*# Community Pulse: Automated Sentiment Analysis from Reddit for East Bay Civic Insights*

*# Created based on feasibility study by Hung Lu & Jesse Katz*

*# CIS 96L NLP - Professors Sanjay Dorairaj & Tuan Nguyen*

import os

import praw

import pandas as pd

import datetime

import time

import re

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import WordNetLemmatizer

from transformers import pipeline, AutoModelForSequenceClassification, AutoTokenizer

import spacy

import streamlit as st

import plotly.express as px

import plotly.graph\_objects as go

from bertopic import BERTopic

from wordcloud import WordCloud

import matplotlib.pyplot as plt

import folium

from folium.plugins import HeatMap

from streamlit\_folium import folium\_static

import pymongo

from pymongo import MongoClient

import json

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from collections import Counter

*# Initialize NLTK components*

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

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

lemmatizer = WordNetLemmatizer()

*# Load spaCy for NER*

nlp = spacy.load('en\_core\_web\_sm')

*# Initialize sentiment analysis model*

sentiment\_model = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

*# Reddit API Configuration*

**def** configure\_reddit\_api():

"""Configure and return Reddit API client"""

reddit = praw.Reddit(

client\_id=os.environ.get('REDDIT\_CLIENT\_ID', 'Fj0Sk2BLVJ0PCBA\_r41zEg'),

client\_secret=os.environ.get('REDDIT\_CLIENT\_SECRET', 'arhY7vtO1j-XiMsyyiwl4kKWI\_PLeQ'),

user\_agent="Community Pulse by /u/Monk481"

)

return reddit

*# Database Configuration*

**def** configure\_database():

"""Configure and return MongoDB client"""

try:

client = MongoClient(os.environ.get('MONGODB\_URI', 'mongodb://localhost:27017/'))

db = client['community\_pulse']

return db

except Exception as e:

st.error(**f**"Failed to connect to MongoDB: {e}")

return None

*# Text Preprocessing*

**def** preprocess\_text(text):

"""Clean and preprocess text data"""

if not isinstance(text, str):

return ""

*# Convert to lowercase and remove URLs*

text = text.lower()

text = re.sub(**r**'http\S+', '', text)

*# Remove special characters and digits*

text = re.sub(**r**'[^a-zA-Z\s]', '', text)

*# Tokenize*

tokens = word\_tokenize(text)

*# Remove stopwords and lemmatize*

cleaned\_tokens = [lemmatizer.lemmatize(token) for token in tokens if token not in stop\_words]

return ' '.join(cleaned\_tokens)

*# Data Collection*

**def** scrape\_reddit\_data(subreddits=['oakland', 'eastbay', 'BayArea'], time\_filter='week', limit=100):

"""Scrape data from specified subreddits"""

reddit = configure\_reddit\_api()

all\_posts = []

for subreddit\_name in subreddits:

try:

subreddit = reddit.subreddit(subreddit\_name)

*# Get posts*

for post in subreddit.top(time\_filter=time\_filter, limit=limit):

post\_data = {

'id': post.id,

'title': post.title,

'text': post.selftext,

'created\_utc': datetime.datetime.fromtimestamp(post.created\_utc),

'score': post.score,

'comments': post.num\_comments,

'subreddit': subreddit\_name,

'url': post.url,

'author': str(post.author),

'processed\_text': preprocess\_text(**f**"{post.title} {post.selftext}"),

}

*# Get top comments*

post.comments.replace\_more(limit=0)

post\_data['top\_comments'] = []

for comment in post.comments.list()[:10]: *# Get top 10 comments*

if hasattr(comment, 'body'):

comment\_data = {

'id': comment.id,

'text': comment.body,

'created\_utc': datetime.datetime.fromtimestamp(comment.created\_utc),

'score': comment.score,

'processed\_text': preprocess\_text(comment.body),

}

post\_data['top\_comments'].append(comment\_data)

all\_posts.append(post\_data)

except Exception as e:

st.error(**f**"Error scraping r/{subreddit\_name}: {e}")

return all\_posts

*# Sentiment Analysis*

**def** analyze\_sentiment(text):

"""Analyze sentiment of text using transformer model"""

if not text or text.strip() == "":

return {"label": "neutral", "score": 0.5}

try:

result = sentiment\_model(text[:512]) *# Truncate to avoid token limit*

return result[0]

except Exception as e:

print(**f**"Error in sentiment analysis: {e}")

return {"label": "neutral", "score": 0.5}

*# Named Entity Recognition*

**def** extract\_entities(text):

"""Extract entities like locations, organizations from text"""

if not text or text.strip() == "":

return []

doc = nlp(text)

entities = {}

for ent in doc.ents:

if ent.label\_ not in entities:

entities[ent.label\_] = []

entities[ent.label\_].append(ent.text)

return entities

*# Topic Modeling*

**def** extract\_topics(texts, n\_topics=10):

"""Extract main topics from a collection of texts"""

if not texts or len(texts) < 5: *# Need sufficient data for topic modeling*

return [{"id": 0, "name": "Insufficient data", "keywords": []}]

try:

