Tweelink : Linking Twitter Hashtags to News Articles

Amisha Aggarwal
amisha19016@iiitd.ac.in

Aryan Behal
aryan19026@iiitd.ac.in

Harman Singh
harman19042@iiitd.ac.in

Mayank Gupta
mayank19059@iiitd.ac.in

Yash Bhargava
yash19289@iiitd.ac.in

Yash Tanwar
yash19130@iiitd.ac.in

1. ABSTRACT

Trends in Twitter are transient and hard to understand due to informality and casualness of the language. This paper tries to capture the context behind a trending hashtag and link it to relevant news articles. Twitter and articles datasets are built from scratch spanning 2 weeks. Keywords from tweets and articles are studied, their relationships examined to retrieve articles in ranked fashion. The rankings are evaluated on metrics like mAP (Mean Average Precision), mAR (Mean Average Recall) and nDCG (Normalized Discounted Cumulative Gain). Baseline models are reported followed by final models implemented as per proposed method. Finally best model is examined in conclusion and future work is proposed.

2. INTRODUCTION

2.1 Motivation and Problem Statement

Twitter is an extremely popular microblogging and social networking site launched in 2006. Today it boasts of hundreds of millions of monthly active users. Users can post, like, tweet and retweet 280 characters at a time. No other social platform comes close to capturing the pulse of the world in real time. Twitter users comprise people from all walks of life. Equally rich is the diversity of topics tweeted by users every second of the day. While a tweet is representative of a person's thought process, a hashtag is the culmination of collective thought of the populace. While the tweets shape the trend of a hashtag, the hashtag shapes the conversation as well, making a self feeding loop.

For better or for worse, Twitter plays a key role in shaping the conversation on topics that affect everyone's lives. From coronavirus to anti-vaccination protests to the Russia-Ukraine crisis, Twitter makes its presence felt everywhere. **Our project aims to capture the essence and context of a hashtag and link it to relevant news articles.** The context behind a hashtag is clouded by its informal structure. A hashtag is generally an amalgamation or portmanteau of several words making it difficult to decipher. This will help the uninformed wade through countless tweets to understand the context of the trend while combating misinformation at the same time.

2.2 Novelty

The novelty of our project lies in its uniqueness and ability to retrieve information out of a hashtag. To the best of our knowledge, there isn't any existing research or technology attempting to link hashtags to news articles. Some existing papers have tried to use the tweets containing URLs, but we are not dependent only on URLs within the tweet. Our novelty lies in the fact that we consider external news articles explaining the trending news/controversy. We face the challenge of working with a corpus of extremely short texts on which regular NLP techniques may yield poor results. The text itself may be informal, multi-lingual and full of grammatical errors.

2.3 Dataset Description

We built a dataset of tweets and news articles. We used Twitter API v2 to track a hashtag across the most relevant tweets every hour for 72 hours. Around 20 hashtags were collected everyday from Feb 14, 2022 to Feb 28, 2022. News articles were collected via the news section of Google. For a particular hashtag, after understanding its context through tweets, we collected around 10 news articles within the time frame of its occurrence which consist of both relevant and less-relevant content.We didn't employ Web scraping techniques using BeautifulSoup and other libraries since a particular news agency might not cover the content related to all the major trends/hashtags we have collected the tweets for. Thus, we have not limited our news articles search space to a particular website, in order to develop a vast and comprehensive dataset.

Our dataset consists of 224213 tweets and 2820 articles. The dataframe for tweets has the following fields - author id, date, geolocation, language, tweet text, hashtags, urls, sensitive tweet or not. The dataframe for articles has the following fields - link of article, corresponding hashtag, geolocation, article heading, article text.

2.3.1 Exploratory Data Analysis

Figure 1 shows most tweets use lesser number of hashtags. Figure 2 shows that hashtags like Modi, RussiaUkraineWar trend for a long time, and hence have been used in a large number of tweets. Figure 3 shows that only 3% of the tweets extracted from the Twitter API have geolocation field in them.

2.4 Proposed Solution

We ask the user to input a hashtag, time and location. Based on the input, we retrieve relevant tweets from our dataset and create a feature vector which is used to retrieve relevant articles in a ranked order. The workflow of our system can be seen in Figure 1. The

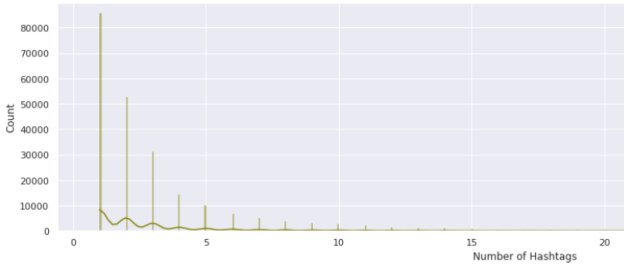

Figure 1: Number of hashtags per tweet

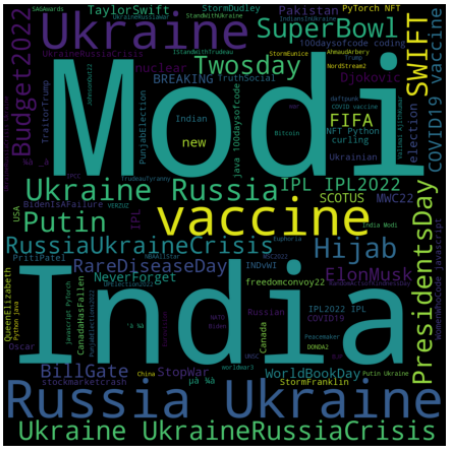

Figure 2: Word Cloud of hashtags

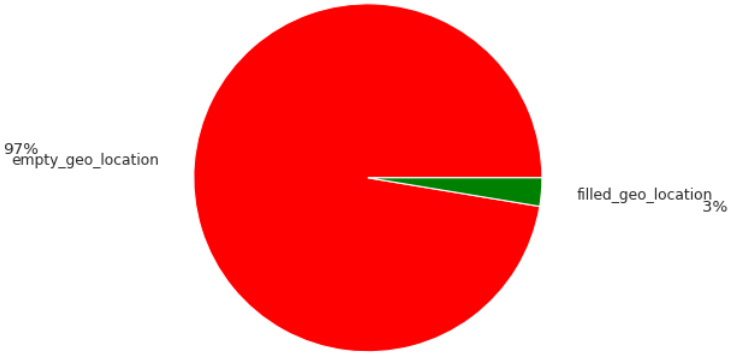

Figure 3: Twitter API geolocation data

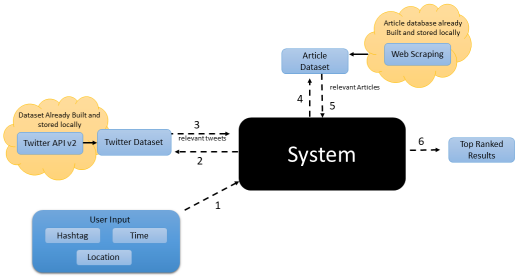

Figure 4: Workflow of our system

ranking is determined as a function of the hashtag, text in relevant tweets, location of tweets, location of articles and difference in time input by the user, time of tweets and date of publication of articles. Time is an important factor in the retrieval of relevant results, as illustrated in Figure 2. We have provided a web interface to facilitate the process.

