Università degli studi di
Milano-Bicocca

Information Retrival

Final Project

Tweetify: personalized search engine for tweets

Authors:
Matteo Pelucchi: 806798- m.pelucchi@campus.unimib.it
Luca Romanato: 807691- l.romanato1@campus.unimib.it

February 4, 2020

1 Introduction

The aim of the project is to develop a search engine or a recommender system on a set of textual content following certain requirements.

The following project implemented a custom search engine for tweets based on user's preferences.

2 Data

The dataset used is a tweets set crawled through the Twitter API via tweepy, containing 77.361 tweets and retweets of the last two month coming from several famous accounts. For each tweet it was stored user, text, location and date for a maximum of 10000 tweets for channel.

Furthermore, it was added the topic based on the channel, in detail topic can assume the value of:

- Photography, for accounts like NatGeoPhotos, NASAHubble and AP_Magazine
- Science, for NASA, sciencemagazine and DiscoverMag
- Music, for SpinninRecords, billboard and RollingStone
- Sport, for NYDNSports, SkySportsNews and BBCSport
- Politics, for nytpolitics, NBCPolitics and CNNPolitics
- Tech, for TechCrunch, ForbesTech and thenextweb
- Finance, for business, CNNBusiness and TheEconomist
- Cinema, for IMDb, netflix and THR

On this tweets it was not done any sort of preprocessing, like stopwords removal, because they will be the system search result.

Instead, for users who will use this search engine it was collected only the text from their tweets and retweets of December and January. This was done to create the users preferences automatically, represented by bag of words. To create bag of words it was applied a preprocessing phase which has been composed by 2 main section. First section is about all text preprocessing, like:

- conversion into a lowercase text
- contractions are expanded, like "aren't" becomes "are not"
- all links in text are removed

Second section is composed by the text tokenization and the removal of:

- stopwords, like numbers
- punctuation
- emoji
- whitespace

Finally the lemmatization was applied and the Bag of Words was created considering the first 15 common words.

the users chosen are:

- Paul Nicklen, Canadian photographer, film-maker, environmentalist and marine biologist
- Kevin Hart, American stand-up comedian, actor, and producer
- Brian Cox, English physicist
- Elon Musk, an engineer and technology entrepreneur
- Edward Snowden, American whistleblower
- Jeff Jarvis, American journalist, associate professor, public speaker and former television critic.
- John Oliver, English-American comedian, writer, producer, political commentator, actor, and television host
- James Cameron, Canadian filmmaker, artist, and environmentalist
- Jeremy Clarkson, English broadcaster, journalist and writer who specialises in motoring
- James White, sport broadcaster

3 Search engine

The search engine has been developed using ElasticSearch, a distributed open source search and analytics engine for all types of data built on apache Lucene.

In particular, the project has been written in Python 3.7 using elasticsearch library (7.5.1) that wraps ElasticSearch's low-level APIs to Python. It provides a more convenient and idiomatic way to write and manipulate queries.

3.1 Index creation

The index is based on tweet file's attributes (crawling phase, section **??**) to enable querying operations.

Before proceeding with the insertion of documents (tweets) it is necessary to define the mapping structure. The **mapping** is structured as a tree and is based on five twitter attributes: data, of type date, text, of type text, and the remaining of type keyword.

Furthermore, different **settings** have been set for index mapping as follow:

- **Index**: The number_of_shards has been set up equal to 6 and number_of_replicas to 1. That means each document is stored into six shards and one replica. That parameters allow to boost the indexing phase, so it results more efficient from the temporal point of view.
- **tweet_text_analyzer**: This setting is used for the text field, consists of a standard tokenizer based on whitespace separator, a filter for removing english stopwords, a normalize phase to lowercasing the text and finally a stemmer algorithm for stemming.
- **keyword lowercase**: This setting is used for keyword fields (i.e. topic, user and location), consists in a normalize phase to lowercasing the text.

Finally, for date field have been specified the current **format**: "yyyy-MM-dd HH:mm:ss", that is the one used by twitter.

