"""

mental\_health\_signal\_detection.py

Starter implementation for:

"Mental Health Signal Detection from Social Media Posts Using Machine Learning and NLP"

Usage:

1. Install dependencies (see below).

2. Prepare CSV with columns: 'text', 'label' (0 or 1).

3. Update DATA\_PATH and run.

Dependencies (example):

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

Notes:

- This script uses TF-IDF for classical models and BERT mean-pooled embeddings for a stronger classifier.

- For full BERT fine-tuning, you can extend this script using Hugging Face Trainer or PyTorch training loops.

"""

import re

import os

import argparse

import joblib

from typing import List, Tuple, Any

import numpy as np

import pandas as pd

from tqdm import tqdm

# NLP libs

import nltk

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

# sklearn

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (

accuracy\_score,

precision\_score,

recall\_score,

f1\_score,

roc\_auc\_score,

confusion\_matrix,

classification\_report,

)

# Transformers for embeddings

from transformers import AutoTokenizer, AutoModel

import torch

# Visualization

import matplotlib.pyplot as plt

# Ensure necessary NLTK assets are downloaded

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('omw-1.4')

# ----------------------------

# Config

# ----------------------------

DATA\_PATH = "data/social\_posts.csv" # <-- change to your dataset path

RANDOM\_STATE = 42

TEST\_SIZE = 0.15

VALID\_SIZE = 0.15 # portion for validation from train (will be handled below)

BERT\_MODEL\_NAME = "bert-base-uncased" # can change to other models

DEVICE = "cuda" if torch.cuda.is\_available() else "cpu"

MODEL\_OUTPUT\_DIR = "saved\_models"

os.makedirs(MODEL\_OUTPUT\_DIR, exist\_ok=True)

# ----------------------------

# Preprocessing

# ----------------------------

lemmatizer = WordNetLemmatizer()

STOPWORDS = set(stopwords.words("english"))

def clean\_text(text: str) -> str:

"""Basic text cleaning: removes URLs, mentions, hashtags, punctuation, digits, extra spaces."""

text = str(text)

text = text.lower()

# remove urls

text = re.sub(r"http\S+|www\.\S+", "", text)

# remove mentions and hashtags (optionally keep hashtag text)

text = re.sub(r"@\w+", "", text)

text = re.sub(r"#", "", text)

# remove non-alphanumeric chars

text = re.sub(r"[^a-z\s]", " ", text)

# collapse whitespace

text = re.sub(r"\s+", " ", text).strip()

return text

def tokenize\_and\_lemmatize(text: str) -> str:

"""Tokenize, remove stopwords, and lemmatize. Returns processed string."""

tokens = nltk.word\_tokenize(text)

tokens = [t for t in tokens if t not in STOPWORDS and len(t) > 1]

lemmas = [lemmatizer.lemmatize(t) for t in tokens]

return " ".join(lemmas)

def preprocess\_series(series: pd.Series) -> pd.Series:

"""Apply cleaning and preprocessing to a pandas Series of text."""

cleaned = series.fillna("").map(clean\_text)

processed = cleaned.map(tokenize\_and\_lemmatize)

return processed

# ----------------------------

# BERT Embeddings (mean pooling)

# ----------------------------

class BertEmbedder:

def \_\_init\_\_(self, model\_name: str = BERT\_MODEL\_NAME, device: str = DEVICE):

self.device = device

self.tokenizer = AutoTokenizer.from\_pretrained(model\_name)

self.model = AutoModel.from\_pretrained(model\_name)

self.model.to(self.device)

self.model.eval()

def embed\_texts(self, texts: List[str], batch\_size: int = 16) -> np.ndarray:

"""Return mean-pooled BERT embeddings for a list of texts."""

embeddings = []

with torch.no\_grad():

for i in range(0, len(texts), batch\_size):

batch = texts[i : i + batch\_size]

enc = self.tokenizer(batch, padding=True, truncation=True, return\_tensors="pt", max\_length=256)

input\_ids = enc["input\_ids"].to(self.device)

attention\_mask = enc["attention\_mask"].to(self.device)

outputs = self.model(input\_ids=input\_ids, attention\_mask=attention\_mask)

last\_hidden = outputs.last\_hidden\_state # (batch, seq\_len, hidden)