*# Initialize BERTopic*

topic\_model = BERTopic(nr\_topics=n\_topics)

*# Fit the model (might take time for large datasets)*

topics, \_ = topic\_model.fit\_transform(texts)

*# Get topic information*

topic\_info = topic\_model.get\_topic\_info()

topic\_keywords = {}

for topic in topic\_info['Topic']:

if topic != -1: *# Skip outlier topic*

words = [word for word, \_ in topic\_model.get\_topic(topic)]

topic\_keywords[topic] = words[:5] *# Get top 5 keywords*

*# Format for return*

formatted\_topics = [

{"id": topic, "name": **f**"Topic {topic}", "keywords": keywords}

for topic, keywords in topic\_keywords.items()

]

return formatted\_topics

except Exception as e:

print(**f**"Error in topic modeling: {e}")

return [{"id": 0, "name": "Error in topic modeling", "keywords": []}]

*# Keyword Extraction*

**def** extract\_keywords(text, n=10):

"""Extract key terms using TF-IDF"""

if not text or text.strip() == "":

return []

try:

*# Use TF-IDF for keyword extraction*

vectorizer = TfidfVectorizer(max\_features=n)

tfidf\_matrix = vectorizer.fit\_transform([text])

*# Get feature names*

feature\_names = vectorizer.get\_feature\_names\_out()

*# Get scores*

scores = tfidf\_matrix.toarray()[0]

*# Create tuples of terms and scores*

tuples = [(feature\_names[i], scores[i]) for i in range(len(feature\_names))]

*# Sort by score*

tuples.sort(key=**lambda** x: x[1], reverse=True)

*# Return top n keywords*

return [word for word, score in tuples[:n]]

except Exception as e:

print(**f**"Error extracting keywords: {e}")

return []

*# Store data in MongoDB*

**def** store\_data(data, db):

"""Store scraped and processed data in MongoDB"""

if not db:

return False

try:

posts\_collection = db['posts']

*# Add sentiment, entities, and keywords to posts*

for post in data:

*# Analyze sentiment for post*

sentiment = analyze\_sentiment(post['processed\_text'])

post['sentiment'] = sentiment

*# Extract entities*

post['entities'] = extract\_entities(post['processed\_text'])

*# Extract keywords*

post['keywords'] = extract\_keywords(post['processed\_text'])

*# Process comments*

for comment in post['top\_comments']:

comment\_sentiment = analyze\_sentiment(comment['processed\_text'])

comment['sentiment'] = comment\_sentiment

*# Store timestamp for tracking*

post['processed\_at'] = datetime.datetime.now()

*# Store in MongoDB (update if exists, insert if new)*

posts\_collection.update\_one(

{'id': post['id']},

{'$set': post},

upsert=True

)

return True

except Exception as e:

st.error(**f**"Error storing data: {e}")

return False

**def** load\_data(db, time\_filter='week'):

"""Load data from MongoDB with optional time filter"""

if not db:

return []

try:

posts\_collection = db['posts']

*# Calculate time threshold*

if time\_filter == 'day':

threshold = datetime.datetime.now() - datetime.timedelta(days=1)

elif time\_filter == 'week':

threshold = datetime.datetime.now() - datetime.timedelta(weeks=1)

elif time\_filter == 'month':

threshold = datetime.datetime.now() - datetime.timedelta(days=30)

else: *# all time*

threshold = datetime.datetime(2000, 1, 1)

*# Query database*

cursor = posts\_collection.find({'created\_utc': {'$gte': threshold}})

*# Convert to list and handle MongoDB ObjectId*

posts = list(cursor)

for post in posts:

post['\_id'] = str(post['\_id'])

return posts

except Exception as e:

st.error(**f**"Error loading data: {e}")

return []

*# Generate insights*

**def** generate\_insights(posts):

"""Generate insights from posts data"""

if not posts:

return {

"sentiment\_summary": {"positive": 0, "negative": 0, "neutral": 0},

"top\_topics": [],

"trending\_keywords": [],

"location\_mentions": {},

}

insights = {}

*# Sentiment analysis summary*

sentiment\_counts = {"positive": 0, "negative": 0, "neutral": 0}

for post in posts:

sentiment = post.get('sentiment', {}).get('label', 'neutral')

if sentiment == "POSITIVE":

sentiment\_counts["positive"] += 1

elif sentiment == "NEGATIVE":

sentiment\_counts["negative"] += 1

else:

sentiment\_counts["neutral"] += 1

insights["sentiment\_summary"] = sentiment\_counts

*# Extract all preprocessed texts for topic modeling*

all\_texts = [post['processed\_text'] for post in posts if post.get('processed\_text')]