3. LITERATURE REVIEW

3.1 Keyword Extraction and Semantic Similarity

Bordoloi et al. [1] proposed a modified graph based approach KWC (Keywords from Collective Weights) for automatic keyword extraction from a twitter corpus using frequency, centrality, position, strength of the neighbours and other influencing graph measures. Their method was based on node-edge rank centrality with node weight depending on various features. The control flow of their approach followed data preprocessing, textual graph representation, collective node-weight assignment, and keyword extraction, resulting in top keywords as the output. For the textual graph representation part, they represented each token as a vertex and if two tokens co-occurred within the same window (i.e., a tweet), then an edge was created between them. The generated graph revolved around a single topic and thus held many common tokens that were associated with more than one tweet. The edges were weighted according to co-occurrence frequency. The nodes were weighted based on important graph measures like term frequency, selectivity centrality, etc., which made their algorithm unique and more powerful. Finally, keyword extraction was performed using NE-Rank and degree centrality and best n keywords were given as output. Their novel approach outperformed several existing graph based methods. Since our project revolves around similar lines of extracting keywords for a given set of tweets for a hashtag, their superior method might come handy in our methodology.

Mihalcea et al. [2] suggested a method for measuring the semantic similarity of texts by using the information that can be drawn from the similarity of the words. Their main focus was to find the semantic similarity instead of lexical similarity. In addition to the similarity of words, they also took into account the specificity of words, which was determined using the inverse document frequency (idf). Starting with each of the two text segments, and for each word, they determined the most similar word in the other text segment, according to the PMI-IR similarity measure. Next, they combined the word similarities and their corresponding specificity, and determined the semantic similarity of the two texts. They achieved an accuracy of 70.3%, representing a significant 13.8% error rate reduction with respect to the traditional vector-based cosine similarity baseline. Our project also aims at finding the semantic similarity between the tweets and the articles. This technique of measuring the similarity using the

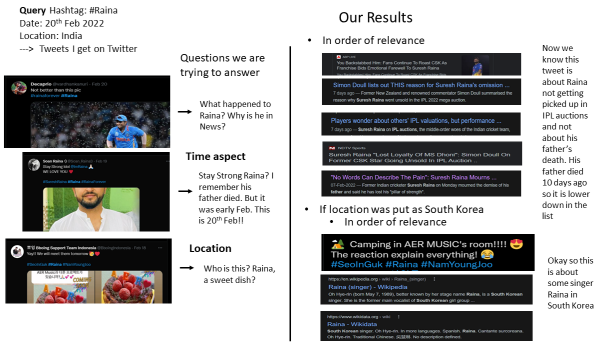

Figure 5: An example scenario of our problem statement

PMI-IR method can be used in the future to extend our work.

Prakoso et al.[3] conducted a systematic literature review on the various short text similarity(STS) measurements available. They grouped the different STS methods into 4 different categories depending on the attributes, procedures and resources that the techniques used: string-based, knowledge-based, corpus-based and hybrid-based similarity. The string-based methods reviews compared word and character sequences. The knowledge-based methods used word-level affinity by making use of the semantic information present and scale it up to sentences. The corpus-based methods used techniques such as Latent semantic analysis(LSA), Latent Dirichlet Allocation(LDA) and word2vec to find the relations between words in a corpus, which were then utilized at the sentence level to measure similarity. Hybrid-based methods were a combination of several different methods from the above categories and were employed to improve the accuracy. The authors analyzed the benefits and limitations of these similarity measures on 7 different features, namely (i) domain independence, (ii) language independence, (iii) semantic knowledge requirement, (iv) corpus and training data, (v) capability to identify semantic meaning, (vi) word order similarity and (vii) polysemy. For the string-based STS methods, they concluded that although it's relatively easier to implement and is advantageous in cases where we need the similarity between sequences of words, sentences can be structured in different ways to convey the same meaning and string-based methods fail to justify their similarity. For the knowledge-based category, the semantic meaning of a sentence was captured and then used to determine the similarity. The authors concluded that knowledge-based methods in conjunction with semantic information on a specific domain is more helpful in a scenario where we have a specialized use case. For the corpus-based category, the authors concluded that conventional techniques such as LSA and LDA, although very reliable, are computationally expensive and the bag-of-words model employed by these methods disables word-similarity measurement. Lastly, hybrid-based methods were concluded to increase the accuracy as well as the processing time required for measuring the similarity as a combination of semantic and corpus based methods was used.

Pradhan et al.[4] also take on the problem of obtaining the most relevant documents from a web search by analyzing the different similarity methods used for document retrieval. They've categorized the similarity measures into 2 categories: lexical similarity and semantic similarity. Lexical Similarity assigns similarity based on character and sentence matching[19]. It works on different sets of strings and assigns a score depending on the degree of overlap between them. The authors analysed 4 character based similarity measures namely: Longest Common Subsequence, N-Gram similarity, Levenshtein distance Similarity and Jaro Distance, which were followed up with 5 statement based similarity measures namely: Cosine similarity, Centroid based similarity, Web Jaccard Similarity, Web Simpson Similarity and PMI similarity. Semantic similarity on the other hand distinguishes itself from lexical similarity by calculating the similarity between texts based on their meaning rather than sequence of characters. The authors subdivided the methods under semantic similarity into 2 categories, Corpus-based and knowledge-based. Corpus-based similarity measures determine the similarity between words that have the same meaning based on the information derived from a corpus and included Normalized Google Distance(NGD), Normalized Information Distance(NID), Normalized Compression Distance(NCD) and Latent Semantic Similarity (LSA), which is the most popular vectorial semantics method. Knowledge-based similarity measures on the other hand use semantic networks such as "WordNet" and Natural Language Tool Kit (NLTK) to measure the contextual similarity between words. These measures include Resnik Similarity and Vector similarity. Lastly, a hybrid method that used kernel based similarity and cosine similarity was discussed and it was concluded that the kernel based similarity measure gives a better accuracy compared to cosine similarity.

