# app\_feedback\_pipeline.py

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

# End-to-end pipeline for app-store review analytics

# - Ingest + clean (Google Play + Apple)

# - ML sentiment (DistilBERT SST-2) with hybrid fallback

# - Zero-shot topic tagging (BART-MNLI), batched

# - Optional summaries per topic (BART-CNN)

# - N-gram mining & feature-request flags

# - App-level summary report

#

# Configuration is at the top; all outputs go to ./outputs/.

# Set HUGGINGFACE\_TOKEN in your environment for Inference API use (optional).

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

import os

import re

import math

import json

from pathlib import Path

from typing import List, Tuple

import numpy as np

import pandas as pd

from tqdm.auto import tqdm

# Transformers (local pipelines)

from transformers import pipeline

import torch

# Sklearn for n-grams

from sklearn.feature\_extraction.text import CountVectorizer

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

# CONFIG

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

DATA\_DIR = Path(".")

OUT\_DIR = Path("./outputs")

OUT\_DIR.mkdir(parents=True, exist\_ok=True)

# Input file names (adjust if yours differ)

PLAY\_FILE = DATA\_DIR / "Play Store Data.csv"

APPLE\_FILE = DATA\_DIR / "Apple\_Store\_Reviews.csv"

# Canonical outputs

COMBINED\_CSV = OUT\_DIR / "combined\_app\_reviews.csv"

CLEAN\_CSV = OUT\_DIR / "super\_cleaned\_app\_reviews.csv"

ML\_SENT\_CSV = OUT\_DIR / "ml\_sentiment\_reviews.csv"

HYBRID\_SENT\_CSV = OUT\_DIR / "hybrid\_sentiment\_reviews.csv"

TOPIC\_CSV = OUT\_DIR / "topic\_classified\_reviews.csv"

APP\_SUMMARY\_CSV = OUT\_DIR / "app\_level\_summary\_report.csv"

NEG\_SUMMARY\_TXT = OUT\_DIR / "negative\_review\_summaries.txt"

NGRAMS\_CSV = OUT\_DIR / "top\_ngrams\_negative\_neutral.csv"

FEATURE\_REQUESTS\_CSV = OUT\_DIR / "feature\_requests.csv"

# Models

MODEL\_SENTIMENT = "distilbert-base-uncased-finetuned-sst-2-english"

MODEL\_ZS = "facebook/bart-large-mnli"

MODEL\_SUM = "facebook/bart-large-cnn"

# Zero-shot labels (use multi\_label to allow multiple topics)

ZS\_LABELS = ["user interface", "performance", "bugs", "features", "pricing", "ads", "login issues", "customer support"]

ZS\_MULTI\_LABEL = True

ZS\_SCORE\_THRESH = 0.35 # keep labels scoring >= threshold when multi\_label=True

# Placeholder review detection

PLACEHOLDERS = [

"no review available", "no reviews available", "no comment",

"n/a", "none", "", "null"

]

# Hybrid sentiment thresholds (when no usable text)

RATING\_POS = 4.0

RATING\_NEG = 2.0

# Chunking for summarization

MAX\_REVIEWS\_PER\_TOPIC\_FOR\_SUMMARY = 50 # safety on input length

SUMMARY\_MAX\_LEN = 120

SUMMARY\_MIN\_LEN = 40

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

# UTILS

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

def safe\_lower(s: str) -> str:

try:

return str(s).strip().lower()

except Exception:

return str(s)

def is\_placeholder\_text(s: str) -> bool:

s = safe\_lower(s)

return any(p == s or p in s for p in PLACEHOLDERS)

def batch\_iter(lst: List[str], batch\_size: int):

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

yield lst[i:i + batch\_size], i

def ensure\_columns(df: pd.DataFrame, cols: List[str]):

for c in cols:

if c not in df.columns:

df[c] = np.nan

return df

def map\_sst2\_label(label: str) -> str:

# Normalize to UPPER fixed set

label = safe\_lower(label)

if "pos" in label:

return "POSITIVE"

if "neg" in label:

return "NEGATIVE"

return "NEUTRAL"

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

# 1) INGEST + STANDARDIZE

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

def ingest\_and\_standardize() -> pd.DataFrame:

# Load

g = pd.read\_csv(PLAY\_FILE)

a = pd.read\_csv(APPLE\_FILE)

