Information Retrieval

2021-2-F1801Q110

**Gianmaria Balducci**

Project A: Personalized Search Engine for microblog

# Introduction

The project consist in implementation of a personalized search engine for tweets, the search of twitter’s data must be affected by the user preferences taking into account his past tweets.

For this purpose this project is developed with **Python 3.8** with **tweepy** API that is build over Twitter API and offers many functions in order to download tweets and manage twitter data.

Tweets preprocessing is developed using regex and **nltk** and for word embeddings is used **Word2vec** with CBOW architecture. **Elastic search** is used to index all the documents downloaded and to manage the query processing and the web app is dveloped with **Flask** framework.

I choose 9 users realted to 4 topics and then downloaded all tweets for every user and all possible tweets allow by Twitter API for every topic. All the tweets are preprocessed and then retrived by elastisearch. To represent user profile i have implemented word embeddings approach, so for an user query are retrived 10 top tweets that are affected by the user profile.

This project is entirely developed and testes on Ubuntu 20.04.

# Dataset

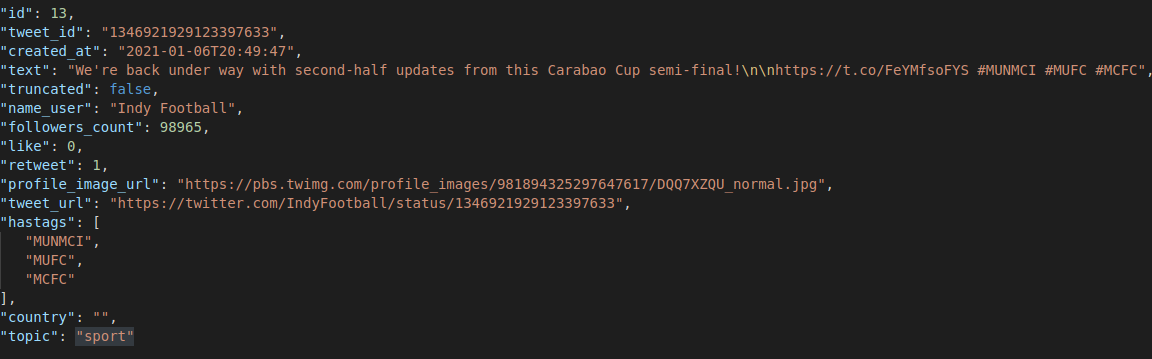
Dataset has been downloaded with tweepy which allows you to collect more tweets than Twitter API (that allow to collect 100 tweets for request) allows by keeping track of the ids of the tweets.

With the use of *max\_id* and *since\_id* **Hunter** (name of this application) for every topic iterates over the request developed with tweepy until it reaches a predefined number of tweets.

While user’s tweets can reach a maximum of 3200 for every user because Twitter API doesn’t allow to collect more.

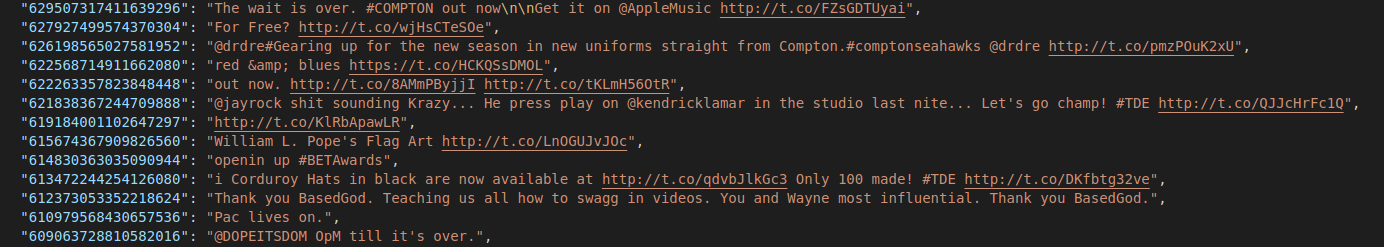
Repository of tweets is divided in two sets:

**-topics\_tweets** that contain all the tweets retrived by 4 topics: **music, cinema, technology** and **sport.** Tweepy can get tweets in **JSON** format but not all the attributes are stored in *topics.json* but few of them that are listed below:



So I’ve keeped the structure of tweepy response excluding entities and keeping these attributes, adding **hashtag** attribute in order to allow search by hashtags in the user interface.

