INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN

PROGRAMMING WITH PYTHON
PROJECT FILE

Name - Sada Chouhan
Enrolment number - 16501172025.
Branch - CSE AI 3
Department - AI and DS

Faculty Name:-
Dr. Dipty Tripathi

Index

Introduction

The Problem: Fake News in the Digital Age

In today's world, we live in an era where information travels at the speed of internet literally. A single WhatsApp message, an Instagram post, or a Twitter thread can reach millions of people in minutes. But here's the catch: not all information is accurate.

Misinformation and fake news have become one of the biggest challenges facing democracies worldwide, and India is no exception. In fact, with over 400 million internet users and a highly engaged social media community, India faces unique and sometimes extreme challenges when it comes to false information.

Think about it: How many times have you seen a sensational headline on social media, clicked it, and realized halfway through reading that it's completely made up? Or received a forwarded message claiming something dramatic happened, only to find out later it was baseless? These aren't just annoying; they can have real-world consequences.

Real-World Impact of Fake News:

- Misinformation campaigns have sparked violence in communities based on false rumors.
- Fake health claims (especially post-COVID) have led people to avoid legitimate medical treatments.
- Business and financial scams disguised as credible news have caused financial losses to thousands.

Why This Project? A Technical Solution to a Social Problem

I decided to build Intelligent Fake News Detector because I believe technology can be part of the solution. But here's my approach: I didn't want to just throw one AI model at the problem and hope for the best. Instead, I combined:

1. Classical Machine Learning (SVM, Random Forest)—fast, interpretable, battle-tested.
2. Modern Deep Learning (BERT Transformers)—understands context, nuance, and meaning at a level humans find impressive.
3. Active Learning with User Feedback—the system doesn't just predict; it learns from its mistakes through user corrections.

System Requirements/Software

System Requirements and Softwares need to run and develop this project:

Hardware:

- Any halfway modern laptop or desktop (Mac, Windows, Linux).
- If you have 4GB+ RAM, you're set. With 8GB or more, BERT training runs much smoother.
- Disk space: 1-2GB free, mostly for news datasets and saved models.

Software & Libraries:

- Python 3.8+ (I used 3.13, but anything 3.8 or newer is solid).
- Streamlit: For the web interface—makes showcasing and testing models super simple.
- scikit-learn: Handles all the classic ML stuff, like SVM and text vectorization.
- PyTorch: The backend for BERT and other deep learning.
- Transformers (Hugging Face): Loads pre-trained models like DistilBERT, fine-tunes them on our dataset.
- pandas & numpy: For all the data wrangling and manipulation.
- matplotlib & seaborn: For graphs—confusion matrix, ROC curves, accuracy/recall scores, etc.
- joblib: For saving and loading trained ML models.
- sqlite3: No need to install separately; Python's built-in database handles user corrections.
- nltk: For handy language processing tools (stopwords, tokenization).

IDE/Tools: VS Code is my personal favorite, but PyCharm or Cursor work fine too

Install everything in one command:

pip install streamlit scikit-learn torch transformers pandas numpy matplotlib seaborn joblib nltk

System Workflow

A. **Data Pipeline**

- Sources: I collected a mix of Indian news datasets, including IFND, BharatFakeNewsKosh, and other publicly available CSVs.
- Merge & Clean: Used Python scripts to merge and clean the data, making sure columns like headline, body, label (fake/real) were standardized.
- Result: Ended up with a big file (combined_news.csv) with 6,335 labeled examples plenty of variety for our models.

B. **Feature Engineering**

- For traditional ML, I used TF-IDF vectorization. Basically, it turns words into numbers based on their importance in each document (so "scam" counts a lot when relevant).
- For BERT, I just let it handle tokenization with Hugging Face Transformers

C. **Model Training**

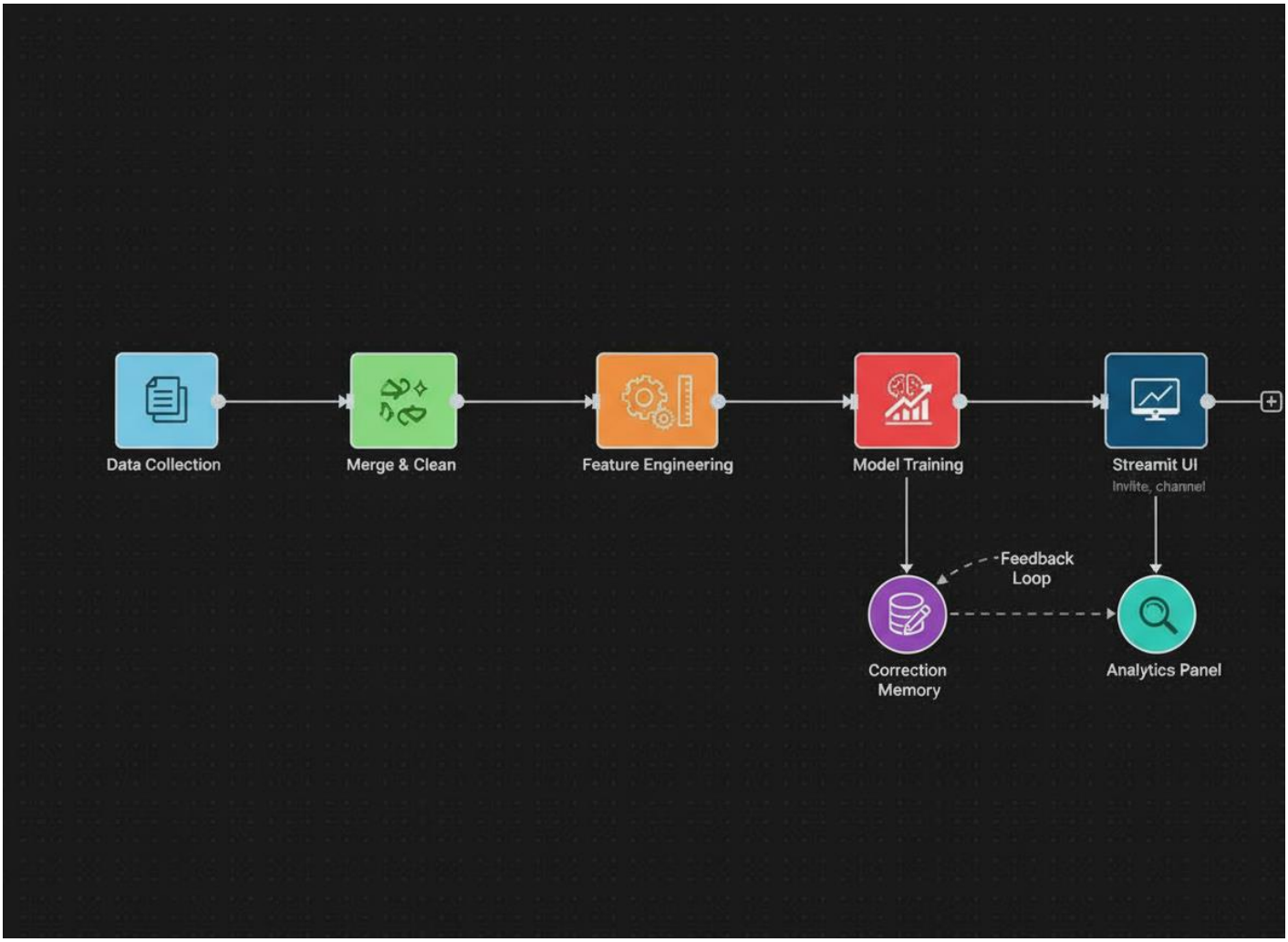
- SVM/Random Forest/Logistic Regression: Ran on TF-IDF features. SVM gave the highest accuracy (93.6%), which is honestly pretty good for ML
- Deep Learning/BERT: Used DistilBERT, fine-tuned it on our custom Indian news data. Required a few iterations (epochs) but managed to push test accuracy to 95.1%.
- Active Learning & Memory: Added a cool feature if a user corrects a prediction; it gets saved in SQLite. The system recognizes the correction for that news next time (almost like "learning by experience").