Reading and summing up large texts is very difficult as the amount of information generated keeps on increasing. Keyword extraction systems are required to automate the process of keyword generation owing to the lack of descriptive terms present in large volumes of text. Campos et al. [5] resort to an unsupervised automated keyword extraction method YAKE! that is independent of the dataset and it's description. They make use of statistical information and the local specific text features such as term frequencies to identify the important keywords, without the need of a corpus. They adopt the following two steps to extract the keywords: computing the importance of the terms individually using statistical features and applying a n-gram model that forms multi-word terms and evaluating them heuristically. The proposed algorithm comprised of 5 main steps: pre-processing the text and identifying the candidate terms, representing the input by a set of statistical features, calculating term scores by evaluating them heuristically, n-gram generation and candidate keyword score computation based on their importance, and data de-duplication and ranking. The algorithm first divides the text into sentences, followed by splitting, annotation, tokenization and stopwords identification. They then statistically analyze the sentence structure, term frequencies and co-occurrence for the purpose of feature extraction. They devise a set of 5 features that depict the nature of a candidate term: Casing of the terms, Term position, Term frequency Normalization, Term context relatedness and Number of appearances of the same term in different sentences. The term score is computed according the formula: S(t) = (TRel * TPosition)/(TCase + TFNorm/TRel + TSentence/TRel). After n-gram generation, the candidate keyword scores are computed using $S(kw) = \frac{\Pi_{t \in kw} S(t)}{KF(kw) * (1 + \Sigma_{t \in kw} S(t))}$ where kw represents the candidate keyword terms and S(kw) represents the final score. Due to high chances of similarity among candidate keywords, the next step is to perform data de-duplication where potentially similar candidate keywords are eliminated. The authors evaluate the effectiveness of their system on 20 different datasets having a diversity of language, domains, text size and types of documents. By observing the results obtained, YAKE! outperformed

not only the statistical methods such as TF-IDF, KP-Miner and Rake, but also the graph-based methods such as TextRank, TopicRank, PositionRank across multiple datasets independent of document languages or a trained model. The keyword extraction mechanism followed by YAKE! can help us in devising a model to extract the important keywords from the tweets and articles present in our dataset.

3.2 Similar Work

Uta Losch et al.[6] tried to map micro-blogs to Encyclopedia articles. The idea of their method was to input a search query (a hashtag) and output an RDF document containing the most recent messages that matched the query, the authors of these messages and the most relevant Wikipedia entities for this result. They extracted the 100 most recently published tweets using Twitter Search API along with author data, geo-location, publishing date and the URLs posted in the messages. All text data of the tweets and articles hyperlinked by the URLs was put in Wikifier for obtaining the content annotations. This tool took a text as input to return a set of DBpedia entities which were relevant for the analysed text in RDF format. Our project also runs along similar lines except for the fact that we are suggesting articles and also ranking them based on relevance. For making our tweet database from hashtags, we have followed a similar approach as they have used by using Twitter Search API and combining text to search for relevant articles. In contrast to their work, we have used our own methods rather than wikifier api for searching relevant articles.

Krestel et al.[7] trained various models to suggest relevant tweets for articles and evaluated their results on the basis of user study. To identify similar content tweets they employed language models (to find word overlap) and topic models (to find concept overlap). To combine the resulting content similarity scores with other features, such as recency or popularity of a tweet, they used logistic regression as well as boosting. For language modeling, they used a document likelihood model to compute the probability that a tweet is generated from a news article instead of a query likelihood model (which assumes that queries are short and documents are long). They used Dirichlet smoothing to smoothen their language model and geometric mean for normalizing tweet lengths. For relevance, they used the document likelihood and latent Dirichlet allocation to find the most relevant tweets. They used logistic regression apart from above techniques to mark a tweet as relevant or irrelevant which was trained, considering 16 additional features, such as publication time, length, follower count, etc and Adaboost and decision stumps to identify less similar tweets. To ensure unique results in the final ranking, they ignored tweets that have a high word overlap with tweets that were already in the set of recommended tweets. In their set-up, the topic model outperformed language model. Adding more features increased the accuracy of the model. In our model, we employ similar techniques of language model and topic model to check for word and concept similarity. For normalizing tweets lengths, we have used the mean.

Ahuja et al.[8] have made a framework which downloads a list of news-related relevant tweets from twitter, extracts URLs associated with those tweets and infers the significance of those URLs in Twitter. They collected tweets using the REST and Streaming API of Twitter, and then extracted URLs from these tweets. Further, the tweets have been ranked based on relevancy. They used a threshold of 15% similarity between headline and the tweet, thus finding news-related relevant tweets and the corresponding news topics. We can use this work for extracting URLs from tweets and then using the corresponding news articles. But we are not dependent only on the URLs. We are also using the tweets that don't have URLs, and are mapping them to relevant news articles from the articles database.

3.3 Ranking Evaluation

In an ideal scenario, our system would link the hashtag directly to the most relevant article present on that particular topic. However, there could be multiple articles that are closely related with only slight variations amongst them. Corso et al.[9] proposed a ranking algorithm that takes into account a number of desirable properties of each piece of news and subsequently assigns them a rank based on these properties. They evaluated the consistency of their algorithm based on its behaviour for two important limiting cases, the mean rank of independent news articles should be independent of the time and size of observation window and two sources, where one acts as a "mirror" source, should have a similar rank. The authors introduced two classes of algorithms: Non-Time-aware ranking algorithms and Time-Aware ranking algorithms. Non-Time-aware ranking algorithms dealt with a static dataset of news articles rather than a continuous stream of news flow. Both algorithms under this class, however, violated the limiting cases and coupled with the loss of temporal information associated with the fixed time-window scheme, Time-aware ranking algorithms were made necessary. According to Corso et. al.[9],the importance of a piece of news and the time of its emission were strictly related. To account for this, the author introduced a "freshness" decay parameter "α" that is obtained from the half-life decay time. Moreover, these algorithms took into account another parameter "β" which could be varied to tune the change in the rank of a news source based on the arrival of a fresh piece of news. Although an improvement from the Non-Time-aware algorithms, each of the proposed Time-aware algorithms also failed to satisfy one desirable property and failed to pass the second limiting case. However, for the purpose of ranking articles, Time-aware algorithms sufficed.

Yilmaz et. al.[10] studied the shortcomings of traditional rank correlation coefficients such as Kendall's τ and Spearman rank correlation coefficient for two ranked lists and proposed a new coefficient that penalized errors at high rankings more than the errors at low rankings and had an underlying probabilistic interpretation that was easy to understand. The AP rank correlation coefficient introduced by the authors possessed two important properties when compared to Kendall's : its equality to Kendall's τ when the errors are uniformly distributed over the entire ranking and the difference in value when the error in ranking is concentrated more at the top/bottom of the list.

