True)
```

[9]:
```
sns.countplot(data=data,
                        x='class',
                        order=data['class'].value_counts().index)
```

[9]:
```
<Axes: xlabel='class', ylabel='count'>
```



[10]:
```
from tqdm import tqdm
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ASUS\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ASUS\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

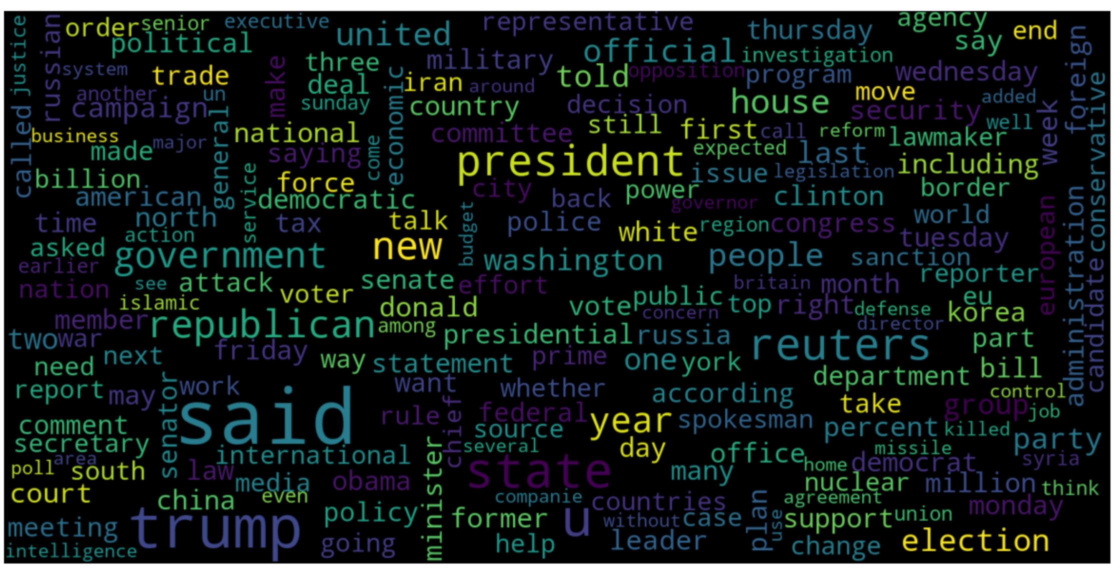
[12]:
```
def preprocess_text(text_data):
        preprocessed_text = []

        for sentence in tqdm(text_data):
```

```
                sentence = re.sub(r'[^\w\s]', '', sentence)
                preprocessed_text.append(' '.join(token.lower()
                                                      for token in␣
↪str(sentence).split()
                                                      if token not in␣
↪stopwords.words('english')))

        return preprocessed_text
```

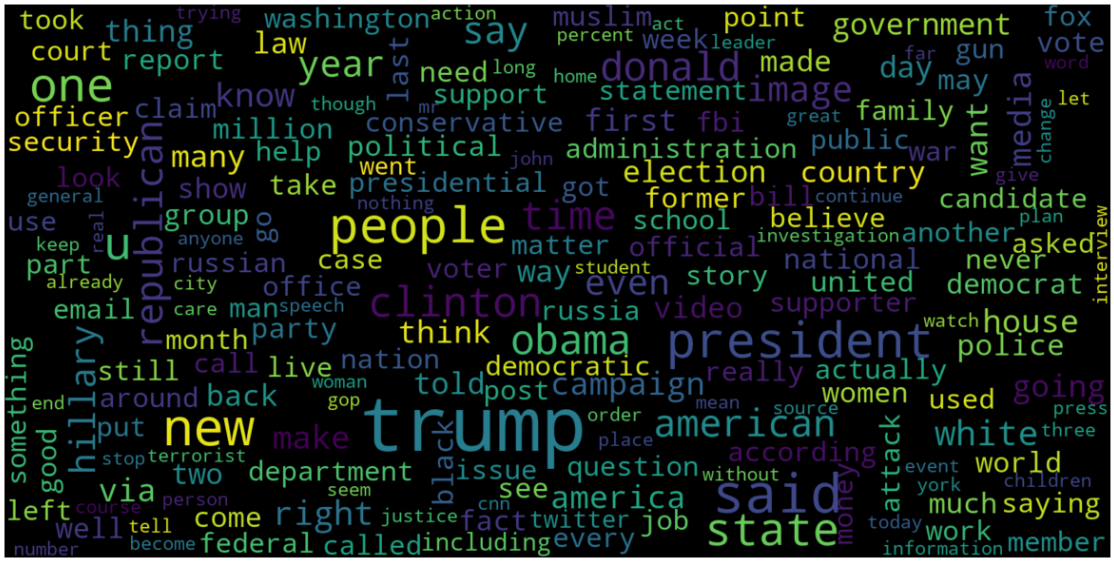
[13]:
```
preprocessed_review = preprocess_text(data['text'].values)
data['text'] = preprocessed_review
```

```
100%|      | 44919/44919 [1:19:00<00:00,  9.48it/s]
```