# mean pooling w.r.t attention mask

mask = attention\_mask.unsqueeze(-1).expand(last\_hidden.size()).float()

summed = torch.sum(last\_hidden \* mask, 1)

counts = torch.clamp(mask.sum(1), min=1e-9)

mean\_pooled = (summed / counts).cpu().numpy()

embeddings.append(mean\_pooled)

return np.vstack(embeddings)

# ----------------------------

# Training & Evaluation Utilities

# ----------------------------

def evaluate\_model(model, X\_test, y\_test, model\_name: str = "Model"):

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

# Some sklearn classifiers may have predict\_proba

try:

y\_proba = model.predict\_proba(X\_test)[:, 1]

except Exception:

# fallback: decision\_function -> minmax scale to 0-1

try:

scores = model.decision\_function(X\_test)

y\_proba = (scores - scores.min()) / (scores.max() - scores.min() + 1e-9)

except Exception:

y\_proba = None

acc = accuracy\_score(y\_test, y\_pred)

prec = precision\_score(y\_test, y\_pred, zero\_division=0)

rec = recall\_score(y\_test, y\_pred, zero\_division=0)

f1 = f1\_score(y\_test, y\_pred, zero\_division=0)

auc = roc\_auc\_score(y\_test, y\_proba) if y\_proba is not None else None

print(f"\n=== {model\_name} Evaluation ===")

print(f"Accuracy: {acc:.4f}")

print(f"Precision: {prec:.4f}")

print(f"Recall: {rec:.4f}")

print(f"F1-score: {f1:.4f}")

if auc is not None:

print(f"ROC-AUC: {auc:.4f}")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, zero\_division=0))

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

return {

"accuracy": acc,

"precision": prec,

"recall": rec,

"f1": f1,

"auc": auc,

}

def plot\_confusion\_matrix(cm, labels, title="Confusion matrix", savepath=None):

"""Simple confusion matrix plot"""

fig, ax = plt.subplots(figsize=(5, 4))

ax.matshow(cm, cmap=plt.cm.Blues)

for (i, j), val in np.ndenumerate(cm):

ax.text(j, i, f"{val}", ha="center", va="center", color="black")

ax.set\_xticks(range(len(labels)))

ax.set\_yticks(range(len(labels)))

ax.set\_xticklabels(labels)

ax.set\_yticklabels(labels)

ax.set\_xlabel("Predicted")

ax.set\_ylabel("Actual")

ax.set\_title(title)

plt.tight\_layout()

if savepath:

plt.savefig(savepath, dpi=200)

plt.show()

# ----------------------------

# Main pipeline

# ----------------------------

def run\_pipeline(data\_path: str = DATA\_PATH):

# 1) Load data

print("Loading dataset from:", data\_path)

df = pd.read\_csv(data\_path)

if "text" not in df.columns or "label" not in df.columns:

raise ValueError("CSV must contain 'text' and 'label' columns.")

df = df[["text", "label"]].dropna().reset\_index(drop=True)

# 2) Preprocess

print("Preprocessing text...")

df["text\_proc"] = preprocess\_series(df["text"])

# 3) Train-test split

X = df["text\_proc"].values

y = df["label"].astype(int).values

X\_train\_val, X\_test, y\_train\_val, y\_test = train\_test\_split(X, y, test\_size=TEST\_SIZE, random\_state=RANDOM\_STATE, stratify=y)

# further split train into train/val

val\_portion = VALID\_SIZE / (1.0 - TEST\_SIZE)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train\_val, y\_train\_val, test\_size=val\_portion, random\_state=RANDOM\_STATE, stratify=y\_train\_val)

print(f"Dataset sizes: train={len(X\_train)}, val={len(X\_val)}, test={len(X\_test)}")

# -----------------------------

# Classical models with TF-IDF

# -----------------------------

print("\nExtracting TF-IDF features...")

tfidf = TfidfVectorizer(max\_features=20000, ngram\_range=(1, 2))

X\_train\_tfidf = tfidf.fit\_transform(X\_train)

X\_val\_tfidf = tfidf.transform(X\_val)

X\_test\_tfidf = tfidf.transform(X\_test)