D. **User Interface**

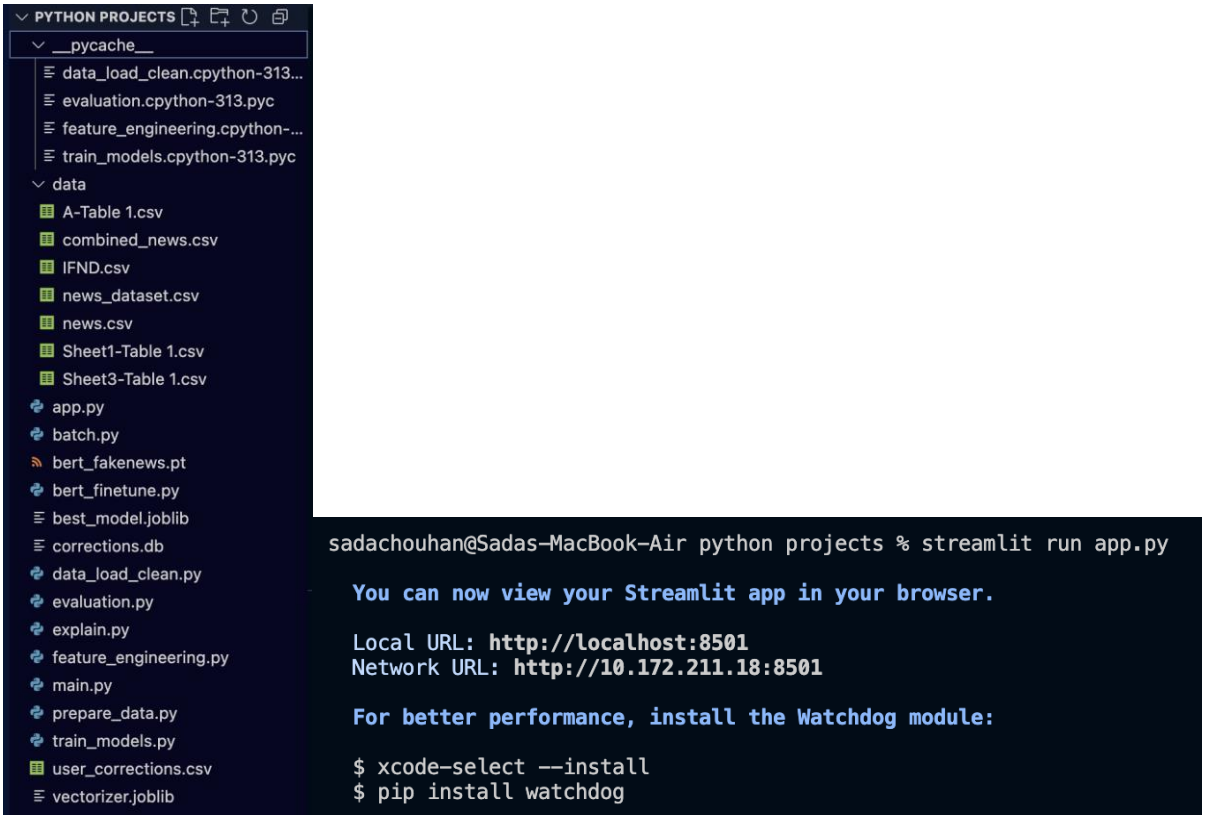
- Designed a Streamlit web app.
- You enter a headline and story; choose ML or BERT for prediction.
- If the model's wrong, click to correct it.
- See live analytics: confusion matrix, precision/recall, ROC curves get an idea of how the model's actually doing, not just a number.

E. **Feedback and Adaptation**

- Corrections aren't just stored they're actively used to override the models for those news items in future.
- Over time, as more corrections are stored, the entire system gets more accurate on repeated or edge-case headlines.
- Super useful for ongoing projects or even social media teams fighting fresh fake news.


Data Collection
Merge & Clean
Feature Engineering
Model Training
Streamlit UI
Invite, channel
Feedback
Loop
Correction
Memory
Analytics Panel

Project Code



Key Scripts & Their Roles:

- prepare_data.py: Handles all dataset merging and column matching.
- main.py / train_models.py: Trains ML models (SVM, RF, LR), selects the best, saves it for use.
- bert_finetune.py: Handles the heavy-lifting for BERT—fine-tunes transformer using PyTorch/Hugging Face.
- app.py: The heart and face of the project—runs the Streamlit server, shows the UI, gets predictions, saves user corrections (with SQLite).
- corrections.db: Gets updated every time a user corrects a label—makes the app "remember" mistakes and fixes.

Code Explanations:

- prepare_data.py:
  Automated column detection, merges all CSVs, handles everything from missing columns to cleaning nulls. Makes adding new datasets in future easy.
- main.py + train_models.py:
  Loads cleaned data. Splits it, vectorizes text. Runs SVM, RF, LR, and outputs test set accuracy. Saves the "best" model so the app uses it.

- bert_finetune.py:
  Uses Hugging Face's DistilBERT model. Tokenizes data, trains for ~3 epochs. Reports accuracy and other scores. Saves the model for inference in the app.
- app.py:
  Inputs go through correction check (SQLite), then ML or BERT. User can add a correction by click—future predictions for same story are instantly overridden. Analytics show live confusion matrix, ROC curve, etc.

