Clara Pena - u186416
Yuyan Wang – u199907

Project Report Part 1

Project GitHub URL: https://github.com/yuyanwang03/IRWA

Project GitHub Tag: IRWA-2024-part-2

The steps required to execute the Python Jupyter Notebook are detailed in the README.md file. To improve the user experience and reduce the execution time, we are also providing the preprocessed data in a CSV file. This allows users to bypass the preprocessing stage, which otherwise takes approximately 10 minutes, and jump directly to creating the visualizations.

Indexing

To build the inverted index, as the hint suggested, we have used the practical labs we have been doing, and implemented a function called 'create_index' to build an inverted index. We have taken the function given in Lab 1, and adapted it in order to work with this dataset, as we have already preprocessed the tweets. The function maps each term of each document to a list that includes the document ID and the positions of the term in the document. To do that, we take each term in the document and we build a temporary dictionary, 'current_page_index', that stores what we have mentioned before, the document and position data. Then, we merge this index with the main index 'index'. In this way, we make sure that each term does not have any duplicated data.

The 'create_index' function is the backbone to implement the search functionality in the project, as it allows to look up for terms and retrieve them efficiently, or in other words, it makes it easier to access directly to documents that contain all the terms in a query.

We have also implemented the same exact function of Lab 1 to preprocess each query that we use for testing our index, to ensure consistency; something to say here is that we added an additional step so that it converts abbreviation of verbs to its extended form (e.g. we're -> we are) and are removing ''s' that indicate possessiveness. It lowercases terms, splits the query into terms, removes stopwords and applies stemming to simplify terms to their root forms. In this way, we improve retrieval performance, as we are grouping similar terms under a common root.

Once we have built the inverted index, we have taken the function 'search' and modify it again in order to work with the dataset we have. The function 'search' retrieves all documents that match all terms in a query. We are ensuring that we are implementing a conjunctive search (AND) instead of a disjunctive one, it requires all the terms in the query to be present in the document for it to be considered a match; this is being assured by doing an intersection with the documents instead of a union (look at code for more detail). It first preprocesses the query using the 'build_terms' function, then it retrieves the documents IDs for the term, and finally, it intersects the set of document IDs for each term in the query, so we ensure that we only return the documents that contain all the documents that contain all the query terms.

To simulate the query searches, we got the top 20 terms in the dataset with the function 'get_top_terms', which identifies the top n terms by frequency that are the most commonly discussed topics. It gets the content of the tweets and puts it altogether to then split it into terms and count the frequency of all of them. After having an understanding of the 20 most common terms, we have defined our 5 additional queries:

- "India protest": This query aligns with the high frequency of terms like 'india' (7724 occurrences) and 'protest' (4787 occurrences). It targets the significant discussion around protests in India, capturing a broad yet significant topic within your dataset. This is important for understanding general sentiments or events related to protests in the region.

- "support farmers": Given the prevalence of terms such as 'farmersprotest' (50272 occurrences) and 'support' (6004 occurrences), this query is likely to be highly relevant. It specifically addresses the widespread discourse on supporting farmers, likely related to agricultural policies or farmer welfare, which looks to be a prominent issue in the data.
- "Modi shame": This query is crafted around the high occurrence of 'modi' (3113 occurrences). Including a sentiment or descriptive term like 'shame' might target specific discussions or criticisms related to policies or actions associated with Modi, the Prime Minister of India, offering insights into public sentiment regarding his administration.
- "BJP party": With 'bjp' being mentioned 2649 times, focusing on the political party directly allows for an analysis of content specifically related to the Bharatiya Janata Party. This could reveal discussions on political activities, policies, or public opinions directly connected to the party's actions and governance.
- "human rights violated": Although 'right' appears 3594 times, coupling it with 'violated' expands the context to human rights issues. This query is intended to explore discussions or reports on human rights violations, a topic of critical importance that may encompass various aspects of social and political discourse.

Then, using the function 'simulate_serach', we performed a test search with the queries we predefined to evaluate the search engine we have implemented before. So, the function, given the list of queries and the index we have created, uses the 'search' function to search each query term and get the documents the query appears in. For a more clear visualization, we only display the top 10 results.

