# Fake News Detector Project Report

---

## Table of Contents



---

Abhishek Chourasia

# 1. Project Overview

The Fake News Detector project is an end-to-end solution that uses machine learning and deep learning techniques to classify news articles as either "REAL" or "FAKE." The system provides:

- **Real-Time Predictions:** Users paste news text into a web form, and the system returns predictions along with confidence percentages.
- **News Headlines:** The application dynamically retrieves and displays the top news headlines from India via an external API.
- **Responsive User Interface:** A modern front-end built with HTML, CSS, and JavaScript ensures a smooth user experience on both mobile and laptop devices.
- **Local Deployment:** The entire application is deployed on a local host using Flask, making it ideal for development, testing, and future enhancements.

---

# 2. Dataset and Model Development

### Data Collection and Pre-processing

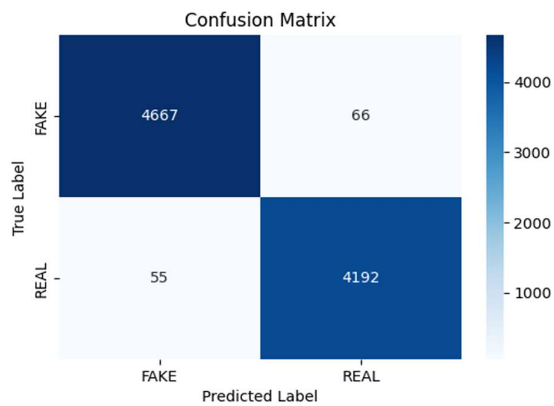
- **Dataset Sources:**
  Two datasets are used:
  - A dataset containing fake news articles.
  - A dataset containing true news articles.
- **Pre-processing Steps:**
  - **Labelling:** Each dataset is assigned a label ("FAKE" for fake news and "REAL" for true news).
  - **Combining Data:** The datasets are merged into a single data frame.
  - **Feature Extraction:** The text column is used as the feature, with an optional combination of title and text. A TF-IDF vectorizer (excluding English stop words) is used to convert the text into numerical features.

### Model Training and Evaluation

- **Data Splitting:** The data is split into training and testing sets using an 80/20 ratio.
- **Vectorization:** The TF-IDF vectorizer transforms the text data into a numerical format.
- **Classifier:** A Logistic Regression classifier (with a maximum of 1000 iterations) is trained on the training data.
- **Evaluation:** The model is evaluated using accuracy on the test set. The accuracy score is printed to verify model performance.
- **Model Persistence:** The trained model is saved as model_updated.pkl and the TF-IDF vectorizer as vectorizer_updated.pkl using the Python pickle module.

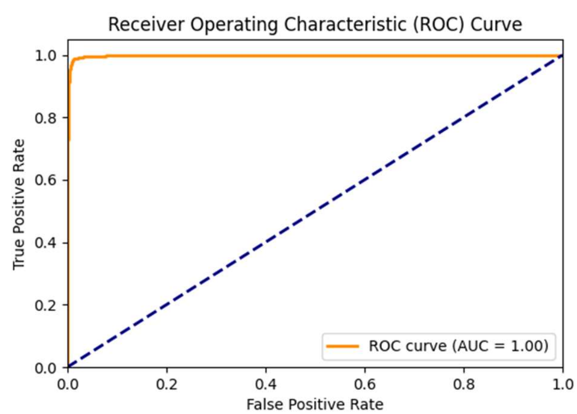
### Data Visualization (DV)

1. **Confusion Matrix:**
   - **Description:** A heatmap that displays the number of correct and incorrect predictions for each class (FAKE and REAL).
   - **Insight:** This visualization helps in understanding the performance of the classifier by showing how many instances were misclassified.



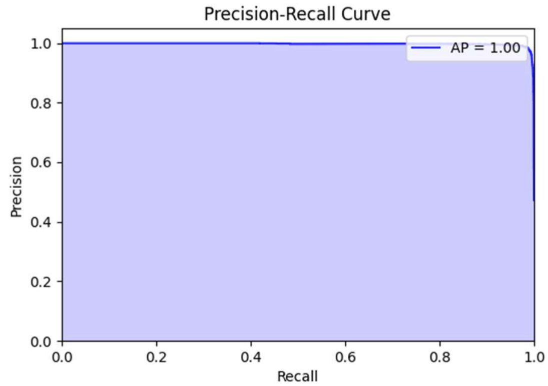

2. **ROC Curve (Receiver Operating Characteristic Curve):**

- **Description:** A plot of the True Positive Rate (TPR) versus the False Positive Rate (FPR) at various threshold settings. The area under the curve (AUC) is also displayed.
- **Insight:** This curve shows the model's ability to distinguish between the classes. A higher AUC indicates better performance.



