A personalized search engine for microblog contents

Fabio Cimmino 807070
Roberto Lotterio 807500

Abstract

The goal of this project is to develop a personalized search engine for microblog contents.Twitter was chosen as the microblog with the related tweets contained therein. The search for terms can be both simple and customized based on a user profile. There are several user profiles and each user profile is organized in multi-layers that represent his interests according to various categories. Various interests are associated with a category of content for each layer. The user will be provided with a ranking of results based on topical relevance and other dimensions of relevance.

1 Data collection and dataset creation

We used Twitter4j [1], an unofficial Java library for the Twitter API, to collect tweets belonging to the following five categories: news, cinema, sport, science and music. The tweets found cover a period of one month.
We have chosen to take "mixed" tweets, that is, both popular and "common" tweets, excluding retweets. For each tweet we have chosen to store the following fields:

- date and time of publication of the tweet
- username
- content of the tweet

- number of tweet's likes
- number of tweet's retweets
- tweet's URL

All the tweets with the related data collected have been saved in a CSV file.

2 Stages of text processing

We carried out the following processing steps for the date and content of the tweet and then created the index necessary for the retrieval of the tweets.

Date and time: The format that Twitter provided us was of the *Simple-DateFormat* type, that is "EEE MMM dd HH:mm:ss zzz yyyy" (example: Mon Jan 20 01:07:12 CET 2020). We have therefore transformed the date into a simpler format, that is "dd-MM-yyyy HH:mm".

Content of the tweet: For this attribute we have created a Custom Analyzer which performs the following steps:

1. we used the Lucene's Standard Tokenizer that performs tokenization using the standard grammar suitable for the majority of uses in an IR system
2. each token has been transformed into lowercase characters
3. we removed the stop words using a pre-existing set of English words in Lucene
4. we have removed words consisting of less than two characters, numbers, links and special characters through the use of regular expressions
5. the last step was to perform the stemming on tokens with Porter's algorithm

3 Index creation

The fields in the index can be indexed and/or stored:

- Indexed - The field is analyzed and indexed, and can be searched

- Stored - The field's full text is stored and will be returned with search results

If a document is indexed but not stored, you can search for it, but it won't be returned with search results. The table 1 indicates, for each field taken into consideration, whether it is tokenized, indexed and / or stored.

Field	Tokenized	Indexed	Stored
date and time	NO	NO	YES
username	NO	YES	YES
content of tweet	YES	YES	YES
# likes	NO	YES	YES
# retweet	NO	YES	YES

Table 1: fields of each tweet

4 Scoring function

We used BM25 scoring function [2] to compute the similarity between a document d and a query q. BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document. The function is the following:

$$score(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{t \in q} \left(tf(t \in d) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t,d) \right) \tag{1}$$

- $tf(t,d)$ correlates to the term's frequency, defined as the number of times term t appears in the currently scored document d.

$$tf(t,d) = \sqrt{freq_{t,d}} \tag{2}$$

- $idf(t)$ stands for Inverse Document Frequency. This value correlates to the inverse of docFreq (the number of documents in which the term t appears).

$$idf(t) = 1 + \log\left(\frac{numDocs}{docFreq + 1}\right) \tag{3}$$

- $coord(q,d)$ is a score factor based on how many of the query terms are found in the specified document.

- $queryNorm(q)$ is a normalizing factor used to make scores between queries comparable.

$$queryNorm(q) = \frac{1}{\sqrt{sumOfSquaredWeights}} \tag{4}$$

Where:

$$sumOfSquaredWeights = q.getBoost()^2 \cdot \textstyle\sum_{t \in q} \left(idf(t) \cdot t.getBoost() \right)^2 \tag{5}$$

- $t.getBoost()$ is a search time boost of term t in the query q.
- norm(t,d) encapsulates a few (indexing time) boost and length factors:

$$norm(t,d) = doc.getBoost() \cdot lengthNorm \cdot \textstyle\prod_{\text{field f} \in \text{d named as t}} f.getBoost() \tag{6}$$