Then we added another layer to the search engine by implementing the TF-IDF algorithm to rank our search results, which allows documents with higher relevance (based on term importance) to rank higher in results. The function 'create_index_tfidf' builds an index weighted by TF-IDF that is it calculates the term frequency (TF) and the document frequency (DF) for each term of the dataset, and then it computes the inverse document frequency (IDF) to scale the importance of terms across the corpus.

As well as with the 'create_index' function, we build a temporary index that stores the position of each term in the document and the document ID associated with each term. Then, we compute the norm of the document to normalize term frequencies. Once the norm is calculated, we compute the term frequency and the document frequency. Finally, we merge the temporary index with the main index and compute the inverse document frequency for each term of the index.

In order to use the TF-IDF algorithm function, we created a function called 'rank_documents' that scores and ranks documents based on their TF-IDF similarity to the query. The function computes the TF-IDF vector for the query and each document and then it ranks the documents using cosine similarity to the query. Something to keep in mind here is that we are only ranking documents that contain all the query words, in order to reduce computational costs. Documents that do not contain the query words are default considered unimportant and hence will have a ranking score of 0. After that, cosine similarity helps measure the angle between the query and the document vectors, which effectively captures the relevance by how well terms align across documents and the query.

Evaluation

During the evaluation phase, we have used several metrics to measure the effectiveness of our search engine and assess both the relevance and the ranking quality of the results. We have applied metrics such as Precision@K, Recall@K, Average Precision, F1-Score, Mean Average Precision (MAP), Mean

Reciprocal Rank (MRR), and Normalized Discounted Cumulative Gain (NDCG). Moreover, we have implemented Word2Vec embeddings and T-SNE visualization to observe document similarity and clustering. A characteristic that we would like to remark here is that since it is all about binary relevance, the functions are designed in a way that tries to optimize the space usage. The important parameters are ground_truth and results. The former one is only a list of docIds that are considered relevant for the query (from a sample) and the results list contain pairs of (docScore, docId). The documents' scores are not really used nor printed, but we are providing them in the core for testing purposes.

As a first step, we have prepared the data we used in the evaluation process. First, we merged the evaluation data file with the tweets dataset using the document ID as a key, so we make sure that evaluation labels are together with the document content. With the queries we have been given, we perform a TF-IDF weighted search for 'people's rights' and 'Indian government' with the main english index (the one generated from the very beginning which contains information of all the collection in English), which provides a list of ranked documents.

For our five custom queries we are basically doing the same thing as for the previous queries: we are retrieving the documents that the algorithm returns as relevant and their score. The related data is stored in a dictionary not for any other purpose than to make it easier to manage all 5 queries at the same time.

After defining the function for each evaluation method, we have tested them by defining for each query the following K values (5, 10, 15, 20, 50, 100, 150, total number of results); where the K value does not exceed the total number of retrieved documents; this is to ensure that the metrics can be computed and makes sense in their context.

For the exact metric values, please look at the printed output in the notebook.

Precision@K (P@K)

'precision_at_k' computes the precision for the top K documents, in the search results, which is the proportion of the retrieved documents in the top K that are relevant. This metric measures how well the search engine ranks relevant documents within the top positions.

We start by extracting the top K documents from the results list. Then , we iterate over each document we extracted before and we check if the document ID is in the ground truth. The precision is calculated as the ratio of relevant documents in the top K results to the total number of documents K being considered.

- India Protest Query: It shows decreasing precision as more results are considered, from 0.4 at top 5 to 0.0053 at 951. The sharp drop in precision as the document count increases suggests either a high concentration of relevant documents at the top or a limited pool of relevant documents overall.
- Support Farmers Query: It starts with low precision (0.2 at top 5) and continues to decline as more results are included, reaching very low precision (0.0031) by 3197. This indicates a sparse presence of relevant documents throughout. The search system struggles to find relevant documents even at the start, and the relevance thins out considerably as more results are retrieved.
- Modi Shame Query: Initially, precision is low; however, an interesting increase occurs at Precision@50 (0.16), suggesting pockets of relevant documents are being found deeper in the results. This could indicates some kind of inconsistency in the relevance of retrieved documents, with some relevant results appearing intermittently at larger K values.