4. METHODOLOGY

We take the hashtag, date and location as input from the user. After applying our models in the backend, we give as output the top k relevant news articles corresponding to that hashtag, date, location and URL. We have created and stored preprocessed database of tweets and articles. The workflow of our models goes in this order:

1. Pre-processed tweets corresponding to the hashtag, date (including the previous and the next date), and location are extracted from our tweets dataset.

2. The tokens corresponding to all the tweets are combined to make the query vector.

3. Pre-processed articles corresponding to that date, the previous date and the next date are extracted from the articles database.

4. Different similarity metrics are applied between the given query and the news articles. The ranking is then evaluated using metrics like mean average precision, mean average recall, nDCG and the results (top k articles) corresponding to the best evaluation score are reported to the user.

4.1 Preprocessing

For better results, we made adjustments in baseline pre-processing steps. In our earlier pre-processing, we encountered the existence of URLs, Non-UTF, references to user accounts using @ symbol etc. We updated our pre-processing step for both articles and tweets to get rid of these words. We are performing the following steps in pre-processing:

1. Converted the text to lower case
2. Removed @ words (remove references to users)
3. Removed Hashtags from tokens
4. Removed URLs from text
5. Removed Stop-words from tokens
6. Removed Punctuation from tokens
7. Removed Non-UTF words
8. Removed blank spaces from tokens
9. Performed lemmatization

We have employed several keyword extraction techniques on our ranking models to improve our baseline results by improving the quality of query we make. We have also added implementations of state of the art model Okapi BM25 and Soft cosine for ranking articles apart from the baseline models. We provide a Web-based user interface for querying into our system.

4.2 Keyword Extraction Models

The following keyword extraction models have been used:

1. **Keyword Extraction using Collective Node Weight (KECNW)**: Biswas et al.[11] devised a knowledge graph-based keyword extraction model which we have implemented for keyword extraction. It consists of four phases: pre-processing, textual graph representation, node weight assignment and keyword extraction. Preprocessing involves additional steps of removing URLs, hashtags and words with @ symbol which are assumed to be noise. Unimportant tokens which occur less than average occurrence frequency are removed. Textual graph representation involves vertex assignment and establishment of edges between vertices. Node weight assignment phase determines node weight based on its frequency, centrality, position and strength of neighbors. Finally, keywords are extracted using NE rank centrality with degree as tie breaker.

2. **Yet Another Keyword Extractor (Yake)**: It is a light-weight unsupervised automatic keyword extraction method based on text statistical features. It is text corpus and language independent. It finds the keywords in a single document. LIAAD github[12] code implementation has been used for Yake.

3. **Rapid Automatic Keyword Extraction (Rake)**: It is a domain independent algorithm based on the observation that natural language contains a lot of stopwords and punctuations which offer very little to explain the meaning and context of text. It removes these extra words to extract the candidate keywords which are assigned a score based on degree and frequency in a co-occurrence graph. Top k-ranked keywords can be used to make enhanced queries. Vgrabovets github[13] code implementation has been used for Rake.

4. **Text Rank**: It is a graph based ranking algorithm like Google Page-Rank algorithm which measures the relationship between two or more words to find keyphrases. In this technique, individual sentences are found from the document which are vectorized. Similarities between these sentence vectors are calculated and stored in a matrix. This matrix is then converted to a graph with sentences as vertices and similarity scores as edge to calculate final sentence rank in a iterative manner. Summanlp github[14] code implementation has been used for Text Rank.

5. **Keybert**: Rake and Yake typically work based on the statistical properties and not so much on semantic similarity of a text. To improvise, we brought in BERT. BERT is a bi-directional transformer model that allows us to transform phrases and documents to vectors that capture their meaning. KeyBERT is a simple and easy-to-use keyword extraction technique which finds the sub-documents in a document that are the most similar to the document itself using BERT-embeddings and basic cosine similarity. BERT is used to extract document embeddings in order to obtain a document-level representation. Then, for N-gram words/phrases, word embeddings are extracted. Finally, we employ cosine similarity to identify the words/phrases that are most similar to those in the manuscript. The words that are the most similar could then be identified as the terms that best characterise the entire document. MaartenGr github[15] code implementation has been used for KeyBERT.

4.3 Evaluation Metrics/Models

The following evaluation metrics/models have been used:

1. **Jaccard Coefficient**: Jaccard Coefficient measures the similarity between finite sample sets. It is computed as the size of the intersection of the sets divided by their union.

$$\text{Jaccard(A,B)} = \frac{|A \cap B|}{|A \cup B|}$$

We iterate over our set of preprocessed documents one at a time and calculate the Jaccard similarity between the query and the document using the above formula. We sort the relevant documents in descending order and return a list of the most relevant documents along with their Jaccard similarity scores.

2. **TF-IDF similarity matching**: TF-IDF is another statistic that quantifies the importance of a term in a document or a corpus. It increases proportionally to the frequency of the term in the document. It involves computing the Term Frequency which is the number of times each word appears in each document as well as the Inverse Document Frequency which is computed for a term as the inverse function of the number of documents it appears in. TF-IDF is the product of the two statistics.

$$\text{tf-idf}_{t,d} = (1 + \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$$

It acts as a measure of how much information a word provides i.e whether it is common or rare in the corpus.For our system, all the articles are read and preprocessed. User query is taken and preprocessed in a similar way. Vocabulary of the whole dataset is found

and 2 dictionaries: word2id and id2word are created to keep note of index of each word in the vocabulary. For creating TF-IDF matrix, we need both TF and IDF For TF, a word-document matrix to store the frequencies of each word in each doc. 5 different TF-IDF matrices are then formed using the 5 different weighing schemes:
(1) **Binary weighted** where term frequency is simply 1 or 0 which denotes whether the term is present in the document or not.
(2) **Raw Count** uses the number of times the term occurs in a document
(3) **Term Frequency** uses the raw count of a term normalised by the sum of the raw count of all terms in that document.
(4) **Log Normalization** uses simply the logarithm of (1+raw count of term) and
(5) **Double Normalization** uses formula 0.5+0.5(raw count of given term/ max raw count in the doc).
The query is pre-processed in similar way as the dataset and a query vector is created of size as big as vocabulary size. Each index of this query vector corresponds to a word in dataset vocabulary. The query vector is updated with the term-frequencies of each query token at the indices which correspond to their index in vocabulary. The query vector is also weighted according to weighing scores of TF-IDF schemes and multiplied with respective IDF scores obtained from the dataset. Dot Product of each TF-IDF matrix is taken with the processed query vector to obtain the TF-IDF scores of each document for each query token.