# Standardize names

g = g.rename(columns={

"App": "App\_Name",

"Rating": "App\_Rating",

"Category": "Category",

# if your Play CSV has review text, map here, else we'll inject placeholder

})

a = a.rename(columns={

"App\_Name": "App\_Name",

"Rating": "App\_Rating",

"Review\_Text": "Review",

"Category": "Category"

})

# Add review text for Google if missing

if "Review" not in g.columns:

g["Review"] = "No Review Available"

# Source

g["Source"] = "Google Play Store"

a["Source"] = "Apple App Store"

# Keep canonical columns if present

common = ["App\_Name", "App\_Rating", "Review", "Category", "Source"]

g = ensure\_columns(g, common)[common]

a = ensure\_columns(a, common)[common]

df = pd.concat([g, a], ignore\_index=True)

df.to\_csv(COMBINED\_CSV, index=False)

return df

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

# 2) CLEANING

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

def clean(df: pd.DataFrame) -> pd.DataFrame:

df = df.drop\_duplicates().copy()

# Coerce rating

if "App\_Rating" in df.columns:

df["App\_Rating"] = pd.to\_numeric(df["App\_Rating"], errors="coerce")

# per-app median then global

df["App\_Rating"] = df.groupby("App\_Name")["App\_Rating"].transform(lambda x: x.fillna(x.median()))

df["App\_Rating"] = df["App\_Rating"].fillna(df["App\_Rating"].median())

# Fill missing text fields

df["Review"] = df["Review"].fillna("No Review")

df["Category"] = df["Category"].fillna("Unknown")

df["Source"] = df["Source"].fillna("unknown")

df["App\_Name"] = df["App\_Name"].fillna("unknown\_app")

# Normalize (lowercase/strip for processing); keep a display copy if needed

for col in ["App\_Name", "Review", "Category", "Source"]:

df[col] = df[col].astype(str).str.strip()

# Bound ratings to [1, 5]

if "App\_Rating" in df.columns:

df = df[(df["App\_Rating"] >= 1.0) & (df["App\_Rating"] <= 5.0)]

# Add placeholder flag

df["Is\_Empty\_Review"] = df["Review"].apply(is\_placeholder\_text)

df.to\_csv(CLEAN\_CSV, index=False)

return df

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

# 3) SENTIMENT (ML, BATCHED)

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

def run\_ml\_sentiment(df: pd.DataFrame, text\_col="Review", max\_len=512, batch\_size=32) -> pd.DataFrame:

device = 0 if torch.cuda.is\_available() else -1

clf = pipeline("sentiment-analysis", model=MODEL\_SENTIMENT, device=device)

texts = df[text\_col].astype(str).tolist()

results = []

for batch, i0 in tqdm(batch\_iter(texts, batch\_size), total=math.ceil(len(texts)/batch\_size), desc="ML Sentiment"):

# truncate to max\_len (rough; tokenization occurs inside pipeline)

batch = [t[:max\_len] for t in batch]

try:

out = clf(batch)

except Exception as e:

# fallback: neutral for the whole batch on error

out = [{"label": "NEUTRAL", "score": 0.0} for \_ in batch]

for r in out:

results.append((map\_sst2\_label(r.get("label", "")), float(r.get("score", 0.0))))

df = df.copy()

df["Sentiment\_Label"] = [r[0] for r in results]

df["Sentiment\_Confidence"] = [r[1] for r in results]

df.to\_csv(ML\_SENT\_CSV, index=False)

return df

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

# 4) HYBRID SENTIMENT (text if available, else rating)

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

def apply\_hybrid\_sentiment(df: pd.DataFrame) -> pd.DataFrame:

df = df.copy()

# If review is placeholder/empty → fall back to rating

def hybrid(row):

if not row.get("Is\_Empty\_Review", False):

# use ML result

return row.get("Sentiment\_Label", "NEUTRAL"), float(row.get("Sentiment\_Confidence", 0.0))