5 Query formulation

The first step in the query process standardizes the user request using the analysis process previously performed for the creation of the index:

- identification of the field
- stop-words handling
- management of the stemming

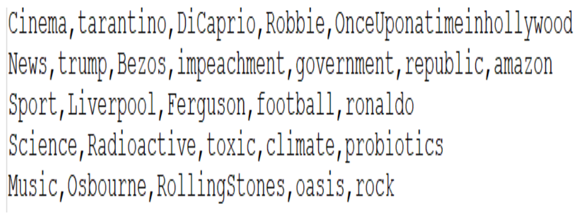
The query entered by the user will be searched in the field relating to the content of the tweet and it is analyzed with the Custom Analyzer used previously for the "content of the tweet" field.
The tokens and the search terms are identified in the string entered by the user and sent to the search engine for the matching phase. In addition, the logical connectors will also be identified. The default operator is the OR otherwise the type of operator (AND, OR, NOT) to be used must be explicitly indicated in the query.
The query supports using parentheses to group clauses to form sub queries. This can be very useful if you want to control the boolean logic for a query.

6 Custom flexible scoring and dimensions of relevance

Using the field type *DocValue* [3] in Lucene it is possible to store the field values of a specific document during indexing. The field will not be managed by the inverted-file but it will be stored in a data structure with direct-access. In this way we can incorporate the frequently changing fields in the boosting factor without re-indexing the document every time this factor changes and we can boosting of the objects which are the most liked or retweeted.
In order to scoring the documents, in addition to the topical relevance, it is also possible to take into account the number of likes or the number of retweets. To do this we multiply the calculated score value of each document with the relative number of likes or retweets (the value of the 'boost' field).

7 User profile

The search for tweets can be customized with the interests represented by a user profile. A user profile is composed of multiple layers where each layer indicates the user preferences associated with a specific category (music, science, news, sports and cinema).
Each user profile is saved in a CSV file and each line shows the interests of that user for cinema, news, sports, science and music respectively. The figure 1 shows an example of a user profile.



```
Cinema,tarantino,DiCaprio,Robbie,OnceUponatimeinhollywood
News,trump,Bezos,impeachment,government,republic,amazon
Sport,Liverpool,Ferguson,football,ronaldo
Science,Radioactive,toxic,climate,probiotics
Music,Osbourne,RollingStones,oasis,rock
```

Figure 1: example of user profile

The interests of each user were selected from the dataset containing the tweets going to inspect both their natural language content and hashtags.

8 Simple search and personalized search

The search can be simple or personalized. By simple search we mean the search of the query string only while for the advanced search you can customize the search based on the user profile, the category in which you are searching and the dimension of relevance. As for the user profile, it is possible to select one of ten with the possibility of indicating the category of interest that you want to consider. If the category is not selected, all interests of the user are taken into consideration. The personalized search can also add relevance for the number of likes or retweets.
For the creation of the final custom query we made the following assumptions:

- the terms entered in the search field must appear necessarily among the retrieved documents
- the terms of the user profile may not appear but, if they appear, the score of the retrieved document will be more highly.

9 Query expansion

Expanding the query can be very useful because the terms the user searched for may not appear in our documents. Furthermore most users find it difficult to formulate queries that are well designed for retrieval purposes. Yet, most users often need to reformulate their queries to obtain the results of their interest. Thus, the first query formulation should be treated as an initial attempt to retrieve relevant information. Documents initially retrieved could be analyzed for relevance and used to improve initial query (that also takes into account the number of retweets as a dimension of relevance).
To expand the query we used the following two methods in the literature [4]:

- Pseudo Relevance Feedback: this directly uses the top retrieved documents in response to the user's initial query for composing query expansion terms. Here, the user is not involved in the selection of relevant documents
- Automatic Local Analysis: identifies terms in the retrieved document that are close to the query (synonymity, morphological and derivational variations, terms that frequently co-occur with the query terms, etc.). The types of local analysis we considered are: Association cluster, Metric cluster and Scalar cluster.