- BJP Party Query: This one has a precision of 0 at top 5, which then improves significantly at higher K, reaching a peak at Precision@15 (0.3333). This pattern could indicate that relevant documents are initially missed but appear as more results are considered.
- Human Rights Violated Query: There is a complete absence of relevant documents in the top 20 results, with a slight increase at Precision@50 and beyond. This suggests either a misalignment between the query terms and document indexing or a very scattered presence of relevant information.

Recall@K (R@K)

'recall_at_k' computes the recall for the top K results, which is defined as the fraction of relevant documents within the top K retrieved documents out of the total relevant documents. In this way we can assess how well the search engine finds relevant documents overall, which is important in situations where missing relevant results can be a very big mistake.

In the function, we first check if K is larger than the number of documents in results, then, if so, it is set to the length of results, so we only consider available results. We iterate through the top K results and count how many of the documents are in the ground_truth. The recall is then calculated as the proportion of relevant documents retrieved within the top K results to the total number of relevant documents.

- India Protest Query: The recall remains constant at 0.2 up to Recall@50, indicating that the same proportion of relevant documents is found regardless of the increased number of documents reviewed. A jump is seen at Recall@100 and Recall@150 (0.4), and finally at Recall@951 (0.5). This would basically mean that ost relevant documents are scattered deeper in the result set.
- Support Farmers Query: Recall is persistently low at 0.1 through Recall@150; it is a similar case which indicates that the majority of relevant documents are retrieved only when almost all the dataset is considered.
- Modi Shame Query: Starts low at Recall@5 and gradually increases, with a significant jump to 0.8 by Recall@50, reaching full recall (1.0) by Recall@150. From this we can infer that a relatively high concentration of relevant documents starts to appear from the 50th result onward, showing better clustering of relevant documents compared to other queries.
- BJP Party Query: No relevant documents at Recall@5, then a sharp increase at Recall@10, continuing to improve rapidly to full recall by Recall@128. This suggests that while initial results are completely irrelevant, relevant documents start to appear soon after and are concentrated in the subsequent results.
- Human Rights Violated Query: No recall at the beginning up to Recall@20, with a sudden increase at Recall@50, maintaining at 0.5 until Recall@150. This indicates that half of the relevant documents are reachable mid-way through the results, with all relevant documents only retrievable when a significant portion of the collection is considered.

Average Precision@K (P@K)

'average_precision_at_k' computes the average precision at K, which is basically taking the precision of different positions (the positions of only the documents that are relevant) up to K and averages them, so it gives them a higher weight to rank documents correctly. With this method we analyze how the search engine keeps relevance across multiple rankings.

We iterate through the top K results, and for each result, it checks if the document ID is in the set of the ground truth. If it is, then it means that the document is relevant and we increment the number of relevant retrieved documents by 1, and we compute the precision at position i. Finally, we compute the average precision as the division between the cumulative precision and the total number of relevant documents.

- India Protest Query: The AP remains constant at 0.2000 across most cutoff points, indicating a consistent retrieval of relevant documents throughout the first 50 results.
- Support Farmers Query: It has notably low AP scores consistently at 0.0250 across most evaluations, with a slight increase to 0.0300 at 3197. This pattern indicates that the system struggles to retrieve relevant documents efficiently for this query, and even extensive retrieval yields only a marginal increase in precision.
- Modi Shame Query: Starting with a low AP of 0.1000, there is a noticeable improvement to 0.1796 at 50 documents, which slightly increases to 0.1930 by 150. From here we can interpret that the system improves in retrieving more relevant documents as more results are processed, suggesting effective retrieval but delayed appearance of some relevant documents.
- BJP Party Query: Starting with an AP of 0.0000 at 5, there's a significant jump to 0.0346 by 10 and a more substantial increase to 0.2341 by 128. This progression indicates a back-loaded retrieval performance, where relevant documents start appearing more frequently after an initial miss.
- Human Rights Violated Query: It shows no retrieval of relevant documents up to 20 results, with a minor improvement to 0.0369 at 50. Similar to Query 2, there is either a scarcity of relevant documents or an inefficiency in their retrieval and ranking.

F1-Score@K

'f1_score_at_k' is a function that calculates the F1-score that is a harmonic mean of precision and recall at K, which offers a balanced measure that captures false positives and false negatives. We are assuming that we want to compute the F1-Score where precision is as important as recall.