# rating fallback

rating = float(row.get("App\_Rating", 0.0)) if not pd.isna(row.get("App\_Rating", np.nan)) else 0.0

if rating >= RATING\_POS:

return "POSITIVE", 1.0

elif rating <= RATING\_NEG:

return "NEGATIVE", 1.0

else:

return "NEUTRAL", 1.0

hybrid\_labels, hybrid\_conf = [], []

for \_, r in df.iterrows():

lab, conf = hybrid(r)

hybrid\_labels.append(lab)

hybrid\_conf.append(conf)

df["Hybrid\_Sentiment\_Label"] = hybrid\_labels

df["Hybrid\_Sentiment\_Confidence"] = hybrid\_conf

df.to\_csv(HYBRID\_SENT\_CSV, index=False)

return df

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

# 5) ZERO-SHOT TOPIC TAGGING (BATCHED)

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

def zero\_shot\_topics(df: pd.DataFrame, text\_col="Review", batch\_size=16) -> pd.DataFrame:

device = 0 if torch.cuda.is\_available() else -1

zsp = pipeline("zero-shot-classification", model=MODEL\_ZS, device=device)

# Only run on non-placeholder text to save time

mask = ~df["Is\_Empty\_Review"].fillna(False)

texts = df.loc[mask, text\_col].astype(str).str.slice(0, 512).tolist()

idxs = df.index[mask].tolist()

topic\_col = pd.Series(index=df.index, dtype=object)

topics\_multi = []

for batch, i0 in tqdm(batch\_iter(texts, batch\_size), total=math.ceil(len(texts)/batch\_size), desc="Zero-Shot Topics"):

try:

out = zsp(batch, candidate\_labels=ZS\_LABELS, multi\_label=ZS\_MULTI\_LABEL)

except Exception as e:

# mark all as unknown on error

if ZS\_MULTI\_LABEL:

out = [{"labels": [], "scores": []} for \_ in batch]

else:

out = [{"labels": ["unclassified"], "scores": [0.0]} for \_ in batch]

for j, res in enumerate(out):

if ZS\_MULTI\_LABEL:

# keep all labels above threshold

kept = [lab for lab, sc in zip(res.get("labels", []), res.get("scores", [])) if sc >= ZS\_SCORE\_THRESH]

topics\_multi.append(kept if kept else ["unclassified"])

else:

topics\_multi.append([res.get("labels", ["unclassified"])[0]])

# Store best single topic and list of topics

best\_single = [ts[0] if ts else "unclassified" for ts in topics\_multi]

topic\_col.loc[idxs] = best\_single

df = df.copy()

df["Topic"] = topic\_col.fillna("unclassified")

df["Topics\_All"] = None

# Insert lists; pandas will store as object (JSON-serializable on save if needed)

for k, ix in enumerate(idxs):

df.at[ix, "Topics\_All"] = topics\_multi[k]

df.to\_csv(TOPIC\_CSV, index=False)

return df

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

# 6) SUMMARIZATION (NEGATIVE REVIEWS PER TOPIC)

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

def summarize\_negative\_by\_topic(df: pd.DataFrame) -> None:

device = 0 if torch.cuda.is\_available() else -1

summarizer = pipeline("summarization", model=MODEL\_SUM, device=device)

neg = df[df["Hybrid\_Sentiment\_Label"] == "NEGATIVE"].copy()

topics = sorted(t for t in neg["Topic"].dropna().unique() if t != "unclassified")

with open(NEG\_SUMMARY\_TXT, "w") as f:

for t in topics:

reviews = (

neg.loc[neg["Topic"] == t, "Review"]

.dropna()

.astype(str)

.tolist()

)[:MAX\_REVIEWS\_PER\_TOPIC\_FOR\_SUMMARY]

if len(reviews) < 3:

f.write(f"Topic: {t}\nSummary: Not enough data to summarize.\n\n")

continue

text = " ".join(reviews)[:2048] # truncate safety

try:

s = summarizer(text, max\_length=SUMMARY\_MAX\_LEN, min\_length=SUMMARY\_MIN\_LEN, do\_sample=False)

summary = s[0]["summary\_text"]

except Exception:

summary = "Summarization failed."

f.write(f"Topic: {t}\nSummary: {summary}\n\n")

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

# 7) N-GRAM MINING (NEGATIVE/NEUTRAL)

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

def mine\_ngrams(df: pd.DataFrame, ngram\_range=(2, 3), top\_k=50) -> pd.DataFrame:

focus = df[df["Hybrid\_Sentiment\_Label"].isin(["NEGATIVE", "NEUTRAL"])].copy()

if focus.empty:

pd.DataFrame({"Feature": [], "Frequency": []}).to\_csv(NGRAMS\_CSV, index=False)

return pd.DataFrame()

vec = CountVectorizer(

ngram\_range=ngram\_range,

stop\_words="english",

max\_features=top\_k

)

