IR Project Report

Nour Osama 202201874 section: 01

Layla Zayed 202201906 section: 02

Introduction

This report presents the implementation of an Information Retrieval (IR) system designed to process and index Arabic tweets. The project focuses on preprocessing a dataset of 5,000 Arabic tweets to enable efficient retrieval, leveraging text preprocessing techniques tailored for the Arabic language. The primary objectives are to clean and normalize the tweet data, remove irrelevant content, and prepare the collection for indexing and querying using the PyTerrier framework.

1.1) Data and Preprocessing

One of the main challenges we faced during preprocessing was that the dataset was in Arabic, which required language-specific preprocessing steps. This included installing the **arabic-stopwords** library and applying custom normalization techniques related to Arabic characters Additionally, the dataset contained emojis, so we implemented a function to remove them. We also used an Arabic stemmer to reduce tokens to their root forms.

<u>The following steps are applied:</u>

- Stop Words Removal: Using the arabicstopwords library, common Arabic stop words

  The function **remove_stop_words** filters out stop words from the tokenized text.

- Special Character Removal: The remove_special_char function employs regular expressions to remove URLs, mentions (e.g., @username), hashtags, punctuation, and excessive whitespace, ensuring only relevant text remains.
- Emoji Removal: The emoji library is used to strip emojis from the text via the **remove_emojis** function.
- Normalization: The normalize function standardizes Arabic characters by replacing variations of letters (e.g. ه ,to, ة) and normalizing certain forms (e.g. ئ ,to, ى).
- Stemming: The snowballstemmer library is used to apply Arabic stemming, reducing
  words to their root forms

The preprocess function integrates these steps, applying them sequentially to each tweet. The
preprocessed tweets are stored in a new column (processed_tweets) in the dataset.

1.2) Indexing

The preprocessed tweets are indexed using PyTerrier's IterDictIndexer to create an inverted index stored in the directory ./arabic_index. The indexer is configured with the following parameters:

- Path: The index is saved to ./arabic_index, with overwrite=True to replace any existing index.
- Metadata: The index stores document IDs (docno, up to 20 characters) and preprocessed tweet text (text, up to 2048 characters).
- Stemming and Stop Words: Stemming and stop word removal are disabled (stemmer=None, stopwords=None) since these were handled during preprocessing.
- Tokenizer: The UTFTokeniser is used to handle Arabic text, ensuring proper tokenization of Unicode characters.

The dataset is converted into a list of dictionaries, where each dictionary contains a docno (tweet ID) and text (preprocessed tweet).

2) Query Processing

To enable retrieval of relevant tweets from the indexed collection, a TF-IDF-based retrieval model is implemented using PyTerrier's BatchRetrieve class.

The retrieval process is configured as follows:

- Retrieval Model: The BatchRetrieve object is initialized with the indexed collection and configured to use the TF-IDF weighting model (wmodel="TF_IDF"). This model ranks documents based on term frequency and inverse document frequency, prioritizing tweets with higher relevance to the query terms.
- Result Limit: The retrieval is set to return the top 10 results (num_results=10) to focus on the most relevant tweets.
- Query Preprocessing: The input query is processed using the same preprocess function applied to the tweet dataset.
- Search Execution: The preprocessed query is passed to the search method of the BatchRetrieve object, which retrieves and ranks the top 10 tweets based on their TFIDF scores. The results are stored in a DataFrame (tfidf_results) containing columns such as document IDs, scores, and ranks.

3) Query Expansion

To retrieve relevant tweets from the indexed collection, multiple retrieval and query expansion techniques are implemented using PyTerrier and Sentence-BERT.

The process involves BM25 retrieval, RM3 query expansion, and BERT-based term expansion, applied to a sample query

- BM25 Retrieval: A BatchRetrieve object is configured with the BM25 weighting model to rank documents based on term frequency and inverse document frequency. The retrieval returns the top 30 results.
- RM3 Query Expansion: The RM3 pseudo-relevance feedback model is used to expand the query, implemented via PyTerrier's rewrite. RM3 with 10 expansion documents and 4 expansion terms (fb_terms=4). The BM25 results are piped into the RM3 expander (bm25 >> rm3_expander), generating an expanded query. The expanded query terms and their scores are extracted and formatted for further retrieval. A new BM25 search is performed using the formatted expanded query, yielding results. A comparison of document IDs and scores before and after expansion is displayed for the top 5 results.
- BERT-based Term Expansion: The CAMeL-Lab/bert-base-arabic-camelbert-mix-sentiment Sentence-BERT model is used to extract semantically relevant terms for query expansion. A custom function, **extract_top_terms**, tokenizes the top 10 retrieved documents, computes cosine similarity between the query embedding and term embeddings, and selects the top 5 terms. Two expansion strategies are applied:
  - BERT After RM3 + BM25: The top 10 documents from the RM3-expanded BM25 results are used to extract terms, which are appended to the preprocessed query to form bert_expanded_query_1. A BM25 search is performed, yielding bert_results_1.
  - BERT Only Expansion: The top 10 documents from the initial BM25 results are used to extract terms, forming bert_expanded_query_2. A BM25 search yields bert_results_2. The top 5 results from the original BM25, BERT-only, and BERT-after-RM3 searches are compared, showing document IDs and scores.

- Result Analysis with Original Tweets: A custom function, **print_top_docs_with_original_ids**, retrieves the original tweet text for the top 5 documents from each retrieval method (original BM25, BERT-only, BERT-after-RM3). It maps internal document IDs to docno values using the index's meta-index, then extracts the corresponding tweetText from the dataset. The function prints the docno and tweet text for each result, providing insight into the relevance of retrieved tweets. The original query is also displayed for context.

4) Results

The results from three retrieval methods— BM25, RM3-expanded BM25, and BERT-expanded BM25—are displayed and analyzed for a user-provided query (user_query). A custom function, **display_results**, is implemented to present the results in a human-readable format, showing the document ID (docno) and the original tweet text for each retrieved document.