Again, we iterate through each document extracted from the top K results, and if the document ID is in the ground truth set, we increment the number of relevant retrieved documents by 1. Then, we compute the precision by dividing the number of relevant retrieved documents by the number of documents extracted from the top K results. We calculate the recall by dividing the number of relevant retrieved documents by the number of relevant documents for the query. Finally, the F1 score is computed as twice the product of precision and recall, divided by the sum of precision and recall.

- India Protest Query: The highest F1 Score is at th.e smallest cutoff, F1Score@5 = 0.2667, and it decreases as more results are included. Hence the system seems effective at retrieving a few relevant documents early but struggles to maintain relevance as more results are considered.
- Support Farmers Query: It has consistently low F1 Scores across all cutoffs, with very minor improvement at an extensive depth of 3197. This could indicate a poor balance of precision and recall throughout, reflecting overall inefficiency in retrieving relevant documents.
- Modi Shame Query: It initially has low F1 Scores that increase significantly at F1Score@50. This would mean that while the initial retrieval is poor, relevance improves significantly at deeper result levels.
- BJP Party Query: It starts with zero relevance at F1Score@5, followed by a marked improvement to a peak at F1Score@15, followed by a gradual decline. This shows an effective retrieval of relevant documents at a specific range but not at the very top or as results continue to expand.
- Human Rights Violated Query: No relevant documents are retrieved in the top 20 results, with a sudden improvement starting from F1Score@50. This may be due to a significant delay in retrieving relevant documents.

Clara Pena - u186416
Yuyan Wang – u199907

Mean Average Precision (MAP)

'mean_average_precision_at_k' averages the average precision for multiple queries, which provides another precision measure among all queries. With this metric we evaluate the effectiveness across multiple queries.

We start by iterating over pairs of the ground truth and results of a specific query, and for each query, we calculate the average precision at K using the previous function we have defined 'average_precision_at_k', which returns a score that represents how well the search engine has ranked relevant documents within the top K results for the current query. Then, we compute the mean average precision by dividing the scores we got earlier by the number of queries.

Obtained results for our self defined queries are: MAP@5: 0.0650; MAP@10: 0.0719; MAP@15: 0.0927; MAP@20: 0.0927; MAP@50: 0.1333; MAP@100: 0.1347; MAP@150: 0.1392.

The results reveals a gradual improvement in the system's effectiveness in retrieving relevant documents as more results are considered, beginning with notably low effectiveness at MAP@5 and seeing incremental increases up to MAP@150. This pattern indicates that while the system can eventually find a significant number of relevant documents, it struggles to rank them effectively in the top positions, suggesting that relevant documents are often dispersed across a broader range of search results.

Mean Reciprocal Rank (MRR)

'mean_reciprocal_rank' computes the average reciprocal rank for the first relevant documents across multiple queries, which measures how early relevant documents appear in the results. Using MRR to evaluate the search engine shows how well it ranks relevant documents near the top of results.

In the function , we iterate again over pairs of the ground truth and results of a specific query and we create another loop inside, where we iterate through the results for the current query, so if the document ID of the current result is in the ground truth set, then the document is relevant, so we set 'reciprical_rank' to 1/'rank' where rank is the position of the first relevant document in the results. Then, we compute the mean reciprocal rank as the average of all values of the reciprocal ranks we computed before.

The obtained MRR for the five queries is 0.4775. This indicates a moderate level of effectiveness in the search system's ability to rank the first relevant document highly across different queries. This score suggests that on average, the first relevant document appears around the second position in the search results.

Normalized Discounted Cumulative Gain (NDCG)