*# Get topics*

if len(all\_texts) > 5: *# Need at least 5 documents for meaningful topics*

insights["top\_topics"] = extract\_topics(all\_texts)

else:

insights["top\_topics"] = []

*# Get trending keywords*

all\_keywords = []

for post in posts:

keywords = post.get('keywords', [])

all\_keywords.extend(keywords)

keyword\_counter = Counter(all\_keywords)

insights["trending\_keywords"] = keyword\_counter.most\_common(15)

*# Get location mentions*

location\_mentions = {}

for post in posts:

entities = post.get('entities', {})

locations = entities.get('GPE', []) + entities.get('LOC', [])

for location in locations:

if location.lower() not in location\_mentions:

location\_mentions[location.lower()] = 0

location\_mentions[location.lower()] += 1

insights["location\_mentions"] = location\_mentions

return insights

*# Visualization Functions*

**def** create\_sentiment\_chart(data):

"""Create sentiment distribution chart"""

sentiment\_data = data["sentiment\_summary"]

fig = px.pie(

values=[sentiment\_data["positive"], sentiment\_data["negative"], sentiment\_data["neutral"]],

names=["Positive", "Negative", "Neutral"],

title="Sentiment Distribution",

color\_discrete\_sequence=["#2E8B57", "#CD5C5C", "#6495ED"]

)

return fig

**def** create\_wordcloud(keywords):

"""Create wordcloud from keywords"""

if not keywords:

return None

*# Create a dictionary of words and their frequencies*

word\_freq = {}

for word, count in keywords:

word\_freq[word] = count

*# Generate wordcloud*

wordcloud = WordCloud(

width=800,

height=400,

background\_color='white',

colormap='viridis'

).generate\_from\_frequencies(word\_freq)

*# Create figure*

fig, ax = plt.subplots(figsize=(10, 5))

ax.imshow(wordcloud, interpolation='bilinear')

ax.axis('off')

return fig

**def** create\_topic\_chart(topics):

"""Create horizontal bar chart for topic distribution"""

if not topics:

return None

topic\_names = [**f**"Topic {topic['id']}: {', '.join(topic['keywords'][:3])}" for topic in topics]

topic\_sizes = [1] \* len(topics) *# Placeholder for actual topic sizes*

fig = px.bar(

x=topic\_sizes,

y=topic\_names,

orientation='h',

title="Topic Distribution",

labels={"x": "Count", "y": "Topic"}

)

return fig

**def** create\_location\_map(location\_data):

"""Create a map with location mentions"""

if not location\_data:

return None

*# East Bay area map centered roughly on Oakland*

m = folium.Map(location=[37.8044, -122.2711], zoom\_start=11)

*# Placeholder for demo - in a real app, you'd geocode these locations*

*# This is a mockup for visualization purposes*

sample\_locations = {

"oakland": [37.8044, -122.2711],

"berkeley": [37.8715, -122.2730],

"alameda": [37.7652, -122.2416],

"richmond": [37.9358, -122.3478],

"hayward": [37.6688, -122.0810],

"san leandro": [37.7249, -122.1561],

"emeryville": [37.8312, -122.2852],

"el cerrito": [37.9156, -122.3108],

"albany": [37.8868, -122.2977],

"castro valley": [37.6941, -122.0858],

"san lorenzo": [37.6810, -122.1244],

"concord": [37.9779, -122.0301],

"walnut creek": [37.9101, -122.0652],

"fremont": [37.5485, -121.9886],

"union city": [37.5936, -122.0438],

"livermore": [37.6819, -121.7680],

"pleasanton": [37.6624, -121.8747],

"dublin": [37.7021, -121.9358],

"piedmont": [37.8243, -122.2316],

"martinez": [38.0194, -122.1341],

"pittsburg": [38.0279, -121.8850],

"antioch": [38.0049, -121.8058],

"brentwood": [37.9319, -121.6958]

}

*# Create heat map data*

heat\_data = []

for location, count in location\_data.items():

location\_lower = location.lower()

if location\_lower in sample\_locations:

coords = sample\_locations[location\_lower]

*# Add a point for each mention*

for \_ in range(count):

heat\_data.append(coords)

*# Add heat map layer*

if heat\_data:

HeatMap(heat\_data).add\_to(m)

return m

*# Streamlit Dashboard*

**def** build\_dashboard():

"""Build and display the Streamlit dashboard"""

