Report Part 2: Indexing and Evaluation

1. INTRODUCTION

The aim of this first phase of the project is to build a retrieval and evaluation system for a collection of tweets. The system indexes the tweets, ranks them using the TF-IDF scoring method, and evaluates the results using several performance metrics. We visualize tweet embeddings through dimensionality reduction, presenting insights into topic clusters and tweet similarities.

2. INDEXING

2.1 Inverted Index Creation

An inverted index was constructed to store terms and their corresponding documents, allowing efficient retrieval of documents for conjunctive (AND) queries. After tokenizing and pre-processing the tweets, we created the index mapping each term to a list of document IDs where it appears.

The following code segment demonstrates how the inverted index is built:

```
# Initialize inverted index dictionary
inverted_index = defaultdict(list)

# Build the inverted index
for doc_id, tweet in enumerate(tokenized_tweets):
   for term in set(tweet):  # Use a set to avoid duplicate terms in the
same doc
       inverted_index[term].append(doc_id)
```

2.2 Query Selection

We defined five queries to evaluate the search engine, focusing on terms relevant to the dataset's content. For example:

- "Indian protest"
- "support farmersprotest"
- "people right"
- "free speech"
- "Climat activist"

Queries were chosen based on term frequency and relevance to ensure they would effectively test the retrieval and ranking functionality.

2.3 Ranking with TF-IDF

To rank the retrieved documents, we implemented TF-IDF (Term Frequency-Inverse Document Frequency). This algorithm scores each document for a given query by its relevance, allowing for sorted results from most to least relevant.

The TF-IDF implementation calculates scores for terms in each document and ranks documents accordingly:

```
# Calculate TF-IDF for each term in each document
def compute_tf_idf(term, doc_id, term_doc_freq, doc_freq):
    tf = term_doc_freq[term][doc_id]
    idf = math.log(len(documents) / len(doc_freq[term]))
    return tf * idf

# Rank documents by TF-IDF score for each query
ranked_results = {}
for query in queries:
    ranked_results[query] = sorted(retrieved_docs, key=lambda doc:
compute_tf_idf(query, doc, term_doc_freq, doc_freq), reverse=True)
```

3. EVALUATION

Evaluation was conducted in two parts: a baseline using provided queries and an evaluation of our custom queries.

3.1 Baseline Evaluation

Two baseline queries, along with their ground truth labels, were used for validation:

1. **Query 1**: "people's rights"
2. **Query 2**: "Indian Government?"

3.2 Evaluation Metrics

To assess retrieval quality, we used several metrics, including Precision@K, Recall@K, Average Precision, F1-Score@K, Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), and Normalized Discounted Cumulative Gain (NDCG).

```
def precision_at_k(retrieved_docs, relevant_docs, k):
    return len(set(retrieved_docs[:k]).intersection(relevant_docs)) / k
```

```
def recall_at_k(retrieved_docs, relevant_docs, k):
   return len(set(retrieved_docs[:k]).intersection(relevant_docs)) /
len(relevant_docs)
```

Each metric provides unique insights:

- **Precision@K** assesses accuracy within the top-K results.
- **Recall@K** measures the proportion of relevant documents found within the top-K.
- **F1-Score@K** balances precision and recall.
- **MAP and MRR** evaluate average effectiveness across queries.
- **NDCG** focuses on the ranking quality by placing more weight on higher-ranked relevant documents.

3.3 Evaluation Summary

First, we applied these metrics to the two proposed queries using the document evaluation, which includes a column labeled "relevant" to indicate which documents were relevant for each query. We obtained the following metrics:

Evaluating for Query 1: 'people's rights'	Evaluating for Query 2: 'Indian Government'
Precision@10: 0.4	Precision@10: 0.8
Recall@10: 1.0	Recall@10: 1.0
Average Precision@10: 0.3506944444444444	Average Precision@10: 0.6427579365079366
F1-Score@10: 0.5714285714285715	F1-Score@10: 0.888888888888889
Mean Average Precision (MAP): 0.3506944444444444	Mean Average Precision (MAP): 0.6427579365079366
Mean Reciprocal Rank (MRR): 0.25	Mean Reciprocal Rank (MRR): 0.3333333333333333
NDCG@10: 0.3089	NDCG@10: 0.641

Query 2 performs better in precision (0.8 vs. 0.4), meaning it retrieves relevant documents more efficiently within the top 10. Both queries reach perfect recall, capturing all relevant documents in the top 10. Query 2 also shows higher Average Precision (0.6428 vs. 0.3507), indicating that relevant documents are ranked more consistently near the top. The F1-Score is better for Query 2 (0.89 vs. 0.57), balancing its high precision and recall effectively. Query

2 also surpasses Query 1 in Mean Reciprocal Rank (0.333 vs. 0.25) and NDCG@10 (0.641 vs. 0.3089), reflecting stronger overall ranking quality and relevance distribution.