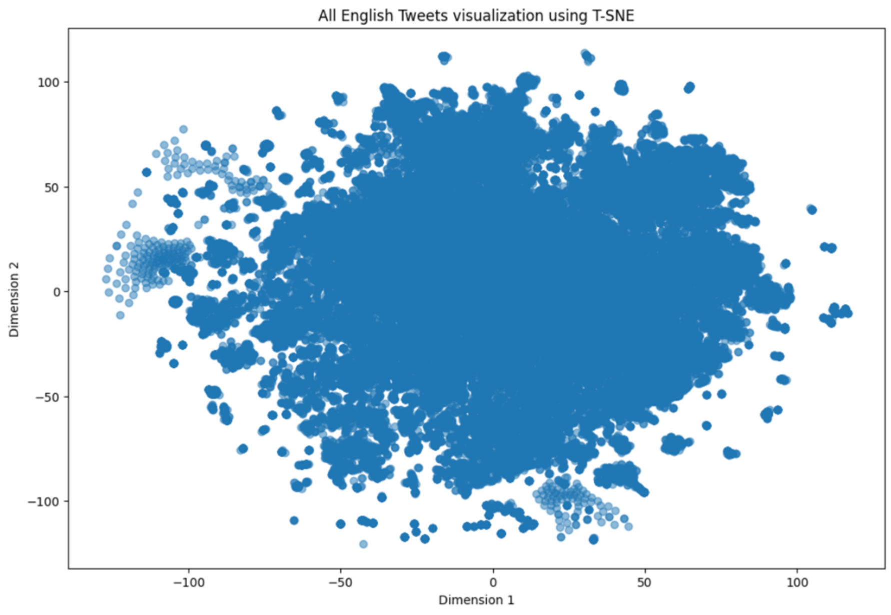
'ndcg_at_k' evaluates ranking quality by assigning higher scores to relevant documents appearing near the top results. Using NDCG is crucial for ranking-based evaluation, rewarding systems that rank relevant documents higher.

For this evaluation metric, we have defined an auxiliar function 'dcg_at_k' to compute the discounted cumulative gain. What the function is doing is computing the DCG either using the standard method or an alternative method, we are providing this because we are given 2 formulas to compute it from theory; however, they end up in the same result. Then, in 'ndcg_at_k', we are computing the normalized discounted cumulative gain by, first, calculating the binary relevance for the actual results, where each document in the top K results is assigned a relevance score of 1 if it is in the ground truth set or 0, otherwise. We call the previous function 'dcg_at_k' to compute the DCG for the actual results based on their relevance scores and, we also compute the ideal DCG by sorting the relevance for the actual results in descending order to represent the best-case scenario for ranking and then we call again the 'dcg_at_k' function using the sorted relevance scores. Finally, we return the normalized DCG, which is the ratio of the actual DCG to the ideal DCG.

- India Protest Query: It achieves perfect NDCG scores (1.0000) up to NDCG@50; this shows exceptional relevance and ranking performance in the top results, with a gradual decline in deeper results.
- Support Farmers Query: It starts with moderate scores, reflecting adequate relevance in the initial results; however, it exhibits a decline or inconsistency at deeper results, indicating issues with maintaining relevance at depth.
- Modi Shame Query: Similar to Query 1, it has perfect scores up to NDCG@20, and has notable drops at deeper cuts.
- BJP Party Query: It displays a significant progression from no relevant documents at NDCG@5 to moderately high scores at deeper cutoffs; this means that relevant documents are present but poorly ranked initially.
- Human Rights Violated Query: The NDCG scores of 0.0000 from NDCG@5 through NDCG@20 highlight a complete failure in either identifying or ranking any relevant documents. A score of 0.2639 at NDCG@50 indicates that some relevant documents are present in the results but only start appearing significantly deeper in the list.

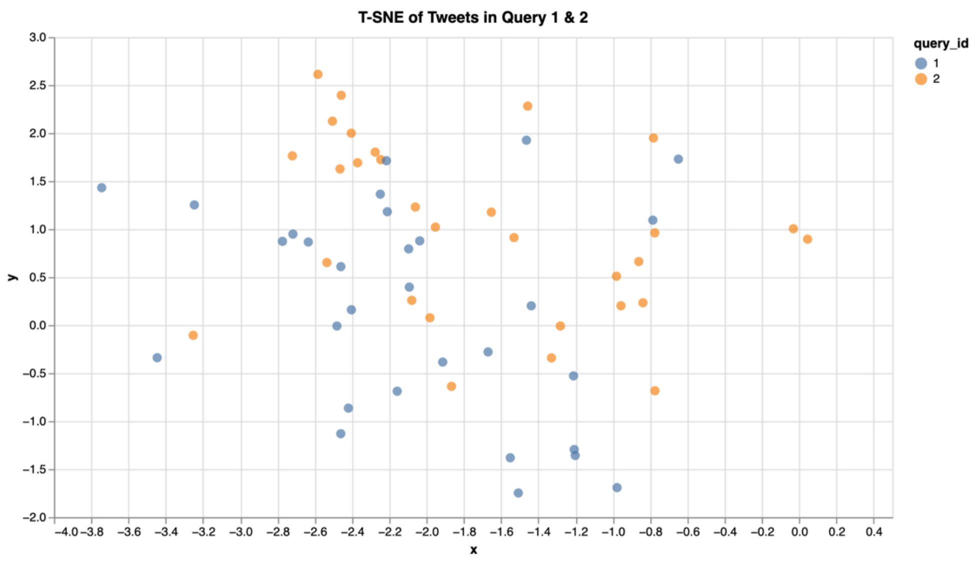
Vector Representation using T-distributed Stochastic Neighbor Embedding (T-SNE)

