from google.colab import files

uploaded = files.upload()

from zipfile import ZipFile

# Unzip fake.csv

with ZipFile("fake.csv (1).zip", 'r') as zip\_ref:

    zip\_ref.extractall("/content/fake\_news")

# Unzip true.csv

with ZipFile("true.csv.zip", 'r') as zip\_ref:

    zip\_ref.extractall("/content/true\_news")

import pandas as pd

# Load the CSVs

fake\_df = pd.read\_csv('/content/fake\_news/fake.csv')

true\_df = pd.read\_csv('/content/true\_news/true.csv')

# Add labels: 1 = Fake, 0 = Real

fake\_df['label'] = 1

true\_df['label'] = 0

# Combine datasets

data = pd.concat([fake\_df, true\_df], ignore\_index=True)

# Shuffle the dataset

data = data.sample(frac=1, random\_state=42).reset\_index(drop=True)

# Check basic info

print("Shape of dataset:", data.shape)

data.head()

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

# Download NLTK resources

nltk.download('stopwords')

nltk.download('wordnet')

nltk.download('omw-1.4')

stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

# Function to clean text

def clean\_text(text):

    text = str(text).lower()

    text = re.sub(r'[^a-zA-Z\s]', '', text)       # Remove punctuation and numbers

    tokens = text.split()

    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop\_words and len(word) > 2]

    return ' '.join(tokens)

# Apply to the text column (combine title + text if both exist)

if 'title' in data.columns and 'text' in data.columns:

    data['content'] = data['title'].fillna('') + " " + data['text'].fillna('')

else:

    data['content'] = data['text']

data['clean\_content'] = data['content'].apply(clean\_text)

# Preview cleaned data

data[['label', 'clean\_content']].head()

from sklearn.feature\_extraction.text import TfidfVectorizer

# TF-IDF Vectorization

tfidf = TfidfVectorizer(max\_features=5000)

X = tfidf.fit\_transform(data['clean\_content']).toarray()

# Labels

y = data['label'].values

print("TF-IDF shape:", X.shape)

import tldextract

# Example: A simple source credibility scoring function

# You can improve this with more sophisticated logic based on your dataset

def extract\_source(text):

    # Try to extract the domain from the source text

    try:

        extracted = tldextract.extract(text)

        return extracted.domain

    except:

        return ""

def assign\_credibility\_score(domain):

    credible\_sources = ['bbc', 'reuters', 'nytimes', 'guardian', 'cnn']

    if any(cred in domain for cred in credible\_sources):

        return 1  # Credible source

    else:

        return 0  # Non-credible source

# Assuming 'source' column exists, if not we simulate with domain info.

data['source'] = data['content'].apply(lambda x: extract\_source(x))

# Apply credibility scoring

data['credibility'] = data['source'].apply(assign\_credibility\_score)

# Preview the data with added source and credibility

data[['label', 'content', 'source', 'credibility']].head()

!pip install tldextract

import tensorflow as tf

from tensorflow.keras import layers, models

from sklearn.model\_selection import train\_test\_split

# Set random seed for reproducibility

tf.random.set\_seed(42)

# Split data 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)

# Generator Model (Fake News Generator)

def build\_generator(input\_dim):

    model = models.Sequential()

    model.add(layers.Dense(128, activation='relu', input\_dim=input\_dim))

    model.add(layers.Dense(256, activation='relu'))

    model.add(layers.Dense(512, activation='relu'))

    model.add(layers.Dense(input\_dim, activation='sigmoid'))

    return model

# Discriminator Model (Fake News Classifier)

def build\_discriminator(input\_dim):

    model = models.Sequential()

    model.add(layers.Dense(512, activation='relu', input\_dim=input\_dim))

    model.add(layers.Dense(256, activation='relu'))

    model.add(layers.Dense(128, activation='relu'))

    model.add(layers.Dense(1, activation='sigmoid'))

    return model

# Build GAN model (combined Generator + Discriminator)

def build\_gan(generator, discriminator):

    discriminator.trainable = False

model = models.Sequential()

    model.add(generator)

    model.add(discriminator)

    return model

# Initialize generator and discriminator

generator = build\_generator(input\_dim=X\_train.shape[1])

discriminator = build\_discriminator(input\_dim=X\_train.shape[1])

# Compile discriminator

discriminator.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Build and compile GAN model

gan = build\_gan(generator, discriminator)

gan.compile(optimizer='adam', loss='binary\_crossentropy')