3. **Cosine similarity**: In NLP, cosine similarity is a standard tool to measure the text-similarity between two documents. The sizes of the 2 documents can be unequal and of any length. The documents are represented in n-dimensional vector space. The n-D vectors are projected to a multidimensional space and the cosine of angle between them is measured. The output will vary between 0 and 1, values closer to 1 representing high matching between the documents and closer to 0 representing poor matching. The word occurrence for both the documents is calculated using Count Vectorizer or TF-IDF Vectorizer. The vocabulary considered is the union of the terms in the query and the document being compared. Dot product is taken between the vectors consisting of binary values and the similarity score is reported. Top k documents with the highest similarity score are reported. Two metrics were used by us: **TF-IDF** and **Count Vectorizer**. For these metrics, each query and document is combined to form a list and converted to vectors to find similarity scores.

4. **Binary Independence Model**: It is a probabilistic model that assumes that either term is present in document or not. A 2-D matrix is created which stores the occurrence of each word in each document. 1 mean presence and 0 is for absence and all terms are considered independent. A probability is assigned to a relevance of document depending on probability of relevance of terms vector of that document. A rank is assigned to the query which is recomputed based on weights for 10 iterations using probabilistic methods giving a final rank. Top k documents are shown.Laurabalasso github[16] code implementation has been used for BIM model.

5. **Okapi BM25 (Okapi Best Matching 25)**: It tries the improve the traditional TF-IDF by casting relevance as a probabilistic model. It scores each document in a corpus according to the document's relevance to a particular text query. For a query Q, with terms q1, q2,.., qn, the BM25 score for document D is:

$$\text{BM25(D,Q)} = \sum_1^n IDF(qi,D)\frac{\text{f(qi,D)}\cdot(\text{k}_1+1)}{\text{f(qi)}+\text{k}_1\cdot(1 - \text{b} + \text{b}\cdot\frac{|\text{D}|}{\text{d}_{avg}})}$$

where:

- f(qi,D) : number of time qi occurs in document D. The higher the value of this factor, the higher the score of the document.
- $|D|$: number of words in document D. This factor penalizes the documents since it is believed higher document size, lesser should be its value.
- d_{avg} : average number of words per document
- b, k1 : hyper-parameters for BM25. If b is bigger, the effects of document length compared to average length are more amplified. If k1 has higher value, the slower the saturation rate is.

For Okapi BM25, a list of corpus is created for each article text and is passed for score evaluation with each query and results are stored in a list. Brown github[17] code implementation has been used for Okapi BM25.

6. **Soft Cosine**: It is a method that measures the similarity between two documents in a meaningful way, even when the words in them are not the same. It uses a measure of similarity between words, which can be derived using word2vec [vector embeddings of words]. It works better than many techniques in semantic text similarity tasks. For soft cosine, 'hi, world' and 'hello, world' are treated to be similar since 'hi' and 'hello' have similar context. For soft cosine, documents are used to make a corpus. Each query and document is combined to form a dictionary and converted to vectors to find similarity score. RaRe-Technologies github[18] code implementation has been used for Soft Cosine.

We have run our code for each combination evaluation metric/model and keyword extraction technique for a single query **'hijab, 2022-02-19, india'**. We have also run our code for all queries. Rankings are evaluated using **mean average precision**, **mean average recall** for baseline results and **nDCG score** is added for final results.

5. RESULTS

5.1 Baseline Results

5.1.1 Evaluation Metrics/Models

1. **Jaccard Coefficient**: Code run for a single hashtag: **hijab, 2022-02-19, India** and all hashtags combined. Jaccard similarity failed to capture term frequency and ordering of terms into account. This is evident from the table below.

Table 1: Jaccard Coefficient

Query	Mean Avg Precision	Mean Avg Recall
hijab, 2022-02-19, India	0.0387	0.0325
For All Queries	0.2615	0.1709

2. **TF-IDF similarity matching**: Code run for a single hashtag: **hijab, 2022-02-19, India** and all hashtags combined. Top 10 keywords are used to improve results.

Normal query: hijab, 2022-02-19, India

Table 2: Binary Tf-Idf scores.

Query	Mean Avg Precision	Mean Avg Recall
Normal query	0.3920	0.3749
Using Top 10 keywords	0.501	0.439
For all queries	0.31	0.29
Using Top 10 keywords	0.44	0.35

Table 3: Raw Count TF-IDF scores

Query	Mean Avg Precision	Mean Avg Recall
Normal query	0.8343	0.775
Using Top 10 keywords	0.834	0.775
For all queries	0.75	0.68
Using Top 10 keywords	0.75	0.68

Table 4: Log Normalisation scores

Query	Mean Avg Precision	Mean Avg Recall
Normal query	0.8343	0.775
Using Top 10 keywords	0.834	0.775
For all queries	0.75	0.68
Using Top 10 keywords	0.75	0.68

Table 5: Term Frequency scores

Query	Mean Avg Precision	Mean Avg Recall
Normal query	0.7652	0.71
Using Top 10 keywords	0.823	0.765
For all queries	0.64	0.61
Using Top 10 keywords	0.70	0.625

Table 6: Double Normalization scores

Query	Mean Avg Precision	Mean Avg Recall
Normal query	0.0502	0.065
Using Top 10 keywords	0.8248	0.765
For all queries	0.1	0.11
Using Top 10 keywords	0.75	0.68

3. **Cosine similarity**: Code run for a single hashtag: **hijab, 2022-02-19, India** and all hashtags combined. Two metrics were used by us: **TF-IDF** and **Count Vectorizer**. TF-IDF metric is better than Count Vectorizer metric as it penalizes the most occuring words in a document thus giving less importance to them.

Normal query: hijab, 2022-02-19, India

Table 7: Cosine (Count Vectorizer)

Query	Mean Avg Precision	Mean Avg Recall
Normal query	0.834	0.3875
For all queries	0.764	0.296

Table 8: Cosine (TF-IDF)

Query	Mean Avg Precision	Mean Avg Recall
Normal query	0.834	0.3875
For all queries	0.759	0.295

4. **Binary Independence Model**: Code run for a single hashtag: **hijab, 2022-02-19, India** and all hashtags combined.