# Save TF-IDF vectorizer

joblib.dump(tfidf, os.path.join(MODEL\_OUTPUT\_DIR, "tfidf\_vectorizer.joblib"))

# Logistic Regression (baseline)

print("\nTraining Logistic Regression (TF-IDF)...")

lr = LogisticRegression(max\_iter=1000, random\_state=RANDOM\_STATE)

lr.fit(X\_train\_tfidf, y\_train)

joblib.dump(lr, os.path.join(MODEL\_OUTPUT\_DIR, "logistic\_tfidf.joblib"))

evaluate\_model(lr, X\_test\_tfidf, y\_test, model\_name="LogisticRegression (TF-IDF)")

# SVM

print("\nTraining SVM (TF-IDF)...")

svm = SVC(kernel="rbf", probability=True, random\_state=RANDOM\_STATE)

svm.fit(X\_train\_tfidf, y\_train)

joblib.dump(svm, os.path.join(MODEL\_OUTPUT\_DIR, "svm\_tfidf.joblib"))

evaluate\_model(svm, X\_test\_tfidf, y\_test, model\_name="SVM (TF-IDF)")

# Random Forest

print("\nTraining Random Forest (TF-IDF)...")

rf = RandomForestClassifier(n\_estimators=200, random\_state=RANDOM\_STATE, n\_jobs=-1)

rf.fit(X\_train\_tfidf, y\_train)

joblib.dump(rf, os.path.join(MODEL\_OUTPUT\_DIR, "rf\_tfidf.joblib"))

evaluate\_model(rf, X\_test\_tfidf, y\_test, model\_name="RandomForest (TF-IDF)")

# -----------------------------

# BERT embeddings + simple classifier

# -----------------------------

print("\nGenerating BERT embeddings (this may take a while)...")

embedder = BertEmbedder(model\_name=BERT\_MODEL\_NAME, device=DEVICE)

X\_train\_emb = embedder.embed\_texts(list(X\_train), batch\_size=16)

X\_val\_emb = embedder.embed\_texts(list(X\_val), batch\_size=16)

X\_test\_emb = embedder.embed\_texts(list(X\_test), batch\_size=16)

# Train a Logistic Regression on top of embeddings

print("\nTraining Logistic Regression on BERT embeddings...")

lr\_bert = LogisticRegression(max\_iter=1000, random\_state=RANDOM\_STATE)

lr\_bert.fit(X\_train\_emb, y\_train)

joblib.dump(lr\_bert, os.path.join(MODEL\_OUTPUT\_DIR, "logistic\_bert\_embeddings.joblib"))

bert\_metrics = evaluate\_model(lr\_bert, X\_test\_emb, y\_test, model\_name="LogisticRegression (BERT embeddings)")

# Optionally: save embeddings to disk for reuse (commented out)

# np.save(os.path.join(MODEL\_OUTPUT\_DIR, "X\_train\_emb.npy"), X\_train\_emb)

# np.save(os.path.join(MODEL\_OUTPUT\_DIR, "X\_test\_emb.npy"), X\_test\_emb)

# Plot confusion matrix for the best model (example using BERT logistic)

y\_pred = lr\_bert.predict(X\_test\_emb)

cm = confusion\_matrix(y\_test, y\_pred)

plot\_confusion\_matrix(cm, labels=["control(0)", "distress(1)"], title="Confusion Matrix - BERT Logistic", savepath=os.path.join(MODEL\_OUTPUT\_DIR, "cm\_bert\_logistic.png"))

print("\nAll done. Models saved under:", MODEL\_OUTPUT\_DIR)

return {

"tfidf\_vectorizer": tfidf,

"lr\_tfidf": lr,

"svm\_tfidf": svm,

"rf\_tfidf": rf,

"lr\_bert": lr\_bert,

"bert\_embedder": embedder,

}

if \_\_name\_\_ == "\_\_main\_\_":

# For convenience allow command-line override

parser = argparse.ArgumentParser(description="Mental health signal detection pipeline")

parser.add\_argument("--data", type=str, default=DATA\_PATH, help="Path to CSV dataset (columns: text,label)")

args = parser.parse\_args()

run\_pipeline(data\_path=args.data)