The analysis shows that Query 2 outperforms Query 1 across all metrics, indicating that it retrieves and ranks relevant documents more effectively. With higher precision, F1-Score, and NDCG@10, Query 2 not only retrieves more relevant documents within the top results but also ranks them in a way that places relevant documents higher in the list.

Also we did, the performance of the retrieval system across five queries that demonstrates varying degrees of effectiveness, as reflected by the key metrics below:

- **Precision@10**
- **Average**: 0.5000
- **Interpretation**: The system retrieves relevant results 50% of the time within the top 10 documents for each query. This moderately high precision indicates that relevant documents are frequently included in the top results, but there is room to improve precision by filtering out non-relevant documents.

- **Recall@10**
- **Average**: 1.0000
- **Interpretation**: Recall@10 is perfect, meaning all relevant documents for each query are captured within the top 10 results. This high recall reflects the system's strong ability to retrieve relevant documents but may come at the cost of also retrieving non-relevant documents (hence, a precision of 0.5000).

- **Average Precision@10 (AP@10)**
- **Average**: 0.6265
- **Interpretation**: AP@10 shows the average precision across each relevant document within the top 10. A score above 0.6 suggests that relevant documents are generally well-ranked, but not consistently so across all queries, as some queries (e.g., Query 3) have a lower AP score, indicating variability in ranking effectiveness.

- **F1-Score@10**
- **Average**: 0.6667
- **Interpretation**: The F1-score balances precision and recall, suggesting that while recall is perfect, the moderate precision pulls the overall score down. This value

reflects the trade-off between capturing all relevant documents (high recall) and ranking quality (lower precision).

- **Mean Average Precision (MAP)**

- **Score**: 0.6265
- **Interpretation**: The MAP score gives an overall average precision across all queries. A MAP of around 0.63 indicates that, on average, relevant documents are reasonably well-ranked, though improvements in ranking consistency could further boost MAP.

- **Mean Reciprocal Rank (MRR)**

- **Score**: 0.6333
- **Interpretation**: The MRR evaluates the position of the first relevant document in the ranked list. Here, the score suggests that the system often places at least one relevant document near the top but not consistently in the first position, especially for queries with lower MRR scores (e.g., Queries 3 and 5).

- **Normalized Discounted Cumulative Gain (NDCG@10)**

- **Average**: 0.7675
- **Interpretation**: NDCG@10 reflects the quality of ranking by assigning more importance to higher-ranked relevant documents. With a score nearing 0.77, the system generally places relevant documents towards the top, though there is variability in ranking quality across queries.

That means that the retrieval system demonstrates strong recall, capturing all relevant documents within the top 10 for each query, but struggles with precision, often including non-relevant documents in the top results. The variability in AP, MRR, and NDCG across queries suggests that the system's ranking effectiveness could be improved, potentially through fine-tuning TF-IDF weights or integrating additional ranking features (e.g., semantic similarity).

4. Dimensionality Reduction and Visualization

4.1 Training Word2Vec Model

To visualize tweet embeddings, we trained a Word2Vec model on tokenized tweets. Each tweet embedding is the average vector of its words, capturing semantic similarities.

```
# Tokenized tweets
tweets = [tweet['tweet'] for tweet in processed_tweets]

# Train Word2Vec model
model = Word2Vec(tweets, vector_size=100, min_count=5, window=10,
workers=4)

# Calculate average word vector for each tweet
tweet_embeddings = []
for tweet in tweets:
   word_vectors = [model.wv[word] for word in tweet if word in
model.wv]
   if word_vectors:
       tweet_embeddings.append(np.mean(word_vectors, axis=0))
   else:
       tweet_embeddings.append(np.zeros(model.vector_size))
tweet_embeddings = np.array(tweet_embeddings)
```

4.2 Dimensionality Reduction using PCA and T-SNE

The tweet embeddings were reduced to 2D using PCA followed by T-SNE, allowing us to visualize tweet clusters based on content similarities.