[14]:
```
# Real
consolidated = ' '.join(
        word for word in data['text'][data['class'] == 1].astype(str))
wordCloud = WordCloud(width=1600,
                                      height=800,
                                      random_state=21,
                                      max_font_size=110,
                                      collocations=False)
plt.figure(figsize=(15, 10))
plt.imshow(wordCloud.generate(consolidated), interpolation='bilinear')
plt.axis('off')
plt.show()
```



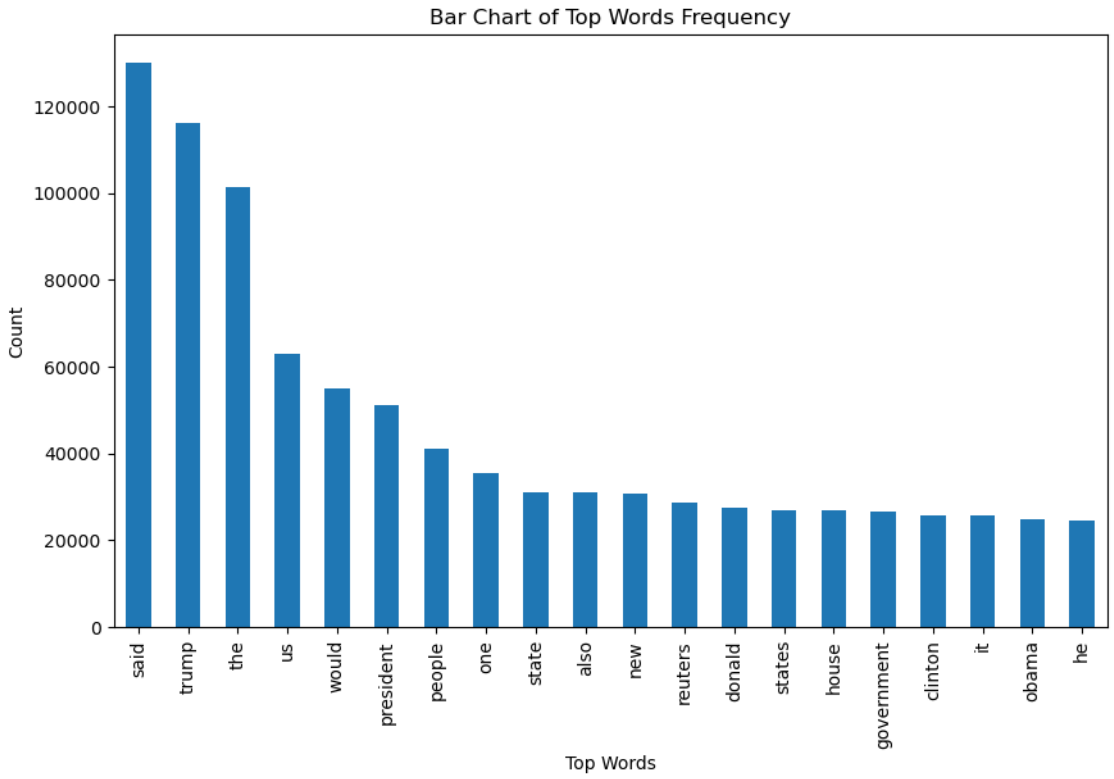
[15]:
```
# Fake
consolidated = ' '.join(
        word for word in data['text'][data['class'] == 0].astype(str))
wordCloud = WordCloud(width=1600,
                                        height=800,
                                        random_state=21,
                                        max_font_size=110,
                                        collocations=False)
plt.figure(figsize=(15, 10))
plt.imshow(wordCloud.generate(consolidated), interpolation='bilinear')
plt.axis('off')
plt.show()
```



[16]:
```
from sklearn.feature_extraction.text import CountVectorizer


def get_top_n_words(corpus, n=None):
        vec = CountVectorizer().fit(corpus)
        bag_of_words = vec.transform(corpus)
        sum_words = bag_of_words.sum(axis=0)
        words_freq = [(word, sum_words[0, idx])
                                for word, idx in vec.vocabulary_.items()]
        words_freq = sorted(words_freq, key=lambda x: x[1],
                                                reverse=True)
        return words_freq[:n]


common_words = get_top_n_words(data['text'], 20)
```

```
df1 = pd.DataFrame(common_words, columns=['Review', 'count'])

df1.groupby('Review').sum()['count'].sort_values(ascending=False).plot(
        kind='bar',
        figsize=(10, 6),
        xlabel="Top Words",
        ylabel="Count",
        title="Bar Chart of Top Words Frequency"
)
```