Normal query: hijab, 2022-02-19, India

Table 9: Binary Independence Model

Query	Mean Avg Precision	Mean Avg Recall
Normal query	0.0247	0.0199
Using Top 10 keywords	0.834	0.3875
For all Queries	0.05	0.03
Using Top 10 keywords	0.54	0.30

From the results, we can see that for 'hijab' query, Jaccard, Binary Independence Model and Double Normalization Tf-Idf work really poorly. Raw Count, Log Normalisation, Cosine similarity work really well. Across all baseline models it is evident that taking top k keywords yield way better results than simply considering all the keywords. Cosine [count verctorizer] has highest precision value[0.764] whereas Log normalization, Raw Count has highest recall value when we run code for all hashtags. Values for precision and recall drop when we run for all queries than just for 'hijab' query.

5.2 Final Results

For final results, we have run our code for each combination of keyword extraction technique and evaluation metric/model. We have used mean average precision, mean average recall and nDCG for evaluating the rankings obtained.

We have run the code first of query: 'hijab, 2022-02-19, India' and all hashtags to gauge improvement from baseline results to final results.

Table 10: Mean Average Precision:Query: hijab, 2022-02-19, india

Keyword Extraction Methods							
Model / Similarity Metric		Plain	KECNW	YAKE	RAKE	TextRank	KeyBert
	Jaccard	0.054	0.994	0.805	0.644	0.870	1.0
	TF-IDF Binary	0.463	0.994	0.934	0.459	0.870	1.0
	TF-IDF Raw count	1.0	0.942	0.997	0.353	0.992	1
	TF-IDF Log normalization	0.997	0.997	0.942	0.446	0.986	1.0
	TF-IDF Term frequency	1.0	1.0	1.0	0.737	0.956	1.0
	TF-IDF Double normalization	0.989	0.89	0.712	0.631	0.771	0.92
	Binary Independence	0.033	0.942	0.997	0.173	0.997	1.0
	Cosine (Count vectorizer)	1.0	0.994	0.994	0.562	0.992	1.0
	Cosine (TF-IDF vectorizer)	1.0	0.994	0.994	0.562	0.992	1.0
	Okapi BM25	1.0	0.997	1.0	0.782	0.986	1.0
	Soft Cosine	1.0	0.998	0.997	0.621	0.996	1.0

Table 11: Mean Average Recall: Query = hijab, 2022-02-19, india

Keyword Extraction Methods							
Model / Similarity Metric		Plain	KECNW	YAKE	RAKE	TextRank	KeyBert
	Jaccard	0.039	0.519	0.412	0.282	0.434	0.524
	TF-IDF Binary	0.210	0.519	0.487	0.24	0.437	0.524
	TF-IDF Raw count	0.524	0.489	0.522	0.194	0.517	0.524
	TF-IDF Log normalization	0.522	0.522	0.489	0.24	0.512	0.524
	TF-IDF Term-frequency	0.524	0.524	0.524	0.35	0.495	0.524
	TF-IDF Double-normalization	0.514	0.532	0.4	0.37	0.522	0.721
	Binary Independence	0.024	0.489	0.522	0.109	0.522	0.524
	Cosine (Count vectorizer)	0.524	0.519	0.519	0.242	0.517	0.524
	Cosine (TF-IDF vectorizer)	0.524	0.519	0.519	0.242	0.517	0.524
	Okapi BM25	0.524	0.522	0.524	0.374	0.512	0.524
	Soft Cosine	0.544	0.582	0.582	0.274	0.542	0.554

Table 12: nDCG score: Query = 'hijab', '2022-02-19', 'india'

Keyword Extraction Methods							
Model / Similarity Metric		Plain	KECNW	YAKE	RAKE	TextRank	KeyBert
	Jaccard	0.046	0.884	0.692	0.465	0.777	0.524
	TF-IDF Binary	0.353	0.918	0.898	0.460	0.813	1.0
	TF-IDF Raw count	0.900	0.900	0.886	0.322	0.876	0.900
	TF-IDF Log normalization	0.915	0.915	0.900	0.484	0.872	0.900
	TF-IDF Term-frequency	0.888	0.894	0.894	0.716	0.879	0.890
	TF-IDF Double-normalization	0.842	0.812	0.56	0.52	0.62	0.89
	Binary Independence	0.023	0.904	0.890	0.170	0.890	0.900
	Cosine (Count vectorizer)	0.892	0.877	0.878	0.500	0.872	0.888
	Cosine (TF-IDF vectorizer)	0.886	0.878	0.878	0.500	0.872	0.888
	Okapi BM25	0.932	0.920	0.707	0.932	0.876	0.890
	Soft Cosine	0.902	0.912	0.894	0.752	0.986	0.907

Table 13: Mean Average Precision: All Queries

Keyword Extraction Methods							
Model / Similarity Metric		Plain	KECNW	YAKE	RAKE	TextRank	KeyBert
	Jaccard	0.175	0.819	0.790	0.313	0.825	0.825
	TF-IDF Binary	0.405	0.807	0.782	0.393	1	1
	TF-IDF Raw count	0.982	0.922	0.930	0.387	0.967	0.992
	TF-IDF Log normalization	0.917	0.947	0.922	0.652	0.976	0.992
	TF-IDF Term frequency	0.959	0.983	0.977	0.921	0.978	0.992
	TF-IDF Double normalization	0.989	0.992	0.823	0.725	0.946	0.995
	Binary Independence	0.050	0.942	0.997	0.488	0.994	1.0
	Cosine (Count vectorizer)	0.812	0.879	0.309	0.929	0.885	0.885
	Cosine (TF-IDF vectorizer)	0.800	0.879	0.310	0.926	0.886	0.886
	Okapi BM25	0.794	0.877	0.90	0.336	0.867	0.867
	Soft Cosine	0.834	0.997	0.432	0.951	0.926	0.905

Table 14: Mean Average Recall: All Queries

Keyword Extraction Methods							
Model / Similarity Metric		Plain	KECNW	YAKE	RAKE	TextRank	KeyBert
	Jaccard	0.083	0.417	0.406	0.163	0.427	0.305
	TF-IDF Binary	0.192	0.498	0.442	0.192	0.395	0.485
	TF-IDF Raw count	0.421	0.445	0.478	0.098	0.456	0.489
	TF-IDF Log normalization	0.488	0.492	0.456	0.192	0.486	0.496
	TF-IDF Term-frequency	0.512	0.504	0.509	0.312	0.472	0.504
	TF-IDF Double-normalization	0.498	0.521	0.362	0.324	0.482	0.634
	Binary Independence	0.261	0.428	0.389	0.371	0.364	0.227
	Cosine (Count vectorizer)	0.423	0.449	0.140	0.479	0.453	0.351
	Cosine (TF-IDF vectorizer)	0.416	0.449	0.140	0.478	0.453	0.351
	Okapi BM25	0.413	0.459	0.469	0.170	0.453	0.340
	Soft Cosine	0.485	0.479	0.231	0.512	0.511	0.384