The code is followed by the filename

- data_load_clean.py

```
import pandas as pd
import re
import nltk
import spacy

nltk.download('stopwords')
from nltk.corpus import stopwords

nlp = spacy.load("en_core_web_sm")



def clean_text(text):
text = str(text)
text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
doc = nlp(text.lower())
lemmatized = " ".join([token.lemma_ for token in doc if token.text not in stopwords.words('english')])
return lemmatized



def load_and_clean_data(filepath):
df = pd.read_csv(filepath)
df['combined_text'] = df['title'].astype(str) + " " + df['text'].astype(str)
df['clean_text'] = df['combined_text'].apply(clean_text)
return df
```

- main.py

```
import pandas as pd
from data_load_clean import load_and_clean_data
from feature_engineering import get_tfidf_features
from train_models import train_models, tune_model
from evaluation import evaluate_model
import joblib
from sklearn.feature_extraction.text import TfidfVectorizer

df = pd.read_csv('data/combined_news.csv')
X_raw = df['combined_text']
y = df['label']

vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(X_raw)
```

```
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, random_state=42)

# 4. Train models using train_models(X_train, y_train)
models, scores, splits = train_models(X_train, y_train)
best_name = max(scores, key=scores.get)
best_model = models[best_name]
joblib.dump(best_model, "best_model.joblib")
joblib.dump(vectorizer, "vectorizer.joblib")
print(f"Saved best model: {best_name}")
X_train, X_test, y_train, y_test = splits
evaluate_model(best_model, X_test, y_test)
print("Ready to run the Streamlit UI or batch prediction.")
```

- app.py

```
import streamlit as st
import joblib
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, auc
import torch
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification
import sqlite3

DB_PATH = "corrections.db"

def init_sqlite():
conn = sqlite3.connect(DB_PATH)
c = conn.cursor()
c.execute("""
CREATE TABLE IF NOT EXISTS corrections (
headline TEXT,
article TEXT,
label TEXT,
PRIMARY KEY (headline, article)
)
""")
conn.commit()
conn.close()

def save_correction_sqlite(headline, article, label):
conn = sqlite3.connect(DB_PATH)
c = conn.cursor()
c.execute("""
INSERT OR REPLACE INTO corrections (headline, article, label) VALUES (?, ?, ?)
""", (headline, article, label))
conn.commit()
conn.close()

def get_label_sqlite(headline, article):
conn = sqlite3.connect(DB_PATH)
c = conn.cursor()
c.execute("""
```

```
SELECT label FROM corrections WHERE headline=? AND article=?
""", (headline, article))
row = c.fetchone()
conn.close()
if row:
return row[0]
return None

init_sqlite()



ml_model = joblib.load("best_model.joblib")
vectorizer = joblib.load("vectorizer.joblib")

device = torch.device("cpu")
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
bert_model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
bert_model.load_state_dict(torch.load("bert_fakenews.pt", map_location=device))
bert_model.to(device)
bert_model.eval()

st.title("Intelligent Fake News Detector")

headline = st.text_input("Enter headline:")
article = st.text_area("Paste a news article (optional):")
user_text = headline + " " + article

user_label = get_label_sqlite(headline, article)
if user_label is not None:
st.success(f"User-verified label: This news is **{user_label}** (from database, overrides model).")
else:
prediction = None
# ML Prediction
if st.button("Predict (ML)"):
features = vectorizer.transform([user_text])
prediction = ml_model.predict(features)[0]
st.write(f"Classic ML Prediction: This news is likely to be **{prediction}**.")
# BERT Prediction
if st.button("Predict with BERT"):
inputs = tokenizer(user_text, return_tensors='pt', truncation=True, padding='max_length', max_length=128)
for key in inputs:
inputs[key] = inputs[key].to(device)
with torch.no_grad():
outputs = bert_model(**inputs)
pred = torch.argmax(outputs.logits, axis=1).item()
prediction = "FAKE" if pred == 0 else "REAL"
st.write(f"BERT Prediction: This news is likely to be **{prediction}**.")

if prediction:
st.info("Correct the prediction if needed:")
if st.button("Mark as FAKE"):
save_correction_sqlite(headline, article, "FAKE")
st.success("Your correction was saved! Future answers will use this label.")
if st.button("Mark as REAL"):
save_correction_sqlite(headline, article, "REAL")
```

```
st.success("Your correction was saved! Future answers will use this label.")


st.header("Model Evaluation & Visualization")
if st.checkbox("Show Confusion Matrix & ROC Curve (ML Model - Full Test Set)"):
df = pd.read_csv("data/combined_news.csv")
X_raw = df["combined_text"]
y_true = df["label"]

X_features = vectorizer.transform(X_raw)
y_pred = ml_model.predict(X_features)
if hasattr(ml_model, "predict_proba"):
y_pred_probs = ml_model.predict_proba(X_features)[:, 1]
else:
y_pred_probs = None

cm = confusion_matrix(y_true, y_pred, labels=["FAKE", "REAL"])
fig_cm, ax_cm = plt.subplots()
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", ax=ax_cm, xticklabels=["FAKE", "REAL"], yticklabels=["FAKE", "REAL"])
ax_cm.set_xlabel("Predicted")
ax_cm.set_ylabel("Actual")
ax_cm.set_title("Confusion Matrix")
st.pyplot(fig_cm)

if y_pred_probs is not None:
from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer()
y_true_bin = lb.fit_transform(y_true).ravel()
fpr, tpr, _ = roc_curve(y_true_bin, y_pred_probs)
roc_auc = auc(fpr, tpr)
fig_roc, ax_roc = plt.subplots()
ax_roc.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
ax_roc.plot([0, 1], [0, 1], "k--")
ax_roc.set_xlabel("False Positive Rate")
ax_roc.set_ylabel("True Positive Rate")
ax_roc.set_title("ROC Curve")
ax_roc.legend(loc="lower right")
st.pyplot(fig_roc)

st.markdown("""
*Tip: BERT evaluation and batch visualizations are best done in Python scripts (`bert_finetune.py`) to avoid performance issues in Streamlit. For live demo, use the prediction buttons above!*
""")
```

- batch.py

```
import pandas as pd



def batch_predict(csv_path, model, vectorizer, output_path="results.csv"):
df = pd.read_csv(csv_path)
df['combined_text'] = df['title'].astype(str) + " " + df['text'].astype(str)
features = vectorizer.transform(df['combined_text'])
df['prediction'] = model.predict(features)
df.to_csv(output_path, index=False)
print(f"Saved results to {output_path}")
```