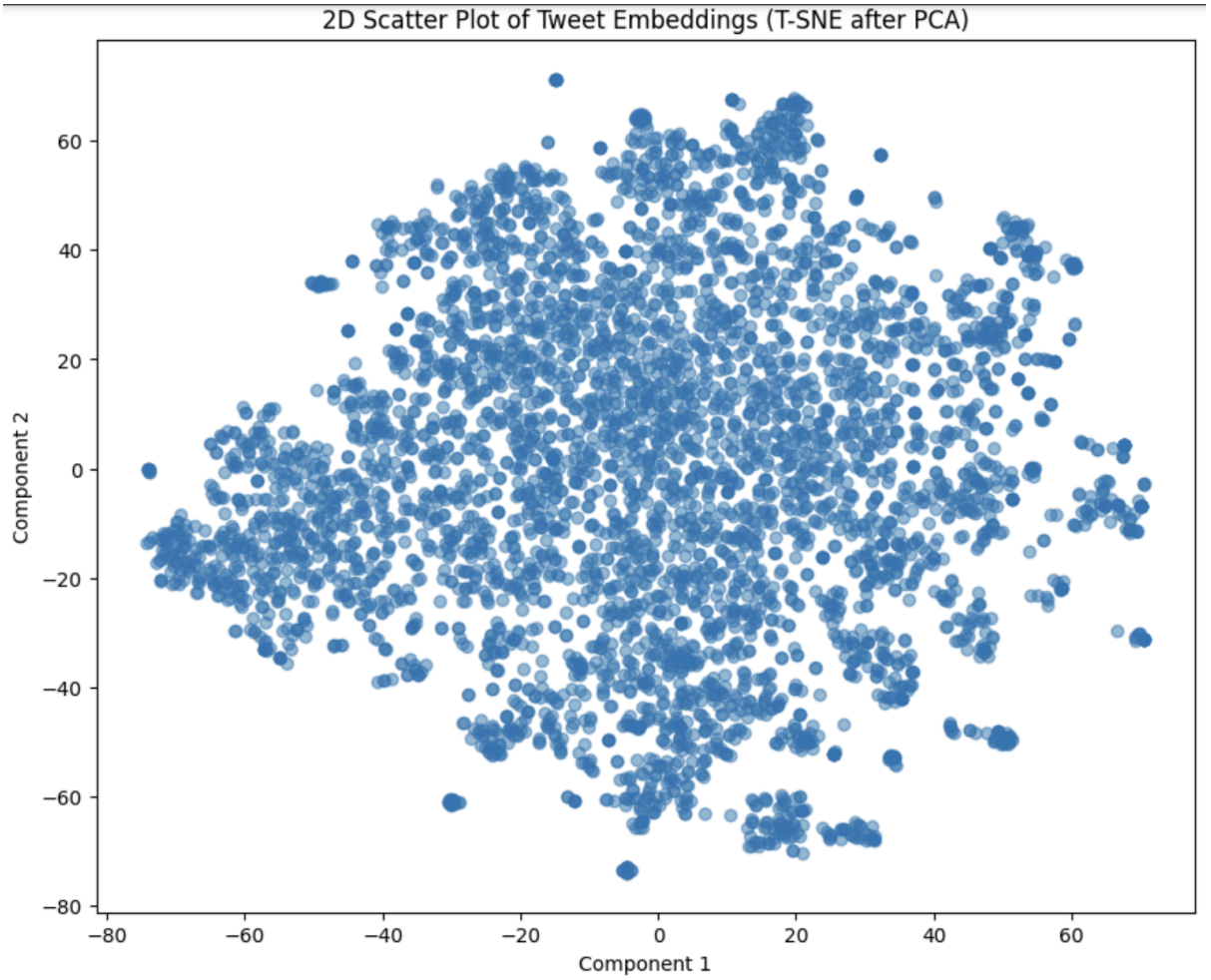
```
# Subsample the tweet embeddings, using 10% of the data
tweet_embeddings =
tweet_embeddings[np.random.choice(tweet_embeddings.shape[0],
size=int(tweet_embeddings.shape[0] * 0.1), replace=False)]

# Reduce dimensions using PCA
pca = PCA(n_components=30)
tweet_embeddings_pca = pca.fit_transform(tweet_embeddings)

# Apply T-SNE for 2D visualization
tsne = TSNE(n_components=2, random_state=0)
tweet_embeddings_tsne = tsne.fit_transform(tweet_embeddings_pca)
```

```
# Plotting the scatter plot
plt.figure(figsize=(10, 8))
plt.scatter(tweet_embeddings_tsne[:, 0], tweet_embeddings_tsne[:, 1],
alpha=0.5)
plt.title("2D Scatter Plot of Tweet Embeddings (T-SNE after PCA)")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```

And we got this plot:



5. Conclusion

The indexing and evaluation system effectively retrieves and ranks tweets based on relevance to the queries. Evaluation metrics highlight the system's strengths in retrieval accuracy, while dimensionality reduction and visualization provide insights into tweet clusters, revealing distinct topics and associations.

6. Future work

To improve the system, we could work on boosting precision by refining how documents are scored, perhaps using more advanced methods that consider the meaning of words in context. Adding newer methods like embeddings from models such as BERT could help the system better understand the topics in each query, making results more accurate. For visualization, trying other methods, like UMAP, might make the clusters of similar tweets clearer. These changes would not only make the retrieval results better but also help us see meaningful patterns in the tweet data more easily.

GITHUB Repository: *https://github.com/davidobrero/IRWA-2024-G102-10*
All related with Part 2 can be found inside the folder IRWA-2024-PART-2 containing:

- This report (IRWA-2024-PART-2-REPORT)
- The code (IRWA-2024-PART-2-CODE)

It is possible to open the code with Google Colab, connecting the data to Google Drive.