9.1 Pseudo relevance feedback

With the pseudo relevance feedback the query expansion terms are extracted from the top-ranked documents in response to the initial query and it is based on Rocchio's algorithm [5]. The idea behind this technique is as follows: if the term weight vectors of the documents identified by the user are similar, then we should modify the query vector, so that it also similar to the ones in the relevant documents.
So we want to find a query vector that maximizes similarity with relevant documents (C_r) while minimizing similarity with nonrelevant documents (C_{nr}). The Rocchio's algorithm implements relevance feedback in the vector space model and it chooses the query $\vec{p}_{opt}$ that maximizes:

$$\vec{p}_{opt} = \text{argmax}_{\vec{q}}[\text{sim}(\vec{q}, C_r) - \text{sim}(\vec{q}, C_{nr})] \tag{7}$$

So $\vec{p}_{opt}$ is the vector that separates relevant and nonrelevant docs maximally. The optimal query is the vector difference between the centroids of the relevant and nonrelevant documents.
With the cosine similarity, this gives:

$$\vec{p}_{opt} = \frac{1}{|C_r|}\sum_{\vec{d}_j \in C_r} \vec{d}_j - \frac{1}{|C_{nr}|}\sum_{\vec{d}_j \in C_{nr}} \vec{d}_j \tag{8}$$

But there is a problem with the above metrics: the set of relevant documents is unknown. Therefore we produce the following modified query m:

$$\vec{q}_m = \alpha\vec{q}_0 + \beta\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}} \vec{d}_j \tag{9}$$

Where:

- $\vec{q}_0$ is the original query vector
- D_r is the set of known relevant documents
- D_{nr} is the set of known non-relevant documents
- α, β, γ are balancing weights

9.2 Automatic Local Analysis with clusters

For a given query q, let:

- D_l: set of documents retrieved by q
- N_l: number of documents D_l
- V_l: set of all distinct words in D_l
- $f_{i,j}$: frequency of occurrence of a term k_i in a document $d_j \in D_l$
- $\mathbf{M}_l = [m_{i,j}]$: term-document matrix with V_l rows and N_l columns
- $m_{i,j} = f_{i,j}$: an element of matrix $\mathbf{M}_l$

The matrix

$$\mathbf{C}_l = \mathbf{M}_l \mathbf{M}_l^T \tag{10}$$

is a local term-term correlation matrix.
Each element $c_{u,v} \in C_l$ expresses a correlation between terms k_u and k_v. This relationship between the terms is based on their joint co-occurrences inside documents of the collection. Higher the number of documents in which the two terms co-occur, stronger is this correlation.
Correlation strengths can be used to define local clusters of neighbor terms and terms in a same cluster can then be used for query expansion.

9.2.1 Association clusters

An association cluster is computed from a local correlation matrix C_l. For that, we re-define the correlation factors $c_{u,v}$ between any pair of terms k_u and k_v, as follows:

$$c_{u,v} = \sum_{d_j \in D_l} f_{u,j} \times f_{v,j} \tag{11}$$

In this case the correlation matrix is referred to as a local association matrix The motivation is that terms that co-occur frequently inside documents have a synonymity association. The correlation factors $c_{u,v}$ and the association matrix C_l are said to be unnormalized. An alternative is to normalize the correlation factors:

$$c'_{u,v} = \frac{c_{u,v}}{c_{u,u} + c_{v,v} - c_{u,v}} \tag{12}$$

In this case the association matrix C_l is said to be normalized.
Given a local association matrix C_l, we can use it to build local association clusters as follows:

- Let $C_u(n)$ be a function that returns the n largest factors $c_{u,v} \in C_l$, where v varies over the set of local terms and $v \neq u$. Then, $C_u(n)$ defines a local association cluster, a neighborhood, around the term k_u.
- Given a query q, we are normally interested in finding clusters only for the $|q|$ query terms. This means that such clusters can be computed efficiently at query time.