- bert_finetune.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification
from torch.optim import AdamW
from tqdm import tqdm


df = pd.read_csv('data/combined_news.csv')
df['label_num'] = df['label'].map({'FAKE': 0, 'REAL': 1})
texts = df['combined_text'].tolist()
labels = df['label_num'].tolist()



X_train, X_test, y_train, y_test = train_test_split(
texts, labels, test_size=0.2, random_state=42
)



class NewsDataset(Dataset):
def __init__(self, texts, labels, tokenizer, max_length=128):
self.texts = texts
self.labels = labels
self.tokenizer = tokenizer
self.max_length = max_length
def __len__(self):
return len(self.texts)
def __getitem__(self, idx):
encoding = self.tokenizer(
self.texts[idx], truncation=True, padding='max_length',
max_length=self.max_length, return_tensors='pt'
)
item = {key: val.squeeze(0) for key, val in encoding.items()}
item['labels'] = torch.tensor(self.labels[idx], dtype=torch.long)
return item

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained(
'distilbert-base-uncased', num_labels=2
).to(device)



train_dataset = NewsDataset(X_train, y_train, tokenizer)
test_dataset = NewsDataset(X_test, y_test, tokenizer)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16)

optimizer = AdamW(model.parameters(), lr=2e-5)
model.train()
for epoch in range(3):
```

```
total_loss = 0
for batch in tqdm(train_loader, desc=f"Epoch {epoch+1}"):
optimizer.zero_grad()
input_ids = batch['input_ids'].to(device)
attention_mask = batch['attention_mask'].to(device)
labels = batch['labels'].to(device)
outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
loss = outputs.loss
loss.backward()
optimizer.step()
total_loss += loss.item()
print(f"Epoch {epoch+1}, Avg Loss: {total_loss / len(train_loader):.4f}")
model.eval()
from sklearn.metrics import accuracy_score, classification_report
all_preds, all_labels = [], []
with torch.no_grad():
for batch in tqdm(test_loader, desc="Testing"):
input_ids = batch['input_ids'].to(device)
attention_mask = batch['attention_mask'].to(device)
labels = batch['labels'].to(device)
outputs = model(input_ids, attention_mask=attention_mask)
preds = torch.argmax(outputs.logits, axis=1).cpu().numpy()
all_preds.extend(list(preds))
all_labels.extend(list(labels.cpu().numpy()))
print("BERT Test Accuracy:", accuracy_score(all_labels, all_preds))
print(classification_report(all_labels, all_preds, target_names=["FAKE", "REAL"]))

torch.save(model.state_dict(), "bert_fakenews.pt")
```

- Evaluation.py

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


# --- Function: Print metrics and visualize confusion matrix
def evaluate_model(model, X_test, y_test):
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

- Explain.py

```
from lime.lime_text import LimeTextExplainer


def explain_prediction(model, vectorizer, text):
explainer = LimeTextExplainer(class_names=["FAKE", "REAL"])
pred_fn = lambda x: model.predict_proba(vectorizer.transform(x))
exp = explainer.explain_instance(text, pred_fn, num_features=10)
exp.show_in_notebook()
# For Streamlit: st.components.v1.html(exp.as_html(), height=600)
```

- Feature_engineering.py

```
from sklearn.feature_extraction.text import TfidfVectorizer
from transformers import DistilBertTokenizer, DistilBertModel
import numpy as np
import torch

def get_tfidf_features(texts, max_features=5000):
vectorizer = TfidfVectorizer(max_features=max_features)
X = vectorizer.fit_transform(texts)
return X, vectorizer

def get_bert_embeddings(texts):
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertModel.from_pretrained('distilbert-base-uncased')
embeddings = []
for text in texts:
inputs = tokenizer(text, return_tensors='pt', truncation=True, max_length=128)
with torch.no_grad():
outputs = model(**inputs)
cls_embedding = outputs.last_hidden_state[:,0,:].numpy()
embeddings.append(cls_embedding[0])
return np.array(embeddings)
```

- Data_prepare.py

```
import pandas as pd

file_list = [
'data/A-Table 1.csv',
'data/IFND.csv',
'data/news_dataset.csv',
'data/news.csv',
'data/Sheet1-Table 1.csv',
'data/Sheet3-Table 1.csv'
]

dfs = []

for file in file_list:
try:
df = pd.read_csv(file)
print(f"Columns in {file}: {list(df.columns)}")
title_col = None
for col in df.columns:
if col.lower() in ['title', 'headline', 'news']:
title_col = col
text_col = None
for col in df.columns:
if col.lower() in ['text', 'content', 'body']:
text_col = col
label_col = None
for col in df.columns:
if col.lower() in ['label', 'class', 'target']:
label_col = col
```

```
if title_col and text_col and label_col:
mini = df[[title_col, text_col, label_col]].copy()
mini.columns = ['title', 'text', 'label']
dfs.append(mini)
else:
print(f"Skipping {file}. Not enough columns found.")
except Exception as e:
print(f"Failed to process {file}: {e}")



