Fake News Detection using Machine Learning

Overview

This project aims to build a web application that can predict whether a given news article is **FAKE** or **REAL** using **Natural Language Processing (NLP)** and **Machine Learning**. The model processes text input, extracts relevant features, and classifies it based on pre-trained data.

Step-by-Step Breakdown

Step 1: Understanding the Problem

Fake news refers to false information disseminated through various media channels. The goal is to develop a machine learning model that classifies news articles as FAKE or REAL based solely on the text content.

Step 2: Importing Required Libraries

Several Python libraries are needed for data processing, model training, and web development. Here is the code snippet:

python

CopyEdit

```
# Data Handling
import pandas as pd
import numpy as np

# Text Processing
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer

# Machine Learning
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


# Save Model

import pickle
```

Step 3: Load and Preprocess the Dataset

1. **Loading the Dataset:**
   Load the dataset containing news articles and their labels (0 for FAKE and 1 for REAL).

```
python

CopyEdit

# Load dataset (Replace with your dataset file)

df = pd.read_csv("news_dataset.csv")

df.head()
```

2. **Data Cleaning & Text Preprocessing:**
   Clean the text data by removing special characters, converting to lowercase, tokenizing, removing stopwords, and applying stemming.

```
python

CopyEdit

nltk.download('stopwords')

ps = PorterStemmer()


def clean_text(text):

    # Remove special characters and convert to lowercase

    text = re.sub('[^a-zA-Z]', ' ', text).lower()

    # Tokenization

    words = text.split()

    # Remove stopwords and apply stemming

    words = [ps.stem(word) for word in words if word not in
stopwords.words('english')]

    return ' '.join(words)
```

Apply preprocessing function

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

Step 4: Feature Extraction Using TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) converts text into numerical vectors for the machine learning model.

python

CopyEdit

vectorizer = TfidfVectorizer(max_features=5000)

X = vectorizer.fit_transform(df['cleaned_text']).toarray()

y = df['label'] # Target variable (0 = Fake, 1 = Real)

Step 5: Train the Machine Learning Model

Split the data into training and testing sets, and train a Logistic Regression model.

python

CopyEdit

Split dataset (80% Train, 20% Test)

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

Train Logistic Regression Model

model = LogisticRegression()

model.fit(X_train, y_train)

Test the model

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy * 100:.2f}%")

Step 6: Save the Model and Vectorizer

Use pickle to save the trained model and vectorizer, so they can be loaded later in the Flask application.

python

CopyEdit

pickle.dump(model, open("model2.pkl", "wb"))

pickle.dump(vectorizer, open("tfidfvect2.pkl", "wb"))

Step 7: Build a Flask Web Application

Create a Flask web app (app.py) to allow user interaction with the model. The app takes user input, processes the text, and outputs a prediction.

python

CopyEdit

```
from flask import Flask, render_template, request, jsonify
import pickle
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

# Initialize Flask App
app = Flask(__name__)

# Load trained model and vectorizer
model = pickle.load(open("model2.pkl", "rb"))
vectorizer = pickle.load(open("tfidfvect2.pkl", "rb"))

ps = PorterStemmer()

# Function to preprocess input text
def preprocess_text(text):
    text = re.sub('[^a-zA-Z]', ' ', text).lower()
    words = text.split()
```

```
    words = [ps.stem(word) for word in words if word not in
stopwords.words('english')]
    return ' '.join(words)


# Web Interface
@app.route("/", methods=["GET", "POST"])
def home():
    if request.method == "POST":
        text = request.form["text"]
        cleaned_text = preprocess_text(text)
        text_vectorized = vectorizer.transform([cleaned_text]).toarray()
        prediction = model.predict(text_vectorized)
        result = "FAKE" if prediction == 0 else "REAL"
        return render_template("index.html", text=text, result=result)
    return render_template("index.html")


# API Endpoint
@app.route("/predict/", methods=["GET"])
def predict():
    text = request.args.get("text")
    cleaned_text = preprocess_text(text)
    text_vectorized = vectorizer.transform([cleaned_text]).toarray()
    prediction = model.predict(text_vectorized)
    return jsonify({"prediction": "FAKE" if prediction == 0 else "REAL"})


# Run Flask App
if __name__ == "__main__":
    app.run(debug=True)
```

Step 8: Install Dependencies

List all required libraries in a requirements.txt file:

ini

CopyEdit

Flask==1.1.1

nltk==3.4.5

numpy==1.13.3

pandas==0.25.1

scikit-learn==0.23.1

gunicorn==20.0.4

Install the dependencies using the following command:

sh

CopyEdit

pip install -r requirements.txt

Step 9: Deploy the Web App

Deploy the Flask application using a WSGI server like Gunicorn on platforms such as Heroku or AWS. For example:

sh

CopyEdit

gunicorn -w 4 app:app

Conclusion

This project demonstrates how to build a machine learning model to detect fake news by processing textual data, extracting features using TF-IDF, and deploying the model in a Flask-based web application. The complete workflow covers data cleaning, model training, saving the model, and creating a user-friendly interface to obtain predictions.