3.2 Query operation

ElasticSearch has several types of predefined similarity algorithms and also allows to define custom ones. For this project, it has been decided to use the

default similarity, that is based on the **BM25** algorithm. It 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. It is a family of scoring functions with slightly different components and parameters:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \tag{1}$$

Where:

- $IDF(q_i)$ is the IDF (inverse document frequency) weight of the query term qi
- $f(q_i; D)$: qi's term frequency in the document D.
- $avgdl$: average document length in the text collection from which documents are drawn
- b and $k1$: free parameters, usually chosen, in absence of an advanced optimization, as k/in [1.2,2.0] and $b = 0.75$

In this project has been used the default options of the free parameters: k = 1.2, b = 0.75 and discount_overlaps = true.

Given that, the goal of this project is to build a personalized search engine. In order to allow personalized search, different relevance dimensions have been provided, based both on user profile and tweet characteristics.

In particular, the user through a interface can decide to perform: a free search, a specific text search (using quotes) or a search in which are specified individually or in combination: topic, date of the tweet and page of the tweet.

As such, to allow different configurations according to the user's preferences, the query structure has been implemented as modular and has been applied pre-processing approach as personalization process.

The query structure is as follows:

- **size/from**: these parameters allow pagination of results. The from parameter defines the offset from the first result you want to fetch. The size parameter allows you to configure the maximum amount of hits to be returned

If it is not an exact text search:

- **Function Score**: is a wrapper of the Boolean query that allow to modify the score of documents that are retrieved by a query according to the fields specify by the user. It contains:
  - **Boolean query**: A query that matches documents matching boolean combinations of other queries. It is built using one or more boolean clauses, each clause with a typed occurrence. In particular has been used two occurrence types: **Must** and **Should**.
    - **Must queryset**: this attribute contains all the clauses that must appear in matching documents and will contribute to the score. In this project, it always contains a query string reference to tweet's text field but can also contain topic field, data field and user field if specified by the user.
    - **Should queryset**: this attribute contains those clauses that should appear in the matching documents. These are terms that don't have to appear in the documents, but their presence increases the score. has been used to implement different relevance dimensions in the retrieval process. The should queryset has been used to mapped user profile, in particular user's interest (bag of words) and language, in term of match queries on fields topic and location
  - **Decay function**: this function parameter scores documents with a decay criteria that depending on the distance of a document numeric field value from a given origin. In this case, it has been used to decrement the value of the retrieval status according to how recent the tweets are (date field). The decay function used is **gauss** with origin date set to **2020-01-01 00:00:00**, scale equals to **30d**, offset equals to **1d** and decay value set to **0.3**. This settings allow to applied decay value to all results not in the 31 days before the origin data and to apply decay function to all results in this range.

If it is an exact text search:

- **Match phrase**: provides only the tweets that contains exactly the query string as sub sentence of the tweet's text

3.3 Query example

The above structure allows you to make the following types of queries:

1. Generic query : *Nasa space shuttle*
2. Query with specific topic field that can be choose on the user interface
3. Query with fields to search text for user and data fields: *user:NasaHUBBLE data:2020-02-01* space this can also be combined with a specific topic field
4. Boolean queries or a query that use the syntax of Boolean operators: *(user:NasaHUBBLE AND data:2020-02-01) OR space*
5. Quotes query (".."), exact search of sentences in the tweet's text: *"Kobe Bryant"*

4 User interface and frameworks

The graphical interface is provided to the user through Flask. Flask is a lightweight WSGI python web application framework. Therefore, we have also used html, css and javascript for the web design.

The interface provides a login page and a search page with a search bar, a topic selection, visualization of user's bow, visualization of search result and selection of other user profile.

The project code and related instructions can be found in the following repository: `https://github.com/LucaRomanato/IR_project.git`

5 Conclusion

In conclusion, the search engine allow to search tweets by keyword, phrases and boolean expressions with a good accuracy in the resulting tweets. The results change evidently according to the user preferences, the query "Trump" searched by Elon Musk return tweets about Tesla and SpaceX, meanwhile, tweets about Trump declarations on CIA, whistleblower and internal security are in the top results if the user is Edward Snowden.