# full\_sentiment\_ssa\_pipeline.py

# Run: python full\_sentiment\_ssa\_pipeline.py

import os

import re

import string

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from docx import Document

from collections import Counter

from datetime import datetime

from nltk.corpus import stopwords

from nltk import download

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import classification\_report, accuracy\_score, roc\_auc\_score

from sklearn.preprocessing import LabelEncoder

from imblearn.over\_sampling import SMOTE

from sentence\_transformers import SentenceTransformer

import umap.umap\_ as umap

from fpdf import FPDF

from textblob import TextBlob

import webbrowser

download('stopwords')

TRAIN\_PATH = "/Users/gulcicek/Downloads/interviews\_train.docx"

TEST\_PATH = "/Users/gulcicek/Downloads/interviews\_test.docx"

OUTPUT\_DIR = "ssa\_output"

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

SSA\_PHRASES = ["always see the same", "endless loop", "feel trapped", "no control", "not truly connecting", "emotionally drained", "manipulative", "stuck", "I can't get out", "I follow accounts", "only what I like"]

EMOTION\_WORDS = ["addictive", "anxious", "love", "frustrating", "satisfying", "isolated", "toxic", "dangerous", "disconnected", "bored", "empty"]

def extract\_interviews(path):

doc = Document(path)

full = "\n".join([p.text.strip() for p in doc.paragraphs if p.text.strip()])

pairs = re.findall(r"(Interviewee:.\*?)(Positive|Negative|Notr)", full, re.DOTALL)

return pd.DataFrame(pairs, columns=["text", "sentiment"])

def extract\_unlabeled(path):

doc = Document(path)

full = "\n".join([p.text.strip() for p in doc.paragraphs if p.text.strip()])

blocks = re.split(r"(?=Interviewee:)", full)

return pd.DataFrame({"text": [b.strip() for b in blocks if b.strip()]})

def clean\_text(text):

text = text.lower()

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

text = text.translate(str.maketrans("", "", string.punctuation))

stop\_words = set(stopwords.words("turkish"))

tokens = text.split()

return " ".join([t for t in tokens if t not in stop\_words and len(t) > 2])

def analyze\_sentiment(text):

return round(TextBlob(text).sentiment.polarity, 3)

def count\_phrase\_matches(text, phrase\_list):

return [phrase for phrase in phrase\_list if phrase.lower() in text.lower()]

def extract\_sentences(docx\_path):

doc = Document(docx\_path)

sents = []

for para in doc.paragraphs:

if not para.text.strip(): continue

for sent in re.split(r"(?<=[.!?])\s+", para.text.strip()):

if len(sent.split()) > 4:

sents.append(sent)

return sents

def run\_ssa\_report(docx\_path):

lines = extract\_sentences(docx\_path)

results = []

for idx, line in enumerate(lines):

results.append({

"id": idx + 1,

"text": line,

"sentiment": analyze\_sentiment(line),

"ssa\_keywords": ", ".join(count\_phrase\_matches(line, SSA\_PHRASES)),

"emotional\_indicators": ", ".join(count\_phrase\_matches(line, EMOTION\_WORDS))

})

df = pd.DataFrame(results)

df.to\_csv(os.path.join(OUTPUT\_DIR, "ssa\_nlp\_docx\_output.csv"), index=False)

return df

def plot\_and\_save(df):

plt.figure()

plt.hist(df["sentiment"], bins=10, edgecolor='black')

plt.title("Sentiment Distribution")

plt.tight\_layout()

plt.savefig(os.path.join(OUTPUT\_DIR, "sentiment\_dist.png"))

plt.close()

for col, title, file in [

("ssa\_keywords", "SSA Phrase Frequency", "ssa\_phrases.png"),

("emotional\_indicators", "Emotion Word Frequency", "emotion\_words.png")

]:

all\_words = Counter(kw.strip() for val in df[col] if val for kw in val.split(','))

if not all\_words: continue

keys, vals = zip(\*all\_words.items())

plt.figure(figsize=(10, 5))

plt.barh(keys, vals)

plt.title(title)

plt.tight\_layout()

plt.savefig(os.path.join(OUTPUT\_DIR, file))

plt.close()

class SSAReport(FPDF):

def header(self):

self.set\_font("Arial", "B", 12)

self.cell(0, 10, "Synthetic Social Alienation (SSA) NLP Report", ln=True, align="C")

self.ln(5)

def footer(self):

self.set\_y(-15)

self.set\_font("Arial", "I", 8)

self.cell(0, 10, f"Page {self.page\_no()}", 0, 0, "C")

def section\_title(self, title):

self.set\_font("Arial", "B", 11)

self.set\_fill\_color(220, 220, 220)

self.cell(0, 8, title, ln=True, fill=True)

self.ln(3)

def paragraph(self, text):

self.set\_font("Arial", "", 10)

self.multi\_cell(0, 6, text.encode("latin-1", "replace").decode("latin-1"))