X = vec.fit\_transform(focus["Review"].astype(str))

feats = vec.get\_feature\_names\_out()

counts = X.sum(axis=0).A1

freq = pd.DataFrame({"Feature": feats, "Frequency": counts}).sort\_values("Frequency", ascending=False)

freq.to\_csv(NGRAMS\_CSV, index=False)

return freq

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

# 8) FEATURE-REQUEST FLAGS (REGEX)

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

FEATURE\_PATTERNS = [

r"\bi wish\b",

r"\bwould love\b",

r"\bplease add\b",

r"\bwould be great\b",

r"\bit would be nice\b",

r"\bplease include\b",

r"\bhope you can\b",

r"\bcan you add\b",

r"\bmissing\b",

r"\bshould have\b",

r"\bmust have\b",

r"\bneed(?:s)?\s+(?:feature|option|support)\b",

r"\badd\s+(?:support|option|feature)\b",

r"\bwould be helpful\b",

r"\bshould include\b",

r"\bcould use\b",

]

def flag\_feature\_requests(df: pd.DataFrame) -> pd.DataFrame:

pat = re.compile("|".join(FEATURE\_PATTERNS), flags=re.IGNORECASE)

out = df.copy()

out["Feature\_Request"] = out["Review"].fillna("").astype(str).apply(lambda x: bool(pat.search(x)))

out[out["Feature\_Request"]].to\_csv(FEATURE\_REQUESTS\_CSV, index=False)

return out

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

# 9) APP-LEVEL SUMMARY

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

def build\_app\_summary(df: pd.DataFrame) -> pd.DataFrame:

d = df.copy()

d["App\_Name"] = d["App\_Name"].astype(str).str.strip().str.lower()

# sentiment distribution per app (normalized)

sent = (

d.groupby("App\_Name")["Hybrid\_Sentiment\_Label"]

.value\_counts(normalize=True)

.unstack(fill\_value=0)

.rename(columns={

"POSITIVE": "Positive(%)", "NEGATIVE": "Negative(%)", "NEUTRAL": "Neutral(%)"

})

)

# total reviews per app

counts = d["App\_Name"].value\_counts().rename("Total\_Reviews").to\_frame()

# most common topic per app (if any topics)

if "Topic" in d.columns:

top\_topic = (

d.groupby("App\_Name")["Topic"]

.agg(lambda x: x.value\_counts().idxmax() if not x.dropna().empty else "unclassified")

.rename("Top\_Topic")

)

else:

top\_topic = pd.Series(dtype=object, name="Top\_Topic")

summary = sent.merge(counts, left\_index=True, right\_index=True, how="left")

if not top\_topic.empty:

summary = summary.merge(top\_topic, left\_index=True, right\_index=True, how="left")

for col in ["Positive(%)", "Negative(%)", "Neutral(%)"]:

if col in summary.columns:

summary[col] = (summary[col] \* 100).round(1)

summary = summary.sort\_values("Total\_Reviews", ascending=False).reset\_index().rename(columns={"index": "App\_Name"})

summary.to\_csv(APP\_SUMMARY\_CSV, index=False)

return summary

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

# MAIN

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

def main():

print("📥 Ingesting & standardizing ...")

df = ingest\_and\_standardize()

print("🧹 Cleaning ...")

df = clean(df)

print("💬 Running ML sentiment (batched) ...")

df = run\_ml\_sentiment(df)

print("🔀 Applying hybrid sentiment fallback ...")

df = apply\_hybrid\_sentiment(df)

print("🏷️ Zero-shot topic tagging (batched) ...")

df = zero\_shot\_topics(df)

print("🧾 Summarizing negative reviews per topic ...")

summarize\_negative\_by\_topic(df)

print("🧩 Mining top n-grams (neg/neutral) ...")

\_ = mine\_ngrams(df)

print("✨ Flagging feature requests ...")

df = flag\_feature\_requests(df)

print("📊 Building app-level summary ...")

\_ = build\_app\_summary(df)

print("\n✅ Done! Artifacts saved in:", OUT\_DIR.resolve())

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

main()