AI for Human Rights: Evaluating the Effectiveness of Custom News Filtering Against Google News Alerts

1. Objective:

The objective of this task is to compare the performance of Google News Alerts with a custom system designed to track events such as killings of women. The custom system leverages AI/ML tools to scan and filter news sources based on specific criteria. The goal is to assess whether the custom system provides better coverage and relevance compared to Google News Alerts.

2. Research Process:

To achieve the objective, the following steps are proposed:

Step 1: Data Collection

- Google News Alerts: Use the `feedparser` library to fetch articles from the Google News Alerts RSS feed provided for the project.

- Custom System: Load the CSV file provided by the custom system, which contains stories processed by the system.

Step 2: Data Preprocessing

- Text Cleaning: Normalize text by converting it to lowercase, removing stop words, and removing unnecessary characters.

- Further Preprocessing: Apply stemming or lemmatization to standardize words.

- Filter Relevant Articles: Use metadata from the custom system (e.g., `above\_threshold`) to filter out irrelevant stories.

Step 3: Matching Stories

- Advanced NLP Techniques: Use Sentence Transformers to generate embeddings for the articles. These embeddings capture the semantic meaning of the text, enabling a more accurate comparison between articles from the two systems.

- Cosine Similarity: Calculate cosine similarity between the embeddings to determine how similar the articles are.

Step 4: Relevance Filtering

- Model Confidence Scores: Use the confidence scores from the custom system to determine which stories are likely relevant.

- Manual Verification: Manually verify the relevance of a subset of articles to establish a ground truth for precision and recall calculations.

Step 5: Metrics for Comparison

- Coverage: Measure the total number of unique events identified by each system.

- Precision: Calculate the percentage of relevant stories (true positives) out of the total stories delivered by each system.

- Recall: Measure the system’s ability to identify all relevant stories (true positives) among the total possible relevant events.

- F1 Score: A combined metric to balance precision and recall.

- Signal-to-Noise Ratio (SNR): Calculate the ratio of relevant stories (signal) to irrelevant ones (noise) for each system.

Step 6: Visualization

- Use histograms and bar plots to visualize the distribution of similarity scores, precision, recall, and other metrics.

3. Technologies and Tools:

- Python Libraries:

- `feedparser`: For fetching articles from Google News Alerts.

- `pandas`: For data manipulation and preprocessing.

- `sentence-transformers`: For generating sentence embeddings.

- `scikit-learn`: For calculating cosine similarity and other metrics.

- `matplotlib`: For creating visualizations.

- Data Visualization: Use `matplotlib` or `seaborn` to create visualizations such as histograms, bar plots, and precision-recall curves.

4. Python Prototype:

The following Python code demonstrates the use of Sentence Transformers to compare articles from Google News Alerts and the custom system.

# Step 1: Install necessary libraries

!pip install feedparser

!pip install sentence-transformers

# Step 2: Fetch Google News Alerts

import feedparser

import pandas as pd

# Fetch Google News Alerts using the RSS feed

google\_news\_feed\_url = "<https://www.google.com/alerts/feeds/00541132964398865042/18046012592204646289>"

google\_news\_feed = feedparser.parse(google\_news\_feed\_url)

# Extract relevant details from the feed, such as titles and links (or descriptions if available)

google\_news\_articles = []

for entry in google\_news\_feed.entries:

google\_news\_articles.append({

'title': entry.title,

'link': entry.link,

'description': entry.summary # Using 'summary' for content, may vary by feed

})

# Convert to a DataFrame for consistency

google\_df = pd.DataFrame(google\_news\_articles)

google\_df['article\_text'] = google\_df['description'] # Simulate article content with 'description'

google\_df.head()

# Step 3: Upload Custom System Data

from google.colab import files

# Upload the custom system data (CSV) directly from your local machine to Google Colab

uploaded = files.upload()

# Load the uploaded CSV file into a DataFrame

df = pd.read\_csv(next(iter(uploaded)))

# Display the first few rows to check the structure of the DataFrame

df.head()

# Step 4: Preprocess and Combine Data

df['article\_text'] = df['url'] # This is a placeholder. Replace with actual content fetching.

df\_relevant = df[df['above\_threshold'] == 't'] # Filter relevant rows

# Add a source column to differentiate between the two systems

df\_relevant['source'] = 'custom\_system'

google\_df['source'] = 'google\_news'

# Combine the two DataFrames

combined\_df = pd.concat([df\_relevant[['article\_text', 'source']], google\_df[['article\_text', 'source']]], ignore\_index=True)

combined\_df.head()

# Step 5: Generate Sentence Embeddings

from sentence\_transformers import SentenceTransformer

import numpy as np

# Load a pre-trained Sentence Transformer model

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

# Generate embeddings for all articles

embeddings = model.encode(combined\_df['article\_text'].tolist(), convert\_to\_tensor=True)

# Convert embeddings to numpy arrays for further processing

embeddings = embeddings.cpu().detach().numpy()

# Step 6: Compute Cosine Similarity

from sklearn.metrics.pairwise import cosine\_similarity

# Compute cosine similarity between all embeddings

cosine\_sim\_matrix = cosine\_similarity(embeddings, embeddings)

# Step 7: Identify Matching Articles

# Set a similarity threshold (e.g., 0.3) to identify matching articles

similarity\_threshold = 0.3

matching\_pairs = []

for i in range(len(combined\_df)):

for j in range(i+1, len(combined\_df)):

if cosine\_sim\_matrix[i, j] > similarity\_threshold and combined\_df.iloc[i]['source'] != combined\_df.iloc[j]['source']:

matching\_pairs.append((combined\_df.iloc[i]['source'], combined\_df.iloc[i]['article\_text'],

combined\_df.iloc[j]['source'], combined\_df.iloc[j]['article\_text'], cosine\_sim\_matrix[i, j]))

# Print out the number of matching pairs found

print(f"Number of matching pairs found: {len(matching\_pairs)}")

# Optionally, print a few matching pairs for inspection

for match in matching\_pairs[:5]: # Show a few examples

print(f"Source 1: {match[0]}")

print(f"Article 1: {match[1]}")

print(f"Source 2: {match[2]}")

print(f"Article 2: {match[3]}")

print(f"Similarity: {match[4]:.2f}\n")

# Step 8: Visualization of the Similarity Scores

import matplotlib.pyplot as plt

similarity\_scores = [match[4] for match in matching\_pairs]

if similarity\_scores:

plt.figure(figsize=(10, 6))

plt.hist(similarity\_scores, bins=10, color='skyblue', edgecolor='black')

plt.title('Distribution of Similarity Scores Between Custom System and Google News Alerts')

plt.xlabel('Similarity Score')

plt.ylabel('Frequency')

plt.show()

else:

print("No matching pairs found to visualize.")

5. Conclusion

This research process proposes a method for comparing the performance of Google News Alerts and a custom system for tracking human rights violations. The process includes data collection, preprocessing, advanced NLP techniques (Sentence Transformers), and metrics for comparison. The prototype code demonstrates one phase of the proposed approach, using Sentence Transformers for text similarity matching and visualizing the results.

This solution should help you assess the hypothesis that your custom system provides better coverage and relevance compared to Google News Alerts. If further refinement is needed, you can consider using more advanced techniques such as topic modeling or clustering to group similar articles.