Consider the problem of expanding a given user query q with neighbor terms One possibility is to expand the query as follows:for each term $k_u \in q$, select m neighbor terms from the cluster $C_u(n)$ and add them to the query. This can be expressed as follow:

$$q_m = q \cup \{k_v | k_v \in C_u(n), k_v \in q\} \tag{13}$$

9.2.2 Metric clusters

Association clusters do not take into account where the terms occur in a document. However, two terms that occur in a same sentence tend to be more correlated. A metric cluster re-defines the correlation factors $c_{u,v}$ as a function of their distances in documents.
Let $k_u(n,j)$ be a function that returns the *nth* occurrence of term k_u in document d_j. Further, let $r(k_u(n,j), k_v(m,j))$ be a function that computes the distance between:

- the *nth* occurrence of term k_u in document d_j;
- the *mth* occurrence of term k_v in document d_j

We define:

$$c_{u,v} = \sum_{d_j \in D_l} \sum_n \sum_m \frac{1}{r(k_u(n,j), k_v(m,j))} \tag{14}$$

In this case the correlation matrix is referred to as a local metric matrix.

9.2.3 Scalar clusters

The correlation between two local terms can also be defined by comparing the neighborhoods of the two terms. The idea is that two terms with similar neighborhoods have some synonymity relationship.
In this case we say that the relationship is indirect or induced by the neighborhood. We can quantify this relationship comparing the neighborhoods of the terms through a scalar measure. For instance, the cosine of the angle between the two vectors is a popular scalar similarity measure.
Let:

- $\vec{s}_u = (c_{u,x_1}, c_{u,x_2}, \ldots, c_{u,x_n})$ vector of neighborhood correlation values for the term k_u
- $\vec{s}_v = (c_{v,y_1}, c_{v,y_2}, \ldots, c_{v,y_n})$ vector of neighborhood correlation values for the term k_v

We define:

$$c_{u,v} = \frac{\vec{s}_u \cdot \vec{s}_v}{|\vec{s}_u| \times |\vec{s}_v|} \tag{15}$$

The local scalar matrix C_l is said to be induced by the neighborhood.
Let $C_u(n)$ be a function that returns the n largest $c_{u,v}$ values in a local scalar matrix $C_l, v \neq u$. Then, $C_u(n)$ defines a scalar cluster around term k_u.

10 Web application

To create the web application we used Sping Boot with Thymeleaf.
Spring Boot is the simplest solution for programming and testing applications on the Spring platform, producing stand-alone Apps that can be run in JAVA.
Thymeleaf is a modern server-side Java template engine for both web and standalone environments. In web applications Thymeleaf aims to be a complete substitute for JavaServer Pages (JSP), and implements the concept of Natural Templates: template files that can be directly opened in browsers and that still display correctly as web pages. In the figure 2 you can see the home page of our search engine. In this page it is possible to carry out a simple search otherwise, by pressing the "options" button, you can carry out a customized search.

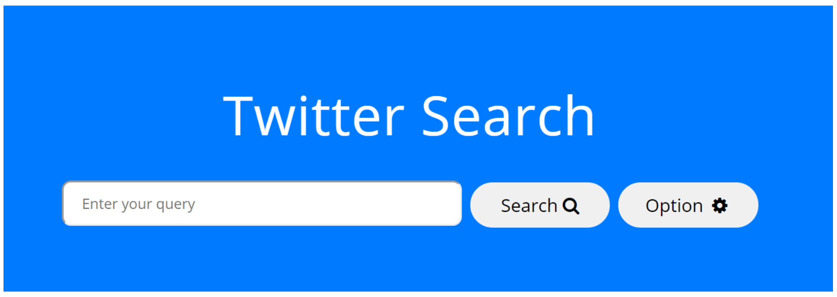


Figure 2: Web app homepage

Clicking the "options" button the web page shown in Figure 3 is displayed. On this page you can carry out by combining different customization options:

