Fake News Detection Using NLP and Machine Learning

Project Overview

This project aims to build a machine learning model that classifies news articles as **real or fake** using natural language processing (NLP). It demonstrates core data science skills including data cleaning, feature engineering, model training, and evaluation.

Objective

To detect fake news articles based on their content using a supervised learning approach.

Dataset

- **Source**: Fake and Real News Dataset on Kaggle
- **Files used**: Fake.csv and True.csv
- **Total records**: 44,898
  - Fake news: 23,481
  - Real news: 21,417
- **Columns**: title, text, subject, date

🛠 Tools & Technologies

- **Platform**: Google Colab
- **Language**: Python
- **Libraries**:
  - Data Handling: pandas, numpy
  - NLP: nltk
  - Modeling: scikit-learn
  - Visualization: matplotlib, seaborn

🔍 Step-by-Step Methodology

1️⃣ Environment Setup

python

import pandas as pd

import numpy as np

```
import nltk
import matplotlib.pyplot as plt
import seaborn as sns

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

nltk.download('stopwords')
nltk.download('wordnet')
```

2️⃣ Data Loading & Labeling

python

```
# Load datasets
fake_df = pd.read_csv("Fake.csv")
real_df = pd.read_csv("True.csv")

# Add labels
fake_df["label"] = 0
real_df["label"] = 1

# Combine datasets
df = pd.concat([fake_df, real_df], ignore_index=True)
```

3️⃣ Text Preprocessing

python

```
stop_words = set(stopwords.words('english'))
```

```
lemmatizer = WordNetLemmatizer()


def clean_text(text):
    tokens = text.lower().split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)


df['clean_text'] = df['text'].apply(clean_text)
```

4️⃣ Feature Extraction

python

```
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['clean_text'])
y = df['label']
```

5️⃣ Model Training

python

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


model = LogisticRegression()
model.fit(X_train, y_train)
```

6️⃣ Model Evaluation

python

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

7️⃣ Confusion Matrix Visualization

python

```
cm = confusion_matrix(y_test, y_pred)


plt.figure(figsize=(6,4))
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

📈 **Results**

- The model showed strong performance with high precision and recall.
- Confusion matrix confirmed effective classification of both fake and real news.

Interview Talking Points (With Detailed Answers)

🔹 Q1: Why did you choose Logistic Regression?

Answer: Logistic Regression is a strong baseline for binary classification tasks. It's fast, interpretable, and works well with high-dimensional sparse data like TF-IDF vectors. Since our goal was to classify news as real or fake, it provided a reliable starting point to benchmark performance.

🔹 Q2: How did you preprocess the text?

Answer: I used NLTK to:

- Lowercase all text
- Remove stopwords (e.g., "the", "is", "and")
- Lemmatize words to reduce them to their base form (e.g., "running" → "run")

This helped reduce noise and improve the quality of features extracted during vectorization.

🔹 Q3: Why did you use TF-IDF instead of Bag of Words?

Answer: TF-IDF captures the importance of words relative to the entire corpus. Unlike Bag of Words, which treats all words equally, TF-IDF downweights common words and highlights unique terms. This improves model performance by focusing on meaningful patterns in the text.

🔹 Q4: How did you evaluate your model?

Answer: I used:

- **Classification Report**: To measure precision, recall, F1-score, and accuracy.

- **Confusion Matrix**: To visualize true positives, false positives, etc.

These metrics helped me understand how well the model distinguishes between fake and real news.

🔹 **Q5: What challenges did you face?**

Answer:

- Balancing the dataset: Fake and real news had slightly different counts.
- Text preprocessing: Ensuring clean and meaningful input for the model.
- Feature selection: Limiting TF-IDF to 5000 features to avoid overfitting.

🔹 **Q6: How would you improve this project?**

Answer:

- Try deep learning models like BERT for better semantic understanding.
- Add real-time scraping to classify live news.
- Deploy the model as a web app for public use.
- Use cross-validation for more robust evaluation.

🔹 **Q7: What did you learn from this project?**

Answer: I learned how to:

- Handle and clean real-world text data
- Apply NLP techniques for feature extraction
- Train and evaluate machine learning models
- Visualize results and interpret performance metrics
- Think critically about model improvements and deployment