self.ln()

def image\_plot(self, path):

if os.path.exists(path): self.image(path, w=180)

self.ln(5)

def summary\_table(self, metrics):

self.set\_font("Arial", "", 10)

for k, v in metrics.items():

self.cell(60, 6, k, border=1)

self.cell(0, 6, str(v), border=1, ln=True)

self.ln()

def generate\_pdf(df):

pdf = SSAReport()

pdf.add\_page()

pdf.section\_title("1. Overview")

pdf.paragraph(f"Generated: {datetime.now().strftime('%Y-%m-%d %H:%M')}")

pdf.section\_title("2. Summary")

metrics = {

"Total Sentences": len(df),

"SSA Matches": df['ssa\_keywords'].astype(bool).sum(),

"Emotional Matches": df['emotional\_indicators'].astype(bool).sum(),

"Avg Sentiment": round(df["sentiment"].mean(), 3)

}

pdf.summary\_table(metrics)

pdf.section\_title("3. Sentiment Distribution")

pdf.image\_plot(os.path.join(OUTPUT\_DIR, "sentiment\_dist.png"))

pdf.section\_title("4. SSA Phrase Frequency")

pdf.image\_plot(os.path.join(OUTPUT\_DIR, "ssa\_phrases.png"))

pdf.section\_title("5. Emotional Word Frequency")

pdf.image\_plot(os.path.join(OUTPUT\_DIR, "emotion\_words.png"))

pdf.output(os.path.join(OUTPUT\_DIR, "ssa\_analysis\_report.pdf"))

# webbrowser.open() disabled to avoid segmentation fault

# === PIPELINE EXECUTION ===

train\_df = extract\_interviews(TRAIN\_PATH)

test\_df = extract\_unlabeled(TEST\_PATH)

train\_df = train\_df[train\_df["sentiment"].isin(["Positive", "Negative"])]

train\_df["clean"] = train\_df["text"].apply(clean\_text)

test\_df["clean"] = test\_df["text"].apply(clean\_text)

train\_df["label"] = train\_df["sentiment"].map({"Negative": 0, "Positive": 1})

# TF-IDF + RF

tfidf = TfidfVectorizer(max\_features=1000)

X = tfidf.fit\_transform(train\_df["clean"])

y = train\_df["label"]

X, y = SMOTE(k\_neighbors=1, random\_state=42).fit\_resample(X, y)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.3, stratify=y)

clf = GridSearchCV(RandomForestClassifier(), {

"n\_estimators": [50, 100], "max\_depth": [None, 10]

}, scoring="accuracy", cv=3)

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

y\_pred = clf.best\_estimator\_.predict(X\_val)

print("\nTF-IDF Results")

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

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

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay, roc\_auc\_score

print("\n[Cross-Validation Accuracy]")

cv\_scores = cross\_val\_score(clf.best\_estimator\_, X, y, cv=5, scoring='accuracy')

print("CV Mean Accuracy: {:.2f}".format(cv\_scores.mean()))

print("CV Scores:", cv\_scores)

print("\n[Confusion Matrix]")

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

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=["Negative", "Positive"])

disp.plot(cmap='Blues')

plt.title("Validation Confusion Matrix")

plt.tight\_layout()

plt.savefig("confusion\_matrix.png")

plt.close()

print("\n[ROC AUC Score (Validation)]")

try:

y\_val\_proba = clf.best\_estimator\_.predict\_proba(X\_val)[:,1]

auc\_score = roc\_auc\_score(y\_val, y\_val\_proba)

print("ROC AUC:", round(auc\_score, 3))

except Exception as e:

print("ROC AUC not available:", e)

test\_df["predicted\_tfidf"] = clf.best\_estimator\_.predict(tfidf.transform(test\_df["clean"]))

test\_df["predicted\_tfidf"] = test\_df["predicted\_tfidf"].map({0: "Negative", 1: "Positive"})

test\_df[["text", "predicted\_tfidf"]].to\_csv("predictions\_tfidf.csv", index=False)

# SSA

model = SentenceTransformer("paraphrase-MiniLM-L6-v2")

X\_embed = model.encode(train\_df["clean"].tolist(), batch\_size=8, show\_progress\_bar=True)

X\_test\_embed = model.encode(test\_df["clean"].tolist(), batch\_size=8, show\_progress\_bar=True)

lr = LogisticRegression(max\_iter=1000).fit(X\_embed, train\_df["label"])

test\_df["predicted\_ssa"] = lr.predict(X\_test\_embed)

test\_df["predicted\_ssa"] = test\_df["predicted\_ssa"].map({0: "Negative", 1: "Positive"})

test\_df[["text", "predicted\_ssa"]].to\_csv("predictions\_ssa.csv", index=False)

# UMAP visualization disabled due to segmentation fault

# SSA PDF

ssa\_df = run\_ssa\_report(TEST\_PATH)

plot\_and\_save(ssa\_df)

generate\_pdf(ssa\_df)