Table 15: nDCG score: All Queries

Keyword Extraction Methods							
Model / Similarity Metric		Plain	KECNW	YAKE	RAKE	TextRank	KeyBert
	Jaccard	0.005	0.098	0.076	0.064	0.086	0.099
	TF-IDF Binary	0.334	0.818	0.803	0.424	0.792	0.822
	TF-IDF Raw count	0.795	0.826	0.813	0.312	0.823	0.815
	TF-IDF Log normalization	0.741	0.782	0.775	0.425	0.714	0.755
	TF-IDF Term-frequency	0.802	0.815	0.815	0.654	0.803	0.822
	TF-IDF Double-normalization	0.763	0.731	0.508	0.452	0.555	0.82
	Binary Independence	0.631	0.812	0.712	0.732	0.739	0.836
	Cosine (Count vectorizer)	0.775	0.803	0.253	0.813	0.820	0.607
	Cosine (TF-IDF vectorizer)	0.764	0.803	0.254	0.822	0.820	0.607
	Okapi BM25	0.75	0.859	0.850	0.298	0.841	0.704
	Soft Cosine	0.792	0.848	0.810	0.303	0.846	0.759

5.3 Analysis Of Results

5.3.1 Analysis of Keyword Extraction Methods

1. Plain: All the keywords post pre-processing are considered. The most important keywords are lost among the noisy ones which reduce the performance across any model/similarity metric.

2. KECNW: The directed knowledge graph is able to understand the context of a topic by creating edges between important terms both within a tweet and beyond tweet level. Extraction of top keywords based on NE rank and degree centrality is able to capture elite keywords which capture the context of a topic brilliantly.

3. YAKE: YAKE retrieves top keywords based on a lightweight unsupervised model based on text statistical features. The nDCG values across different models are top notch, though it is narrowly beaten by KECNW. We theorize that this is because KECNW is trained on our corpus whereas YAKE is corpus agnostic.

4. RAKE: Keyword extraction by RAKE is dependent primarily on stop word and punctuation removal. The results therefore are unsatisfactory as compared to other techniques.

5. TextRank: TextRank extracts the most important sentences. We are considering the terms in the extracted sentences as most relevant. Results of TextRank fall behind KECNW and YAKE as the end goal is keyword extraction and not sentence extraction. There is a possibility of some top keywords being left out which were present in not so important sentences.

6. KeyBERT: KeyBERT is a pre-trained model based on word embeddings for n-gram words/phrases. Again we theorize that KECNW is slighlty better as KECNW is specifically trained on our corpus.

5.3.2 Analysis of Model/Similarity Metric

1. Jaccard : As it can be seen from the results, the Jaccard similarity metric retrieves poor results. The reason might be attributed to the fact that Jaccard is highly influenced by the size of the data. Large datasets can have a big impact on the index as it could significantly increase the union while keeping the intersection similar. This could result in hampering of the actual matching and hence poor retrieval. It is a highly volatile metric, giving better results to some queries and poorer results to other.

2. TF-IDF: It gives importance to terms rare in document collection but frequent in particular documents. TF-IDF computes document similarity directly in word-count space which is slow for large vocabularies. TF-IDF doesn't consider the semantic similarity between terms either. All the weighing schemes result in better nDCG scores than Jaccard since with the TF-IDF's valuation of each word in a document, it can identify words with the highest values and deem them to be keywords. It not only focuses on the frequency of words present in the corpus but also provides the importance of the words across the documents through the IDF score.

3. Binary Independence Model: BIM is a model traditionally used with PRP (Probability Ranking Principle). The 'binary' in BIM refers to documents and queries represented as binary term incidence vectors, and 'independence' means terms are modeled as occurring independently in documents. The model doesn't recognize association between terms but nevertheless is producing satisfactory results.

4. Cosine: Cosine similarity measures the cosine of angle between two vector representation of terms. Cosine similarity looks at 'directional similarity' rather than 'magnitudinal differences'. Cosine similarity produces satisfactory results both when vectors are represented using count vectorization and TF-IDF vectorization.

5. Okapi BM25: BM25 is a bag-of-words retrieval metric that ranks a corpus of documents based on the query terms (regardless of their proximity within the document) appearing in each document. It is a state-of-the-art probabilistic retrieval model, hence giving the best results on average. It improves upon TF-IDF by casting relevance as a probability through a relevance score, according to probabilistic information retrieval that reflects the probability a user will consider the result relevant.

6. Soft Cosine: It allows us to evaluate the similarity between two documents in a semantic way, even when the set of common words amongst them is void. Synonymity is modeled between the documents through word embeddings (Word2Vec, GloVe, etc.). Thus, the words with similar meanings are weighed more, giving better results than most of the other metrics.

One key observation needs to be highlighted here is that the metrics are reported by averaging across approximately 250 different queries. While we have reported and analyzed the best model based on the nDCG values across queries, it is possible that some other model may prove to be slightly better when considering significantly larger number of queries on a dynamic dataset of tweets and articles. This can explored as future work mentioned in Section 8.

6. INTERFACE

We have built a web-based system for the visualization of results, as seen in figure 6. The user enters the hashtag (case-sensitive), enters the location specific to which they need articles and selects the date from the dropdown. For demo purposes, we have given ability to the user to select the model/keyword extractor as per their will. Keeping human-centered design in mind, all such options will be removed if the website is deployed for public use and only the 'Submit' button will need to get clicked for the retrieval of results. The top k relevant articles (localised to the date and location entered by the user) are shown in the tabular form. The user can see the Rank (relevance order), Date, Location and the clickable article link.

Handling New Data: If the user enters a hashtag that is not present in our dataset or enters a date not in the range of our data-collection period (Feb 14, 2022 - Feb 28, 2022), the user has the option to view the Google Search results for the given query instead. The dialog box pops-up, as can be seen in figure7, giving the user such an option.