To represent the tweets in a two-dimensional scatter plot through the T-NSE algorithm we have chosen word2vec to capture semantic relationships between words. Using word2vec has allowed us to perform a deeper semantic analysis by creating vector representations of words that capture the semantic relationships based on the context, so words that appear in similar context will have similar embeddings.

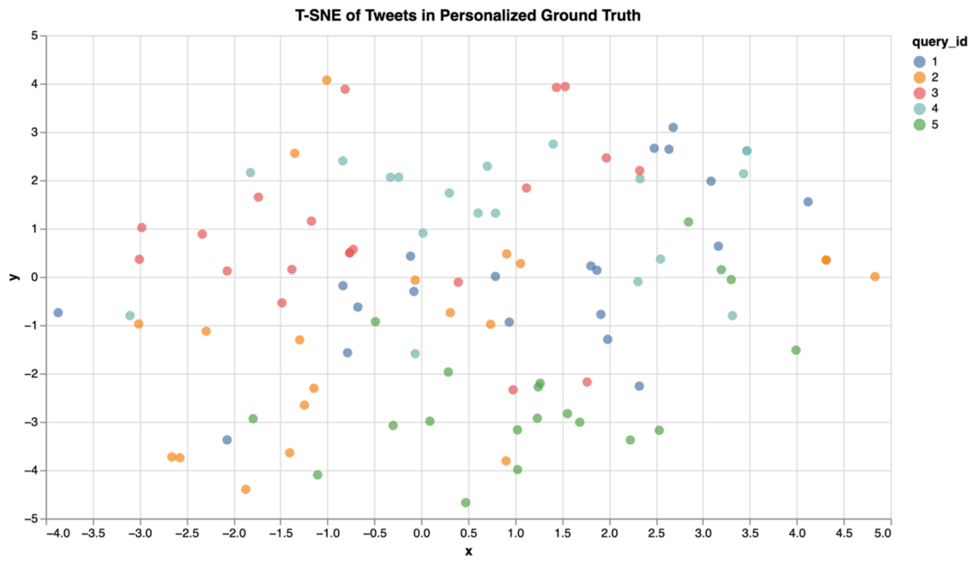


The image above is the plot of all English tweets using T-SNE, and as we can observe there is a dense cluster of points and some outliers. The fact that there is a pretty dense cluster could indicate that most of the English tweets share common topics or language patterns or that the collection is so big that it encovers mostly all topics. The clusters that are more isolated in the borders could represent some unique tweets that are significantly different from the majority of English tweets, as they could be tweets that have specific topics or personal opinions.

A more detailed review can be done if we reduce the number of tweets being analyzed.



The above plot shows the T-SNE of tweets in queries 1 and 2. Keep in mind that colors only represent queries, but inside each color there are relevant and nonrelevant queries. It can be observed that 2 distinct clusters can overlap, probably indicating that the tweets in each query share some similarities but they are different. The points that are far away from the cerntral mass can be regarded as those documents that are not relevent to the query.



The last plot is the T-SNE of the tweets in the personalized ground truth and we can observe that there are several distinct clusters with slime overlapping between them. Again, remember that the colors contain both documents relevant to the query and not. The clear separation between the clusters may indicate that the tweets in those clusters are discussing different topics, but them overlapping suggests that there are some common topics that the tweets are discussing.