[16]: <Axes: title={'center': 'Bar Chart of Top Words Frequency'}, xlabel='Top Words', ylabel='Count'>



[17]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression

x_train, x_test, y_train, y_test = train_test_split(data['text'],


 ↪25)
```

[18]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorization = TfidfVectorizer()
x_train = vectorization.fit_transform(x_train)
x_test = vectorization.transform(x_test)
```

[19]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(x_train, y_train)

# testing the model
print(accuracy_score(y_train, model.predict(x_train)))
print(accuracy_score(y_test, model.predict(x_test)))
```

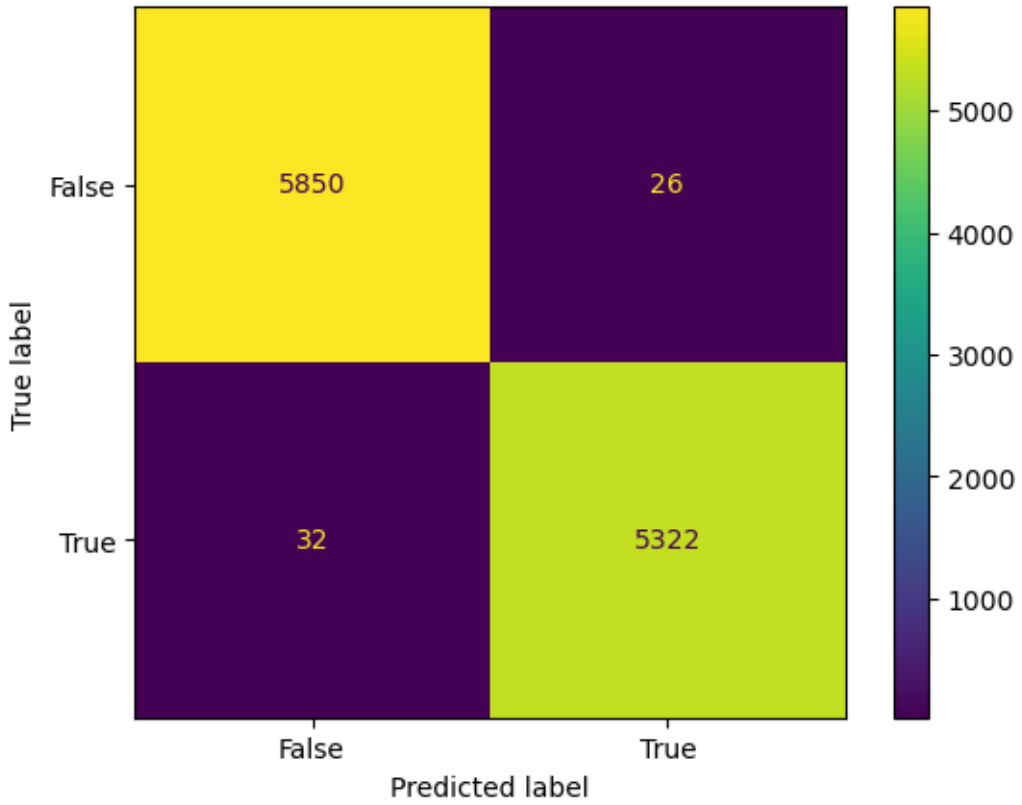
```
0.9937071447653537
0.990026714158504
```

[20]:
```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(x_train, y_train)

# testing the model
print(accuracy_score(y_train, model.predict(x_train)))
print(accuracy_score(y_test, model.predict(x_test)))
```

```
1.0
0.9948352626892253
```

[22]:
```
# Confusion matrix of Results from Decision Tree classification
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, model.predict(x_test))

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=cm,

 ↪True])

cm_display.plot()
plt.show()
```

displa



[47]:
```python
# Define the prediction function
from sklearn.tree import DecisionTreeClassifier

# Train the Decision Tree model
tree_model = DecisionTreeClassifier()
tree_model.fit(x_train, y_train)

def predict_news(input_text, model, vectorizer):
    # Preprocess the input text
    input_data = preprocess_text([input_text])

    # Transform the text using the fitted TfidfVectorizer
    input_vector = vectorizer.transform(input_data)

    # Make a prediction
    prediction = model.predict(input_vector)

    # Interpret the prediction
    if prediction[0] == 1:
        print("The news article is predicted to be: Real")
    else:
```

```
        print("The news article is predicted to be: Fake")

# Example usage with a trained model (Decision Tree model in this case)
input_text = input("Enter a news article text to check if it is Real or Fake:
 ↪\n")
predict_news(input_text, tree_model, vectorization)
```

```
Enter a news article text to check if it is Real or Fake:
 "BREAKING: Scientists confirm that the Earth is flat and all previous reports
were part of a government conspiracy."
100%|      | 1/1 [00:00<00:00, 211.58it/s]
The news article is predicted to be: Fake
```