7. CONCLUSION

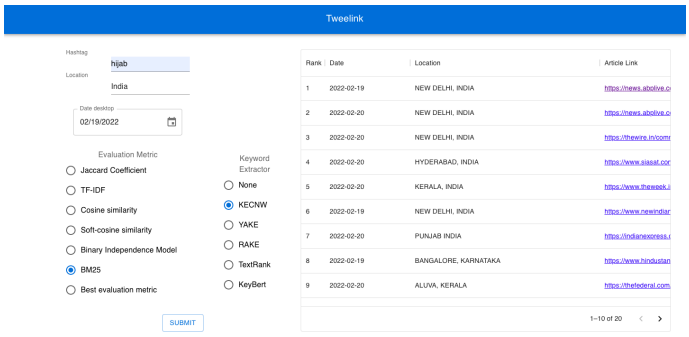

Figure 6: Web Interface of Tweelink (Query : hijab, 2022-02-19, India)

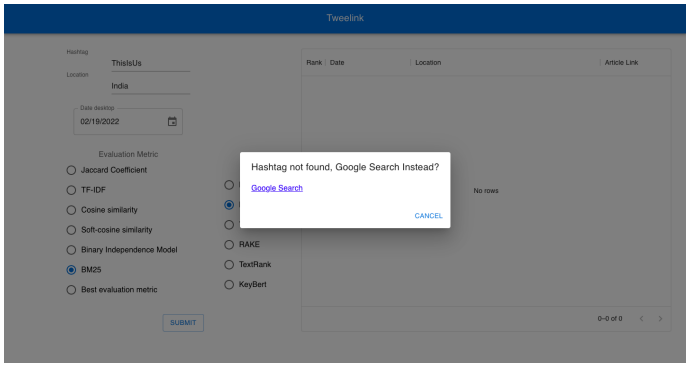

Figure 7: Invalid hashtag entered in our interface

We have been successful in tackling the formulated problem statement. Our proposed methods have improved the baseline models significantly as is evident from mean average precision, mean average recall, nDCG values. Based on the nDCG values we find KECNW terms fed into BM25 model to be best overall. KECNW manages to capture the relations and context between terms much more than YAKE, RAKE, TextRank and KeyBERT. This is due to the fact that KECNW extracts top keywords based on Node Edge (NE) rank and degree centrality. BM25 being state of the art is able to outperform other models.

8. LIMITATIONS AND CHALLENGES

1. Tweets are very short texts; regular NLP techniques may yield poor results.
2. The dataset of tweets and articles is static and should be dynamic in future work.
3. Tweets use very informal language and abbreviations.
4. Computation takes too long as datasets are too large.
5. Currenlty we only consider english dataset, which can be expanded to consider multilingual tweets and articles.
6. Extracting information from multimodal data including images, videos and other links present in the tweets has not been considered for current implementation.

9. FUTURE WORK

Till now we have used only textual information present in a tweet. KECNW considers relations between consecutive terms only. To better capture contextual similarity, using bag of words and skip-grams needs to be employed. Images, videos and external links present within a tweet may improve the model as well. Both our tweet and article datasets are static and can be dynamic and efficiently updatable. Our current algorithms need to be implemented more efficiently and parallelism/load balancing needs to be incorporated for practical deployment of our work. We have given ability to the user to localise their results to a given date. We can extend this and give the user ability to incorporate time stamp as well. As can be seen in the methodology workflow shown in figure 8, the tweets and articles can be penalised according to some gaussian distributions as per the exact time (mean) entered by the user.

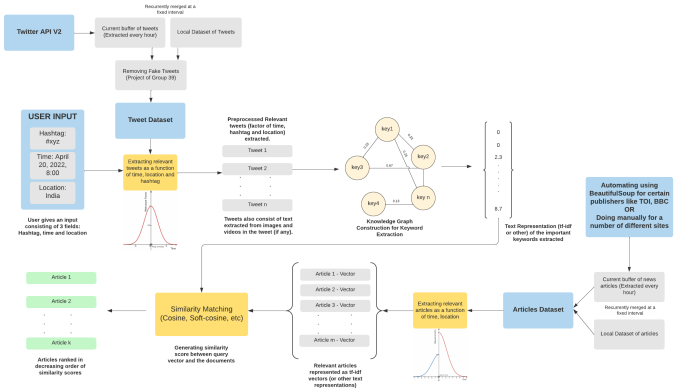

Figure 8: Future work with dynamic dataset

10. REFERENCES

[1] Bordoloi, M. and Biswas, S.Kr. (2018). Keyword extraction from micro-blogs using collective weight. Social Network Analysis and Mining, 8(1).

[2] Mihalcea, R., Corley, C. and Strapparava, C. (2017). Corpus-based and Knowledge-based Measures of Text Semantic Similarity. [online] Available at: https://www.aaai.org/Papers/AAAI/2006/AAAI06-123.pdf

[3] Prakoso, D.W., Abdi, A. and Amrit, C. (2021). Short text similarity measurement methods: a review. Soft Computing, 25(6), pp.4699–4723.

[4] Pradhan, N., Gyanchandani, M. and Wadhvani, R. (2015). A Review on Text Similarity Technique used in IR and its Application. International Journal of Computer Applications, 120(9), pp.29–34

[5] Campos, R., Mangaravite, V., Pasquali, A., Jorge, A., Nunes, C. and Jatowt, A. (2020). YAKE! Keyword extraction from single documents using multiple local features. Information Sciences, [online] 509, pp.257–289. Available at: https://www.sciencedirect.com/science/article/abs/pii [Accessed 11 Feb. 2022].

[6] Lösch, U. & Müller, D. (2011). Mapping microblog posts to encyclopedia articles. GI-Jahrestagung.

[7] Krestel, R., Werkmeister, T., Wiradarma, T.P. and Kasneci, G. (2015). Tweet-Recommender. Proceedings of the 24th International Conference on World Wide Web.

[8] Ahuja, S. (2015). Discovering significant news sources in Twitter. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/7434278 [Accessed 26 Feb. 2022].

[9] Del Corso, G.M., Gullí, A. and Romani, F. (2005). Ranking a stream of news. Proceedings of the 14th international conference on World Wide Web - WWW '05.

[10] Yilmaz, E., Aslam, J.A. and Robertson, S. (2008). A new rank correlation coefficient for information retrieval. Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '08.

[11] Biswas, S.Kr., Bordoloi, M. and Shreya, J. (2018). A graph based keyword extraction model using collective node weight. Expert Systems with Applications, 97, pp.51–59.

[12] github, 2021. GitHub, Available at: https://github.com/LIAAD/yake

[13] github, 2021. GitHub, Available at: https://github.com/vgrabovets/multi_rake

[14] https://github.com/summanlp/textrank

[15] https://github.com/MaartenGr/KeyBERT

[16] github, 2019. GitHub, Available at: https://github.com/laurabalasso/Binary-Independence-Model-

[17] Brown, D. (2020) Rank-BM25: A Collection of BM25 Algorithms in Python. Zenodo. doi: 10.5281/zenodo.4520057.

[18] Řehůřek, R. Sojka, P. (5 2010) 'Software Framework for Topic Modelling with Large Corpora', . Valetta, MT: University of Malta (Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks), . 45–50. : http://is.muni.cz/publication/884893/en.