Name: Pavan Barhate

Report on Detecting Fake News Using Python

1. Introduction

Background:

In an era of rapid information dissemination through various media channels, the proliferation of fake news has become a significant concern. Fake news, characterized by misinformation or misleading content, can have profound implications on public perception, decision-making, and social harmony. Detecting and mitigating the impact of fake news is crucial for maintaining the integrity of information ecosystems.

Objective:

The primary objective of this project is to leverage machine learning techniques to develop a model capable of distinguishing between real and fake news. By harnessing the power of natural language processing and classification algorithms, we aim to create a tool that can assist in identifying potentially misleading information within textual content.

Dataset Overview:

The foundation of our analysis is a dataset comprising various features associated with news articles. These features include unique identifiers (ID), labels indicating the authenticity of the news (Label), textual content of statements (Statement), and additional information such as subject, speaker details, job, state, party affiliation, and counts of truthfulness categories.

Significance:

Detecting fake news is a multifaceted challenge with implications for media literacy, public awareness, and the credibility of information sources. This project aims to contribute to the ongoing efforts to combat misinformation by providing a data-driven approach to discerning the authenticity of news articles.

Approach:

Our approach involves utilizing Python programming language and machine learning libraries to preprocess textual data, build a predictive model, and evaluate its performance. Specifically, we employ the PassiveAggressiveClassifier algorithm in conjunction with TF-IDF (Term Frequency-Inverse Document Frequency) vectorization to transform text into numerical features for model training.

Structure of the Report:

The report is organized into sections that cover data exploration, preprocessing, model building, and evaluation. Each section plays a crucial role in understanding the dataset, preparing the data for analysis, constructing a robust model, and assessing its effectiveness in differentiating between real and fake news.

2. Data Exploration

2.1 Dataset Overview

Dataset Size:

Name: Pavan Barhate

- Begin by obtaining an understanding of the size of the dataset. This includes the total number of instances (rows) and features (columns) present in the dataset.

```
# Display the shape of the dataset

print("Dataset Shape:", train_data.shape)
```

Feature Descriptions:

- Provide a brief overview of each feature in the dataset, including the type of information it represents.

```
# Display feature names and their data types

print("Feature Information:")

print(train_data.dtypes)
```

2.2 Label Distribution

Class Distribution:

- Examine the distribution of labels (real vs. fake) to understand the balance between the classes.

```
# Display the distribution of labels

label_distribution = train_data['Label'].value_counts(normalize=True)

print("Label Distribution:")

print(label_distribution)
```

2.3 Text Data Exploration

Average Text Length:

- Calculate the average length of statements to get a sense of the typical statement length in the dataset.

```
# Calculate and display the average length of statements

average_length = train_data['Statement'].apply(len).mean()

print(f"Average Statement Length: {average_length:.2f} characters")
```

Common Words or Phrases:

- Identify common words or phrases in both real and fake news statements. This can be achieved through basic text processing techniques.

2.4 Additional Exploratory Analysis

Correlations:

- Explore potential correlations between features. For example, investigate if certain speakers or subjects are associated with a higher likelihood of fake news.

```
# Explore correlations between features
```

Name: Pavan Barhate

correlation_matrix = df.corr()

print("Correlation Matrix:")

print(correlation_matrix)

Data Summary Statistics:

- Display summary statistics for numerical features to understand their distribution.

Display summary statistics for numerical features

summary_statistics = df.describe()

print("Summary Statistics:")

print(summary_statistics)

Conclusion of Data Exploration:

- Summarize key findings from the data exploration phase, such as any notable imbalances in class distribution, the presence of outliers, or patterns observed in the data.

  This detailed data exploration provides a foundation for subsequent steps in the project, guiding decisions related to data preprocessing, feature engineering, and model selection. It helps to ensure that the analysis is grounded in a thorough understanding of the dataset's characteristics and nuances.

Python Code:

import pandas as pd

import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.linear_model import PassiveAggressiveClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import itertools

import seaborn as sns

import matplotlib.pyplot as plt

train_data = pd.read_csv('train.csv', header=None, names=['ID', 'Label', 'Statement', 'Subject', 'Speaker', 'Job', 'State', 'Party', 'Barely True Counts', 'False Counts', 'Half True Counts', 'Mostly True Counts', 'Pants on Fire Counts', 'Context'])

train_data.head(10)

train_data.info()

train_data.shape

Split the data into training and testing sets

Name: Pavan Barhate

```
X_train, X_test, y_train, y_test = train_test_split(train_data['Statement'], train_data['Label'], test_size=0.2, random_state=42)

# Initialize TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)

# Fit and transform the training data

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Initialize PassiveAggressiveClassifier

pac_classifier = PassiveAggressiveClassifier(max_iter=50)

# Fit the model

pac_classifier.fit(X_train_tfidf, y_train)

# Make predictions

y_pred = pac_classifier.predict(X_test_tfidf)

# Step 3: Create a confusion matrix to evaluate the model's performance

conf_matrix = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(conf_matrix)

# Step 4: Measure the model's accuracy

accuracy = accuracy_score(y_test, y_pred)

accuracy*100

from sklearn.model_selection import cross_val_score

# Perform 5-fold cross-validation

cv_scores = cross_val_score(pac_classifier, X_train_tfidf, y_train, cv=5)

# Display cross-validation scores

print("Cross-Validation Scores:", cv_scores)

print("Mean Accuracy: ", cv_scores.mean())

Report= classification_report(y_test, y_pred)

print(Report)

#Visualize the confusion matrix

sns.heatmap(conf_matrix, annot=True)

plt.show()

pac_classifier = PassiveAggressiveClassifier(max_iter=50, class_weight='balanced')
```

Name: Pavan Barhate

```
from sklearn.ensemble import RandomForestClassifier

# Replace PassiveAggressiveClassifier with RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
rf_classifier.fit(X_train_tfidf, y_train)

# Print feature importances
feature_importances = rf_classifier.feature_importances_
feature_names = tfidf_vectorizer.get_feature_names_out()
important_features = pd.DataFrame(data={'Feature': feature_names, 'Importance': feature_importances})
important_features = important_features.sort_values(by='Importance', ascending=False)
print(important_features.head(10))
```