- user profile
- interest category associated with the chosen user profile
- additional dimension of relevance (number of likes or retweets)
- method for expanding the query (Rocchio, Association clusters, Metric clusters, Scalar clusters)

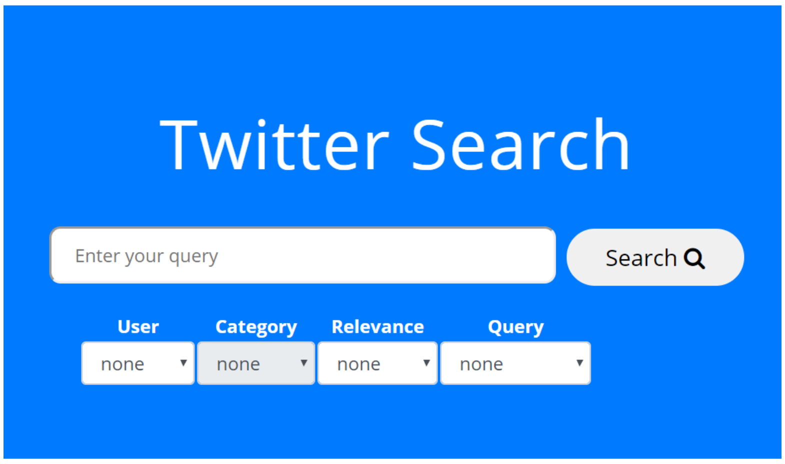
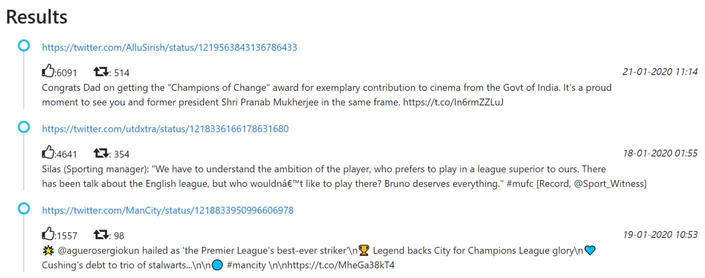


Figure 3: search options for personalized search

The search results are displayed as in Figure 4.



Results

https://twitter.com/AlluSirish/status/1219563843136786433

:6091 : 514 *21-01-2020 11:14*

Congrats Dad on getting the "Champions of Change" award for exemplary contribution to cinema from the Govt of India. It's a proud moment to see you and former president Shri Pranab Mukherjee in the same frame. https://t.co/In6rmZZLuJ

https://twitter.com/utdxtra/status/1218336166178631680

:4641 : 354 *18-01-2020 01:55*

Silas (Sporting manager): "We have to understand the ambition of the player, who prefers to play in a league superior to ours. There has been talk about the English league, but who wouldnâ€™t like to play there? Bruno deserves everything." #mufc [Record, @Sport_Witness]

https://twitter.com/ManCity/status/1218833950996606978

:1557 : 98 *19-01-2020 10:53*

🌟 @aguerosergiokun hailed as 'the Premier League's best-ever striker'\n🏆 Legend backs City for Champions League glory\n💙 Cushing's debt to trio of stalwarts...\n\n🔵 #mancity \n\nhttps://t.co/MheGa38kT4

Figure 4: search results

References

[1] Yusuke Yamamoto. Twitter4j. *"http://twitter4j.org/en/index.html"*, 2008.

[2] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[3] Class docvalues lucene. `https://lucene.apache.org/core/8_0_0//core/org/apache/lucene/index/DocValues.html`.

[4] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[5] Joseph Rocchio. Relevance feedback in information retrieval. *The Smart retrieval system-experiments in automatic document processing*, pages 313–323, 1971.