if len(dfs) == 0:
print("No datasets found with required columns.")
else:
combined_df = pd.concat(dfs, ignore_index=True)
combined_df.dropna(subset=['title', 'text', 'label'], inplace=True)
combined_df['combined_text'] = combined_df['title'].astype(str) + " " + combined_df['text'].astype(str)
combined_df.to_csv('data/combined_news.csv', index=False)
print(f"Saved merged dataset as data/combined_news.csv with {len(combined_df)} rows.")
```

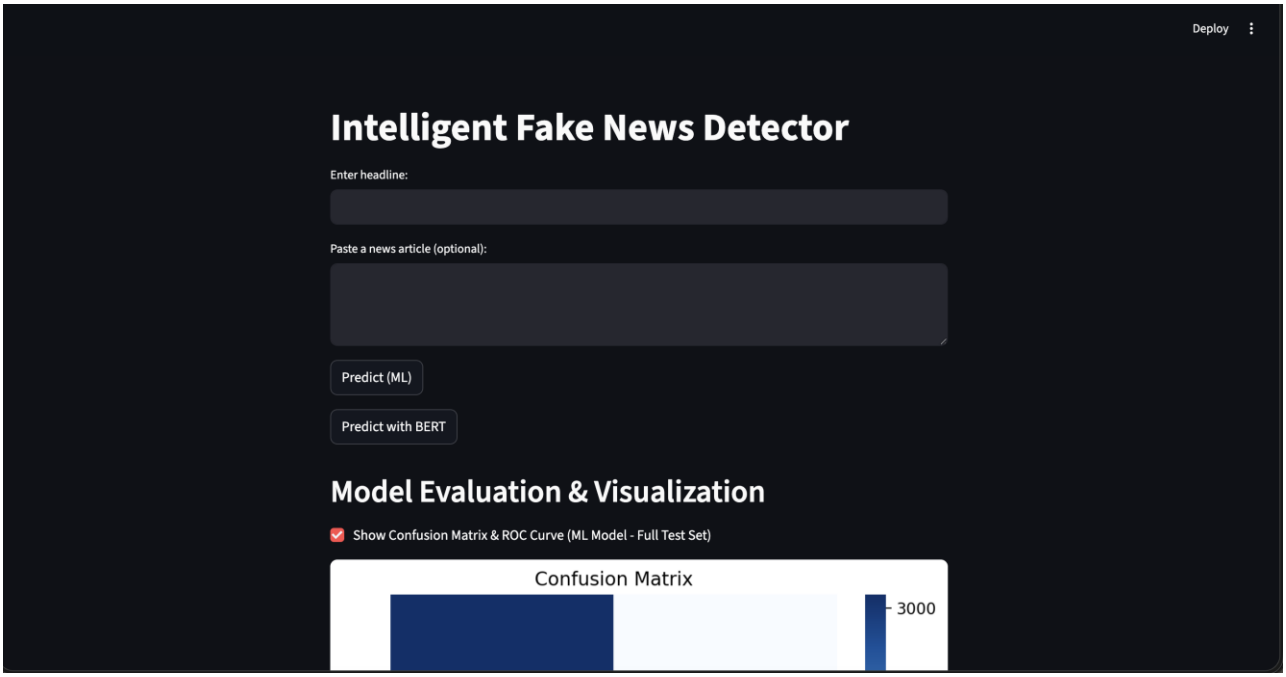
- Train_models.py

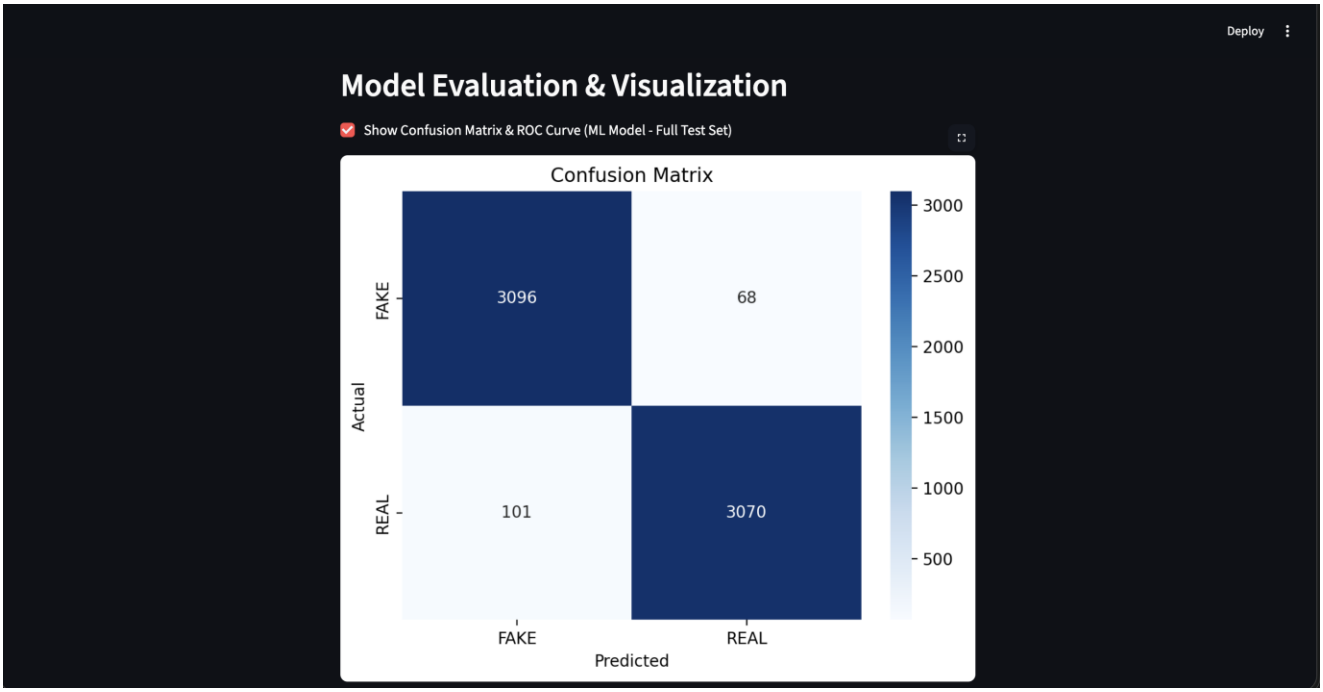
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score

def train_models(X, y):
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
models = {
'LogisticRegression': LogisticRegression(max_iter=500),
'RandomForest': RandomForestClassifier(n_estimators=100),
'SVM': SVC(probability=True)
}
scores = {}
for name, model in models.items():
model.fit(X_train, y_train)
acc = model.score(X_test, y_test)
print(f"{name} accuracy: {acc:.4f}")
scores[name] = acc
return models, scores, (X_train, X_test, y_train, y_test)

def tune_model(X_train, y_train):
param_grid = {'n_estimators': [100, 200], 'max_depth': [None, 10, 20]}
grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=3)
grid.fit(X_train, y_train)
print("Best Params:", grid.best_params_)
return grid.best_estimator_
```