*# Set page config*

st.set\_page\_config(

page\_title="Community Pulse",

page\_icon="📊",

layout="wide"

)

*# Header*

st.title("Community Pulse: East Bay Civic Insights")

st.subheader("Automated Sentiment Analysis from Reddit")

*# Sidebar for filters*

st.sidebar.header("Filters")

*# Subreddit selection*

subreddits = st.sidebar.multiselect(

"Select Subreddits",

["oakland", "eastbay", "BayArea", "berkeley", "SanFrancisco"],

default=["oakland", "eastbay", "BayArea"]

)

*# Time filter*

time\_filter = st.sidebar.selectbox(

"Time Period",

["day", "week", "month", "all"],

index=1

)

*# Action buttons*

if st.sidebar.button("Refresh Data"):

with st.spinner("Scraping new data from Reddit..."):

*# Configure database*

db = configure\_database()

*# Scrape new data*

posts = scrape\_reddit\_data(subreddits, time\_filter)

if posts:

*# Store data*

success = store\_data(posts, db)

if success:

st.sidebar.success(**f**"Successfully scraped {len(posts)} posts!")

else:

st.sidebar.error("Failed to store data.")

else:

st.sidebar.error("No data scraped.")

*# Load data for dashboard*

db = configure\_database()

posts = load\_data(db, time\_filter)

*# Main dashboard*

if not posts:

st.info("No data available. Please scrape data using the 'Refresh Data' button.")

else:

*# Generate insights*

insights = generate\_insights(posts)

*# Overview metrics*

col1, col2, col3 = st.columns(3)

with col1:

st.metric("Total Posts", len(posts))

with col2:

avg\_sentiment = (

insights["sentiment\_summary"]["positive"] -

insights["sentiment\_summary"]["negative"]

) / len(posts) if len(posts) > 0 else 0

st.metric("Sentiment Score", **f**"{avg\_sentiment**:.2f**}")

with col3:

total\_comments = sum(post.get('comments', 0) for post in posts)

st.metric("Total Comments", total\_comments)

*# Sentiment chart*

st.subheader("Sentiment Analysis")

sentiment\_chart = create\_sentiment\_chart(insights)

st.plotly\_chart(sentiment\_chart, use\_container\_width=True)

*# Create two columns*

col1, col2 = st.columns(2)

with col1:

*# Wordcloud*

st.subheader("Trending Keywords")

wordcloud\_fig = create\_wordcloud(insights["trending\_keywords"])

if wordcloud\_fig:

st.pyplot(wordcloud\_fig)

else:

st.info("Not enough data for wordcloud.")

with col2:

*# Topic distribution*

st.subheader("Topic Distribution")

topic\_chart = create\_topic\_chart(insights["top\_topics"])

if topic\_chart:

st.plotly\_chart(topic\_chart, use\_container\_width=True)

else:

st.info("Not enough data for topic modeling.")

*# Map visualization*

st.subheader("Geographic Insights")

location\_map = create\_location\_map(insights["location\_mentions"])

if location\_map:

folium\_static(location\_map)

else:

st.info("No location data available.")

*# Recent posts*

st.subheader("Recent Posts")

for i, post in enumerate(posts[:5]): *# Show only 5 most recent posts*

with st.expander(**f**"{post['title']} (from r/{post['subreddit']})"):

st.write(**f**"\*\*Created:\*\* {post['created\_utc']}")

st.write(**f**"\*\*Score:\*\* {post['score']}")

st.write(**f**"\*\*Comments:\*\* {post['comments']}")

*# Show sentiment with color*

sentiment = post.get('sentiment', {}).get('label', 'NEUTRAL')

score = post.get('sentiment', {}).get('score', 0.5)

if sentiment == "POSITIVE":

st.markdown(**f**"\*\*Sentiment:\*\* <span style='color:green'>Positive ({score**:.2f**})</span>", unsafe\_allow\_html=True)

elif sentiment == "NEGATIVE":

st.markdown(**f**"\*\*Sentiment:\*\* <span style='color:red'>Negative ({score**:.2f**})</span>", unsafe\_allow\_html=True)

else:

st.markdown(**f**"\*\*Sentiment:\*\* <span style='color:blue'>Neutral ({score**:.2f**})</span>", unsafe\_allow\_html=True)

*# Show post content*

if post.get('text'):

st.write("\*\*Content:\*\*")

st.write(post['text'][:500] + "..." if len(post['text']) > 500 else post['text'])

*# Show keywords*

if post.get('keywords'):

st.write("\*\*Keywords:\*\*", ", ".join(post.get('keywords', [])[:10]))

*# Run the app*

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

build\_dashboard()