# Summary of the models

generator.summary()

discriminator.summary()

import numpy as np

# Training function

def train\_gan(generator, discriminator, gan, X\_train, y\_train, epochs=1000, batch\_size=64):

    for epoch in range(epochs):

        # 1. Train the Discriminator

        idx = np.random.randint(0, X\_train.shape[0], batch\_size)

        real\_data = X\_train[idx]

        real\_labels = y\_train[idx]

        fake\_data = generator.predict(np.random.randn(batch\_size, X\_train.shape[1]))

        fake\_labels = np.zeros(batch\_size)  # Fake news has label 0

        d\_loss\_real = discriminator.train\_on\_batch(real\_data, real\_labels)

        d\_loss\_fake = discriminator.train\_on\_batch(fake\_data, fake\_labels)

        d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

        # 2. Train the Generator

        noise = np.random.randn(batch\_size, X\_train.shape[1])

        valid\_labels = np.ones(batch\_size)  # Generator tries to make fake data appear rea

        g\_loss = gan.train\_on\_batch(noise, valid\_labels)

        if epoch % 100 == 0:

            print(f"{epoch} [D loss: {d\_loss[0]} | D accuracy: {100 \* d\_loss[1]}] [G loss: {g\_loss}]")

# Start training the GAN

train\_gan(generator, discriminator, gan, X\_train, y\_train, epochs=1000, batch\_size=64)

# Assuming we have a credibility score function for each article

def get\_credibility\_score(article):

    # Simple example: You can expand this with more sophisticated logic

    # If the source is from a high-credibility domain (example: cnn.com, bbc.com, etc.), assign a high score

    high\_cred\_sources = ['cnn.com', 'bbc.com', 'nytimes.com']

    # Example: Extract the source domain from the URL (you can also use other source metadata)

    domain = tldextract.extract(article)

    source = domain.domain + '.' + domain.suffix

    if source in high\_cred\_sources:

        return 1  # High credibility

    else:

        return 0  # Low credibility

# Example final prediction logic combining both

def final\_prediction(discriminator\_output, article):

    credibility\_score = get\_credibility\_score(article)

    # Weights: Give more weight to content if the source is credible

    if credibility\_score == 1:

        final\_decision = discriminator\_output \* 0.7 + credibility\_score \* 0.3  # More focus on discriminator for credible sources

    else:

        final\_decision = discriminator\_output \* 0.5 + credibility\_score \* 0.5  # Equal weight for fake and non-credible sources

    # If final decision is closer to 1, classify as real (real news), if closer to 0, classify as fake

    return 1 if final\_decision > 0.5 else 0

# Example: Using the trained discriminator to predict and then combining with SCS

def predict\_fake\_news\_with\_scs(article\_text):

    # Step 1: Use the discriminator model to classify the article as real (1) or fake (0)

    article\_vector = tfidf.transform([article\_text]).toarray()  # Transform the text to TF-IDF features

    discriminator\_output = discriminator.predict(article\_vector)

    # Step 2: Combine with SCS to make final decision

    final\_decision = final\_prediction(discriminator\_output, article\_text)

    return final\_decision

# Test with an example article

example\_article = "This is a sample news article about recent political developments."

prediction = predict\_fake\_news\_with\_scs(example\_article)

print("Final Prediction (1 = Real, 0 = Fake):", prediction)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, stratify=y, random\_state=42

)

print("Train:", sum(y\_train==1), "Fake,", sum(y\_train==0), "Real")

print("Test :", sum(y\_test==1), "Fake,", sum(y\_test==0), "Real")

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

# Train the classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

clf.fit(X\_train, y\_train)

# Predict and evaluate

y\_pred = clf.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

import joblib

joblib.dump(clf, 'fake\_news\_model.pkl')

!pip install lime --quiet

from lime.lime\_text import LimeTextExplainer

import numpy as np

# Create a prediction pipeline

class\_names = ['Real', 'Fake']

# Since we used TF-IDF to train, we need the vectorizer and classifier

def predict\_proba(texts):

    vectors = tfidf.transform(texts)

    return clf.predict\_proba(vectors)

explainer = LimeTextExplainer(class\_names=class\_names)

# Example: Predict and explain one article

sample\_text = "Breaking: The government has confirmed a UFO landing in Nevada."

# Show prediction

pred = clf.predict(tfidf.transform([sample\_text]))[0]

print(f"Prediction: {'Fake' if pred == 1 else 'Real'}")

# Explanation

exp = explainer.explain\_instance(sample\_text, predict\_proba, num\_features=10)

exp.show\_in\_notebook(text=True)