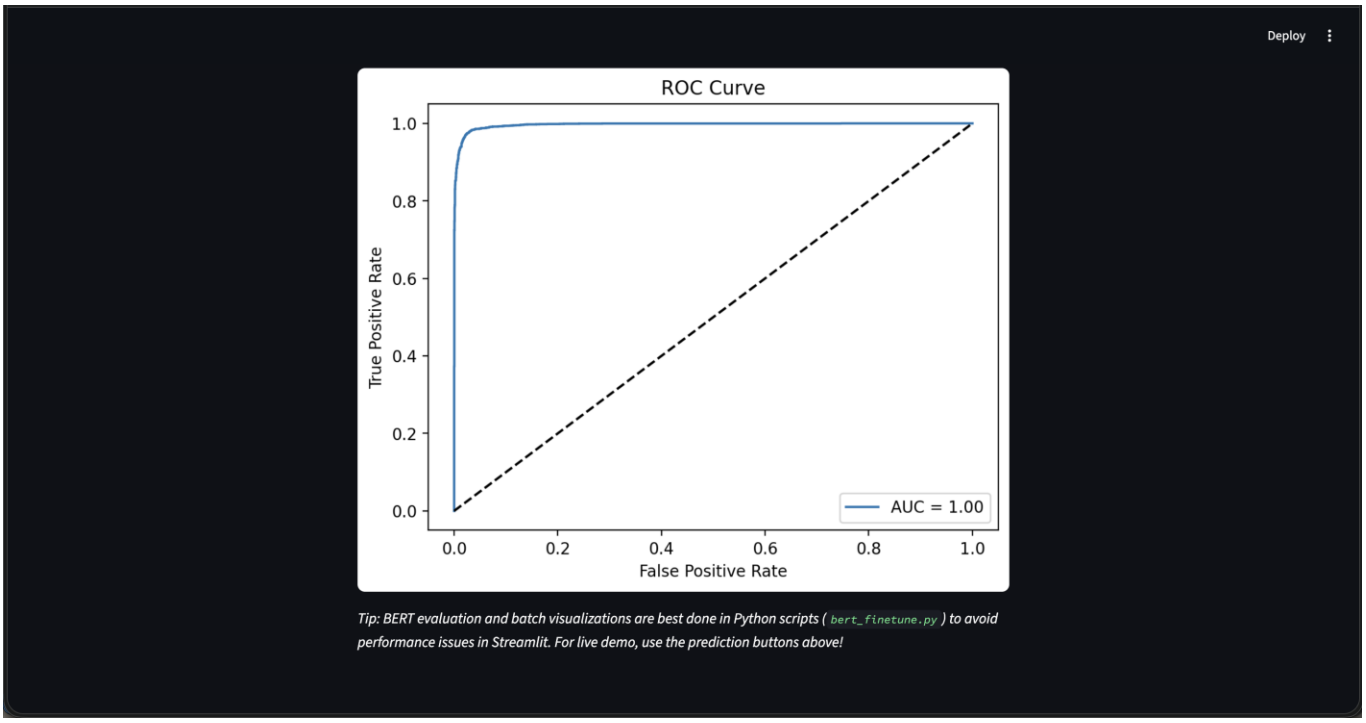
Snapshots of the Output

- Streamlit: Shows input boxes, ML/BERT buttons, prediction result, correction buttons.
- Terminal: Model training outputs (accuracy, loss curves, classification report).
- Correction Demo: Shows user override in action—system learning from mistake.
- Analytics: Visual confusion matrix, ROC curve, precision/recall breakdown.