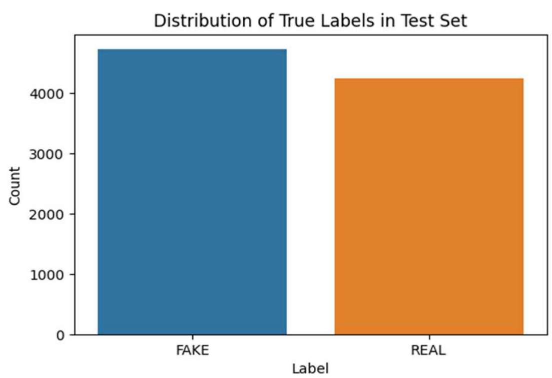

3. **Precision-Recall Curve:**

- **Description:** A plot that illustrates the trade-off between precision and recall for different threshold settings. The average precision (AP) score is included.
- **Insight:** This curve is particularly useful for imbalanced datasets, showing how precision varies with recall.



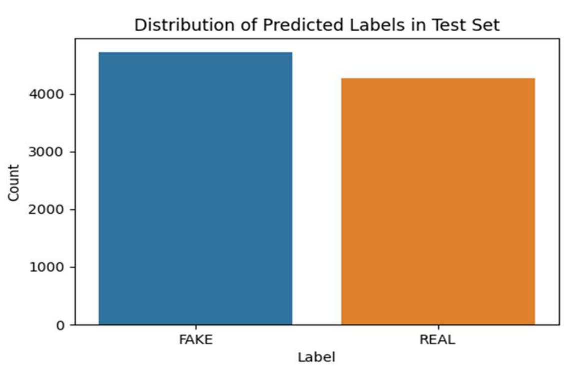

4. **Distribution of True Labels:**

- **Description:** A bar chart showing the count of each label (FAKE and REAL) in the test set.
- **Insight:** This visualization helps verify that the dataset is balanced (or imbalanced) and provides context for interpreting other performance metrics.




5. **Distribution of Predicted Labels:**

- **Description:** A bar chart displaying the count of each label predicted by the model on the test set.
- **Insight:** Comparing this with the true label distribution helps in assessing whether the model is biased toward one class.



# 3. Application Architecture and Implementation

### Backend – Flask Application

- **Flask Routes:**
  - **/ Route:** Renders the main HTML page.
  - **/api/predict Route:** Accepts POST requests with news text, transforms the text using the TF-IDF vectorizer, and predicts whether the news is "REAL" or "FAKE." The response includes the prediction along with confidence percentages.
  - **/api/news Route:** Fetches the top headlines from the NewsData.io API (for Indian news) and caches the results for one hour to minimize API calls.
- **Model Loading:**
  At startup, the Flask application loads model_updated.pkl and vectorizer_updated.pkl so that predictions can be served immediately.

### Frontend – HTML, CSS, and JavaScript

- **User Interface:**
  The front-end is composed of several sections:
  - **Hero Section:** Features a dynamic video background (or image) for an engaging introduction.
  - **Fake News Detector Section:** Includes a live clock, a text area for input, a submit button, a display for prediction results (with animated progress bars), and a history log.
  - **Headlines Section:** Dynamically displays top news headlines fetched from the backend.
  - **Why Detector Section:** Explains the importance of detecting fake news.
- **Interactivity:**
  JavaScript manages:
  - Debouncing user input to prevent overwhelming the server.
  - Sending the input to the backend and updating the UI with predictions.
  - Periodically refreshing the news headlines.

---

# 4. Deployment on Local Host

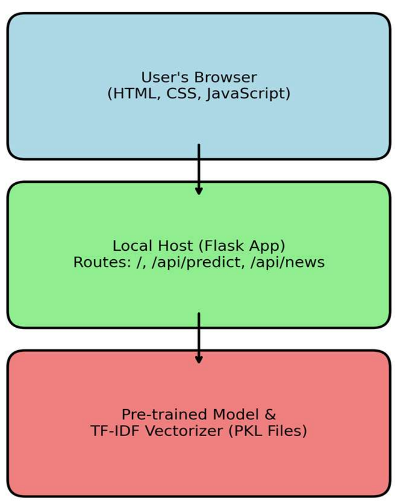
**Steps to Deploy:**

1. **Environment Setup:**
   - Install Python and required libraries (Flask, requests, scikit-learn, etc.).
   - Ensure all project files (including app.py, index.html, model pickle files, and static assets) are in the appropriate directories.
2. **Running the Application:**
   - Open a terminal in the project folder.
   - Run the Flask server with:

     bash
     CopyEdit
     ```
     python app.py
     ```
   - The application will run at http://localhost:5000.
3. **Accessing the Application:**
   - Open your web browser and navigate to http://localhost:5000 to use the Fake News Detector.

