FAKE NEWS DETECTION USING NLP

Fake News Datection

DESCRIPTION:The problem at hand is to develop a fake news detection model using a dataset obtained from Kaggle. The objective is to create a system that can effectively distinguish between genuine and fake news articles based on their titles and textual content. This project requires the utilization of Natural Language Processing (NLP) techniques to preprocess and transform the text data, building a machine learning model for classification, and subsequently evaluating the model's performance

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

DATA LOADING

```
import pandas as pd

# Load the dataset (replace 'path_to_fake_news_dataset.csv' with the
actual path to your dataset)
dataset_path = 'path_to_fake_news_dataset.csv'
fake_data = pd.read_csv(dataset_path)

# Display the first few rows of the dataset to inspect the data
fake_data.head()
```

Here,we need to replace 'path_to_fake_news_dataset.csv' with the actual file path where ve stored the dataset. This code will load the dataset into a Pandas DataFrame and display the first few rows to inspect the data.

TEXT PROCESSING

```
import nltk
from sklearn.preprocessing import LabelEncoder
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download NLTK data (if not already downloaded)
nltk.download('stopwords')
nltk.download('punkt')

# Remove missing values (if any)
fake_data = fake_data.dropna()

# Combine 'title' and 'text' columns for analysis
fake_data['text'] = fake_data['title'] + " " + fake_data['text']

# Tokenization and stop-word removal
stop_words = set(stopwords.words('english'))
fake_data['text'] = fake_data['text'].apply(lambda x: ' '.join([word
for word in word_tokenize(x) if word.lower() not in stop_words]))
```

```
# Label encoding (fake: 1, real: 0)
label_encoder = LabelEncoder()
fake_data['label'] = label_encoder.fit_transform(fake_data['label'])

# Display the preprocessed dataset
fake_data.head()
```

Removes any missing values in the dataset. Combines the 'title' and 'text' columns into a single 'text' column for analysis. Tokenizes the text using NLTK's word_tokenize function. Removes English stopwords from the tokenized text. Performs label encoding to convert 'fake' and 'real' labels to numeric values ('fake' as 1 and 'real' as 0).

FEATURE EXTRACTION

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Create a TF-IDF vectorizer with a maximum of 5000 features
tfidf_vectorizer = TfidfVectorizer(max_features=5000)

# Apply TF-IDF vectorization to the 'text' column
X = tfidf_vectorizer.fit_transform(fake_data['text'])

# Display the TF-IDF features
X.toarray()  # Converts the sparse matrix to a dense array for display (use carefully)

# Display the feature names
feature_names = tfidf_vectorizer.get_feature_names_out()
print("Example Feature Names:", feature_names[:10])
```

This code utilizes the TfidfVectorizer from scikit-learn to convert the text data into a TF-IDF feature matrix. The max_features parameter limits the number of features to 5000, but you can adjust this number as needed.

The resulting X contains the TF-IDF features for each document in your dataset. The feature names can be obtained using get_feature_names_out(). Please note that displaying the entire feature matrix (X.toarray()) may not be practical for a large dataset, so use it judiciously.

MODEL TRAINING

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,
fake_data['label'], test_size=0.2, random_state=42)

# Create a Multinomial Naive Bayes classifier
classifier = MultinomialNB()
```

```
# Train the classifier on the training data
classifier.fit(X_train, y_train)
```

In this code, we use the train_test_split function to split the TF-IDF features (X) and the labels (fake_data['label']) into training and testing sets. We then create a Multinomial Naive Bayes classifier (MultinomialNB) and train it using the training data.

MODEL EVALUATION

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Make predictions on the test set
y_pred = classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Generate confusion matrix
confusion_mat = confusion_matrix(y_test, y_pred)

# Generate classification report
classification_rep = classification_report(y_test, y_pred)

# Print evaluation results
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(confusion_mat)
print("Classification Report:")
print(classification_rep)

# Plot a confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues',
xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

In this code, we first make predictions on the test set using the trained Multinomial Naive Bayes classifier. Then, we calculate the accuracy of the model, generate a confusion matrix, and create a classification report. Finally, we print the results and plot the confusion matrix for visualization.