**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()

1. **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.