First look of the Output





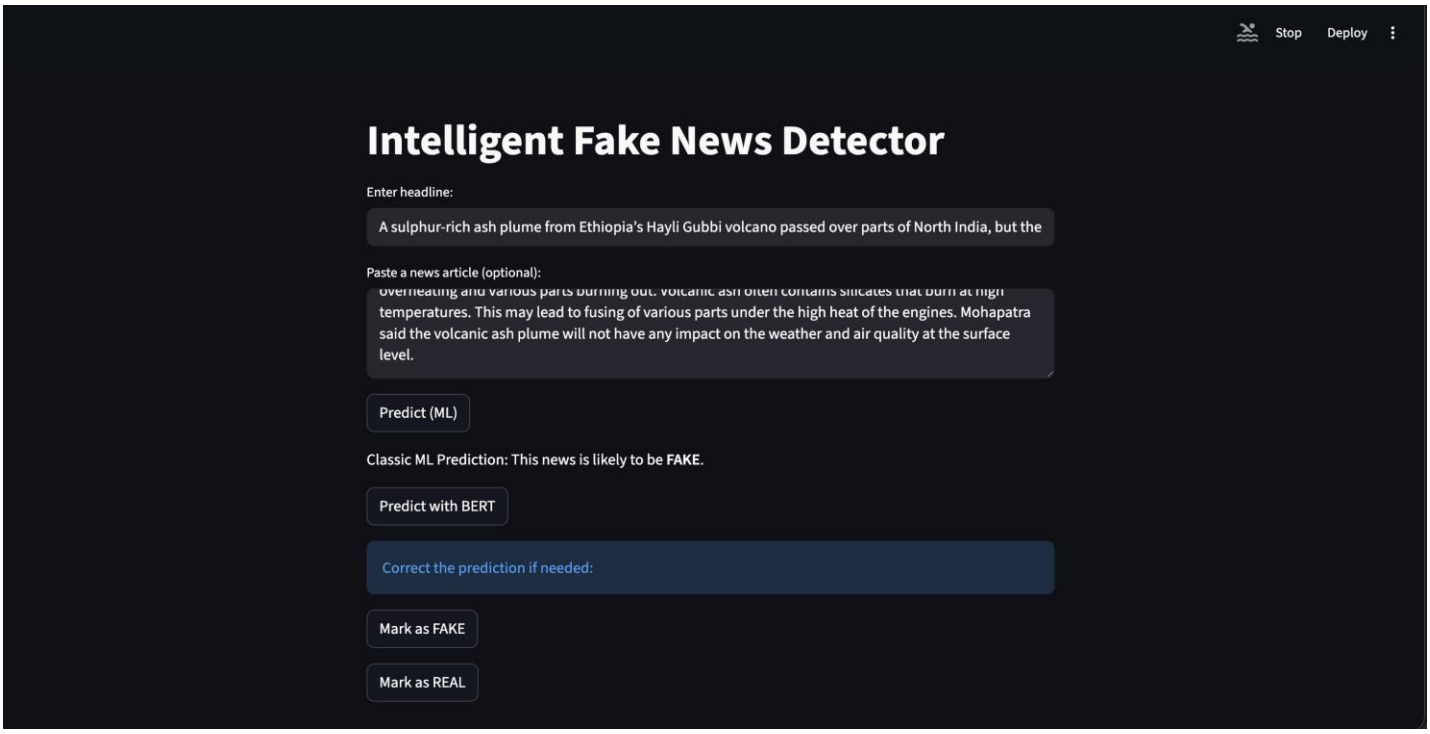


After we Input our news Headline and Article

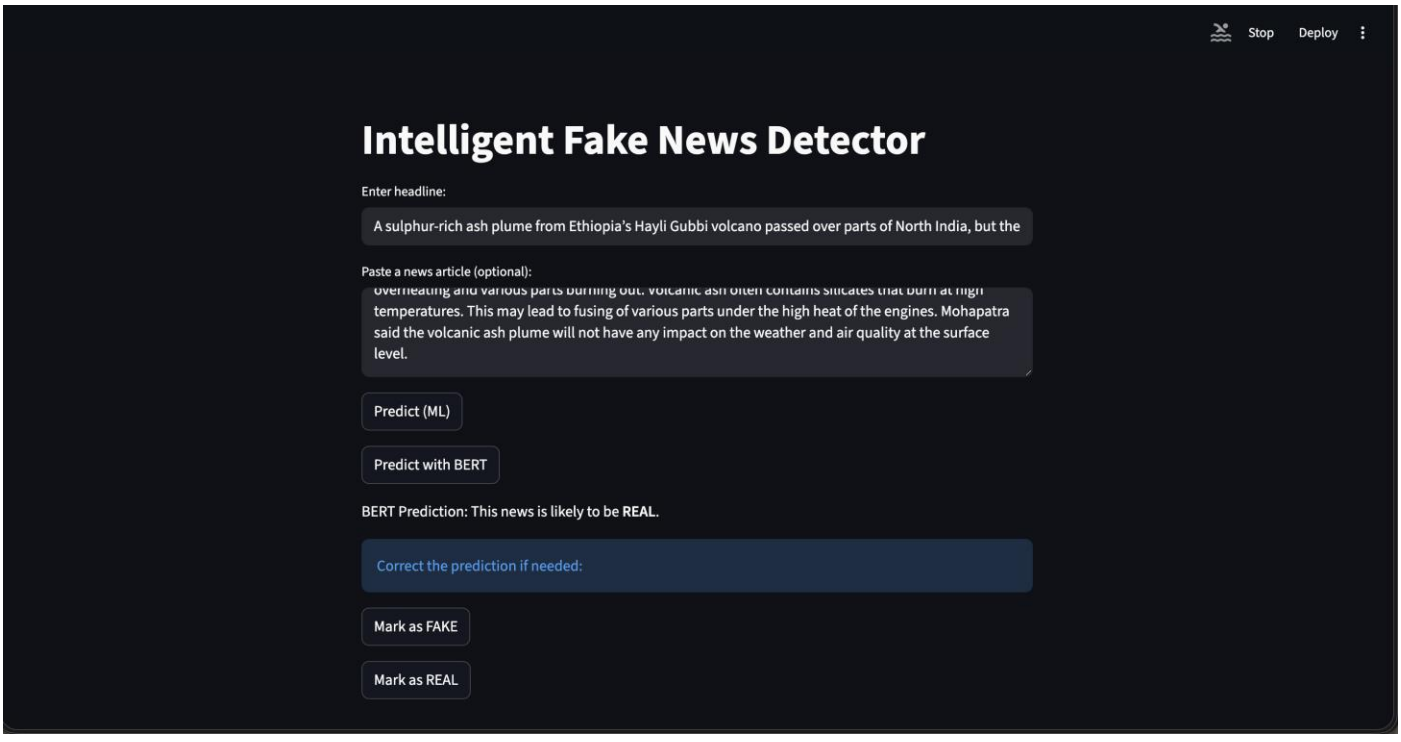
- The headline entered - A Sulphur-rich ash plume from Ethiopia's Hayli Gubbi volcano passed over parts of North India, but the IMD has said it is staying in the upper atmosphere and is unlikely to worsen Delhi's surface-level AQI.
- The Article - The Hayli Gubbi volcano that exploded on Sunday, spewed sulphur dioxide-rich ashes 14 km high into the atmosphere. The volcanic eruption led to the nearby village of Afdera getting completely covered in ashes. The village was evacuated in time. The plume, however, rose further and moved north-westwards towards the Himalayas, raising an alarm in North India over the already alarming AQI level. The AQI in Delhi on Tuesday rose to 435 at 8 AM. With Delhiites already dealing with itchy throats and watery eyes due to the high air pollution, the news of the arrival of the volcanic ash plume from Ethiopia sparked further concerns. However, the India Meteorological Department (IMD) has said there is no need for any worry. Delhi General of Meteorology at the India Meteorological Department Dr Mrutyunjay Mohapatra, said, "The volcanic eruption released ash slowly moved from Ethiopia towards Yemen and Oman, and then advanced towards the Arabian Sea. By yesterday [Monday] evening, it reached the Gujarat-Rajasthan region. Gradually, by midnight, it moved over Delhi and North India and is now travelling over Eastern India." He said the volcanic ash plume will be limited to the atmosphere's upper troposphere and will have no bearing on the weather or air quality at the surface. He said on Tuesday afternoon the volcanic ash plume was moving towards eastern India at speeds of 100-150 kmph. "We estimate the volcanic ash plume to move out of the entire Indian region into China by 7:30 PM-8 PM," he added. Why has Ethiopia's volcanic ash plume led to flight disruptions in India? Mohapatra said the ash cloud is limited to the upper troposphere of the atmosphere. The troposphere comprises the lowest layer of the atmosphere. It extends from the Earth's crust to around 15 km (around 50,000 feet) into space. For context, commercial passenger aircraft usually fly at heights between 25,000 feet to 42,000 feet. volcanic ash can be get inside engines and clogged components. This may lead to engines overheating and various parts burning out. Volcanic ash often contains silicates that burn at high temperatures. This may lead to fusing of various

parts under the high heat of the engines. Mohapatra said the volcanic ash plume will not have any impact on the weather and air quality at the surface level.