---

## 5. System Architecture Diagram




---

# 6. Conclusion and Future Enhancements

**Conclusion:**

This project demonstrates an end-to-end Fake News Detector system that:

- Processes and vectorizes news text.
- Uses a Logistic Regression model to classify news as "REAL" or "FAKE."
- Provides real-time predictions and confidence scores.
- Displays dynamic news headlines and an engaging user interface.
- Is deployed locally via Flask for easy development and testing.

**Future Enhancements:**

- **Model Optimization:** Experiment with more sophisticated machine learning or deep learning models.
- **Scalability:** Containerize the application using Docker and deploy on cloud platforms.
- **Enhanced UI/UX:** Further improve the front-end experience with modern frameworks (React, Vue.js, etc.).
- **Data Persistence:** Implement a database for logging predictions and user interactions.
- **User Feedback:** Add mechanisms for users to provide feedback on predictions to continually refine the model.

---

# 7. Reference: Model Training Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pickle
import matplotlib.pyplot as plt
import seaborn as sns

# ------------------------------
# Data Loading and Preprocessing
# ------------------------------
# Load the datasets
fake = pd.read_csv('/kaggle/input/fake-news-detection/fake.csv')
real = pd.read_csv('/kaggle/input/fake-news-detection/true.csv')

# Add a label column to each dataframe
fake['label'] = 'FAKE'
real['label'] = 'REAL'

# Combine both datasets into one dataframe
data = pd.concat([fake, real], ignore_index=True)

# (Optional) Combine title and text if desired:
# data['text'] = data['title'] + " " + data['text']

# Use only the 'text' column as features and 'label' as target
X = data['text']
y = data['label']

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Initialize the TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# ------------------------------
# Model Training and Evaluation
# ------------------------------
# Train a Logistic Regression classifier
model = LogisticRegression(max_iter=1000)
model.fit(X_train_tfidf, y_train)

# Evaluate the model
predictions = model.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, predictions)
print("Model Accuracy:", accuracy)

# Print the Classification Report
```

```
report = classification_report(y_test, predictions, target_names=["FAKE", "REAL"])
print("Classification Report:")
print(report)

# ------------------------------
# Visualization 1: Confusion Matrix
# ------------------------------
cm = confusion_matrix(y_test, predictions, labels=["FAKE", "REAL"])
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["FAKE", "REAL"],
            yticklabels=["FAKE", "REAL"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.savefig("confusion_matrix.png")  # Save image for download
plt.show()

# ------------------------------
# Visualization 2: ROC Curve
# ------------------------------
from sklearn.metrics import roc_curve, auc

# Convert labels to binary (assuming REAL is the positive class)
y_test_binary = (y_test == "REAL").astype(int)
# Get predicted probabilities for the positive class ("REAL")
proba = model.predict_proba(X_test_tfidf)
# Find the index of "REAL" in model.classes_
real_index = list(model.classes_).index("REAL")
y_proba = proba[:, real_index]

fpr, tpr, thresholds = roc_curve(y_test_binary, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:0.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.savefig("roc_curve.png")  # Save image for download
plt.show()

# ------------------------------
# Visualization 3: Precision-Recall Curve
# ------------------------------
from sklearn.metrics import precision_recall_curve, average_precision_score

precision, recall, thresholds_pr = precision_recall_curve(y_test_binary, y_proba)
avg_precision = average_precision_score(y_test_binary, y_proba)

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

```
plt.step(recall, precision, where='post', color='b', alpha=0.8, label=f'AP = {avg_precision:0.2f}')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall Curve')
plt.legend(loc="upper right")
plt.savefig("precision_recall_curve.png")  # Save image for download
plt.show()

# -------------------------------
# Visualization 4: Distribution of True Labels
# -------------------------------
plt.figure(figsize=(6, 4))
sns.countplot(x=y_test)
plt.title("Distribution of True Labels in Test Set")
plt.xlabel("Label")
plt.ylabel("Count")
plt.savefig("true_label_distribution.png")  # Save image for download
plt.show()

# -------------------------------
# Visualization 5: Distribution of Predicted Labels
# -------------------------------
plt.figure(figsize=(6, 4))
sns.countplot(x=predictions)
plt.title("Distribution of Predicted Labels in Test Set")
plt.xlabel("Label")
plt.ylabel("Count")
plt.savefig("predicted_label_distribution.png")  # Save image for download
plt.show()

# -------------------------------
# Save the trained model and vectorizer
# -------------------------------
with open('model_updated.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)
with open('vectorizer_updated.pkl', 'wb') as vec_file:
    pickle.dump(vectorizer, vec_file)
```

---

**End of Report**