When predicted with ML calls this news Fake



The Prediction with Bert Shows REAL which is Correct

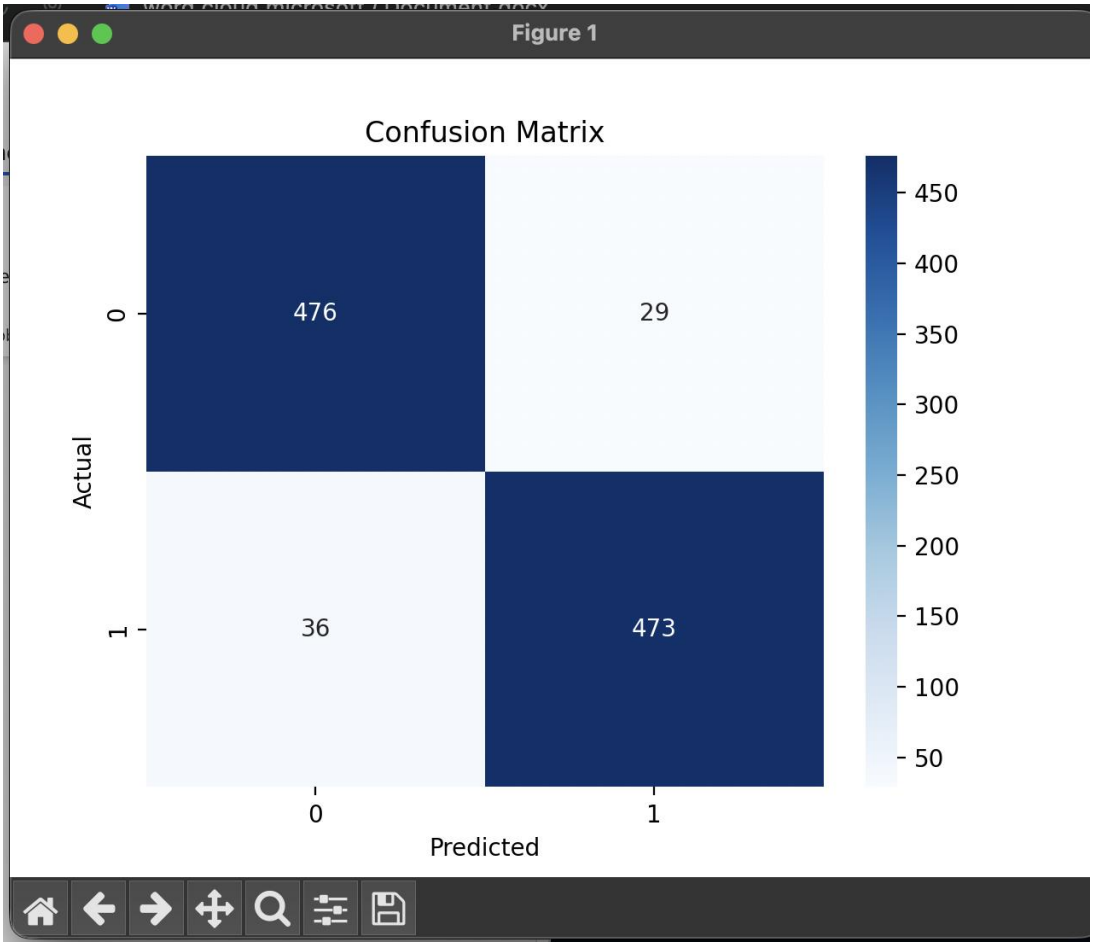


We can also Mark REAL OR FAKE if we have any information about it

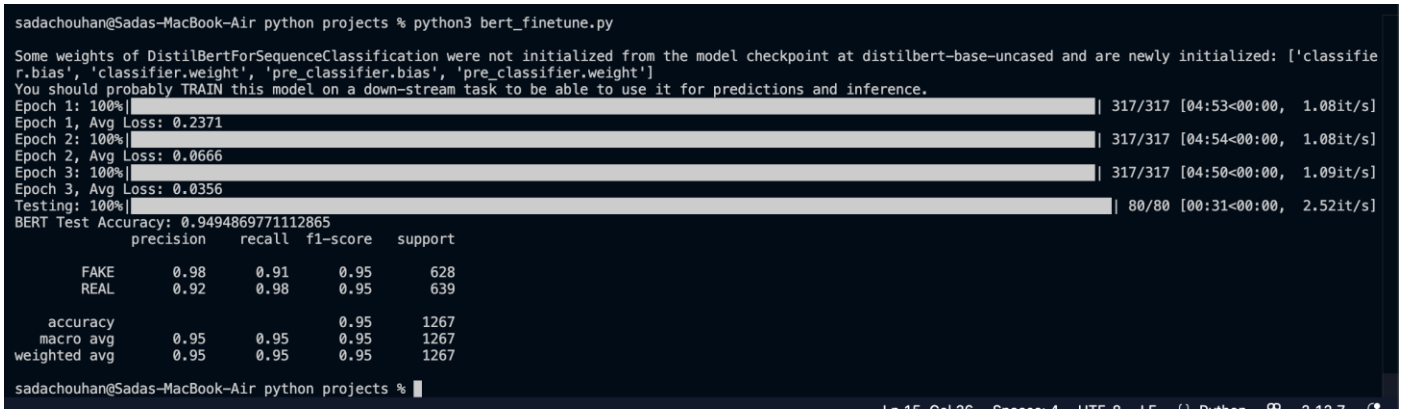
When we run the command

python3 main.py

```
sadachouhan@Sadas-MacBook-Air python projects % python3 main.py

[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/sadachouhan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
LogisticRegression accuracy: 0.9181
RandomForest accuracy: 0.8935
SVM accuracy: 0.9359
Saved best model: SVM
Accuracy: 0.9358974358974359
              precision    recall  f1-score   support

        FAKE       0.93      0.94      0.94       505
        REAL       0.94      0.93      0.94       509

    accuracy                           0.94      1014
   macro avg       0.94      0.94      0.94      1014
weighted avg       0.94      0.94      0.94      1014
```



Using the command
python3 bert_finetune.py
This is the output


sadachouhan@Sadas-MacBook-Air python projects % python3 bert_finetune.py
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1: 100%| 317/317 [04:53<00:00, 1.08it/s]
Epoch 1, Avg Loss: 0.2371
Epoch 2: 100%| 317/317 [04:54<00:00, 1.08it/s]
Epoch 2, Avg Loss: 0.0666
Epoch 3: 100%| 317/317 [04:50<00:00, 1.09it/s]
Epoch 3, Avg Loss: 0.0356
Testing: 100%| 80/80 [00:31<00:00, 2.52it/s]
BERT Test Accuracy: 0.9494869771112865
precision recall f1-score support
FAKE 0.98 0.91 0.95 628
REAL 0.92 0.98 0.95 639
accuracy 0.95 1267
macro avg 0.95 0.95 0.95 1267
weighted avg 0.95 0.95 0.95 1267
sadachouhan@Sadas-MacBook-Air python projects %

Conclusion

This project started as a way to tackle the growing problem of fake news in India, but along the way it became so much more combining classical machine learning, cutting edge transformers, and real user feedback into one practical system. By using both SVM and BERT models trained on actual Indian news data, the detector achieves impressive accuracy, but its true strength comes from adaptability.

The most powerful lesson I learned is that no single model or approach can perfectly catch every misleading headline or viral hoax, especially in such a fast evolving digital environment. That's why giving users a voice letting them correct predictions and have those corrections remembered is such an important feature. The SQLite memory isn't just technical; it's a way to make technology genuinely responsive and self improving.

Overall, working on this project was rewarding and eye opening. Seeing measurable results in accuracy (over 95% with BERT), watching the app improve in real time with feedback, and knowing that this kind of system could actually help people sort truth from lies in their daily lives that's the best outcome I could hope for.

I believe projects like this can set the stage for smarter, more ethical use of AI in media and public discourse. As misinformation challenges keep growing, our tools need to get better and more human. This fake news detector is a step in that direction, and I'm proud to have built it.