**-users\_tweets** that contain the entire timeline for each user that is related with one topic. All the tweets for every user is stored in repository directory in a json file named *@username.json which contain*:



As shown above for a specific use Hunter collect only id and text accessed by keys.

These tweets will represent user profile.

So I’ve collected 3 users with cinema interest, 2 users with sport interest 2 user related to technology interest and 2 related to music interest. Number of tweets, name of the users are described the tables below:

|  |  |  |
| --- | --- | --- |
| **USER** | **INTEREST** | **#TWEETS** |
| Charlize Theron | cinema | 2163 |
| Bryan Cranston | cinema | 784 |
| Samuel L. Jackson | cinema | 3200 |
| Bruce Springsteen | music | 3221 |
| Kendrick Lamar | music | 3148 |
| Lebron James | sport | 3241 |
| Wayne Rooney | sport | 2242 |
| TechCrunch | technology | 3245 |
| Elon Musk | technology | 3222 |

|  |  |
| --- | --- |
| **TOPIC** | **#Tweets** |
| Cinema | 65482 |
| Sport | 150066 |
| Technology | 150090 |
| Music | 128589 |
| **TOT** | **494’227** |

# Search Engine

After collecting all the data the followings steps are implemented and described in details in their section:

* Preprocessing- cleaning and tokenization
* Word embeddings to incapsulate user profile and all the tweets
* Indexing and Search
* Test search with a simple interface

**Preprocessing**

Microblog as tweets are full of social object such as hashtags, tags, username, hyperlinks, emoji, emoticons ecc.. This elements can introduce noisy and tends to increase complessity of the search. So, I’ve decided to remove the majority of these elements and keep the hashtags in the text of microblogs removig only the # sign.

Furthermore for every tweet all the hashtags are extracted to create “hashtags” attribute to allow the search on this fiield.

Emoji indeed are converted in text and separated by a space, iv’e decided to keep this element becuase is one of the most important element of the social media.

In particular Hunter applies the followings operation over every tweet:

* **removal of html elements**
* **remove hyperlinks**
* **remove url**
* **remove numbers**
* **remove usernames**
* **remove puntuaction**

So for example a tweet

“My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… [https://t,co/3tfYom0N1i](https://t.co/3tfYom0N1i)”

became:

My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites happy Friday off…

Emoji in this case:

'Happy with this bit of business? 🔵⚪\nhttps://t.co/ykFkfETWvJ',

become:

'Happy with this bit of business bluecircle’

BEFORE CLEANING:

@GazBigBaller @FredWestPatios @GloucesterBint Is it an Olympic sport yet?

AFTER CLEANING:

Is it an Olympic sport yet

Followings steps are tokenization by wordtokenizer, pos tagging and lemmatization with WordNetLemmatizer.

Pos tagging is done with *nltk.pos\_tag()* that is a function that use a pretrained model and return a pos tagging form a sentence. Elements tagged are : [ADJ, ADJ\_SAT, NOUN, VERB, ADV]. These tagged tokens are used for lemmatize the tokens, without pos tagging lemmatizer tags all the tokens as nouns and this can give different form of token rapresentation. For example *lemmatize(stripes, noun) = stripe while lemmatize(stripe, verb) = strip.* So I’ve decide to implement pos tagging to improve lemmatization.

More in detail these are the operations:

* **token.lower()**
* **remove tokens in stopwords(“english”)**
* **remove single char tokens**
* **pos tagging**
* **lemmatization**

**examples:**

BEFORE TOKENIZATION:

Premier League final table predicted as Man City beat Liverpool Man Utd below Tottenham

AFTER TOKENIZATION:

['premier', 'league', 'final', 'table', 'predict', 'man', 'city', 'beat', 'liverpool', 'man', 'utd', 'tottenham']

BEFORE TOKENIZATION:

LeBron “Sport is not sport without fans

AFTER TOKENIZATION:

['lebron', 'sport', 'sport', 'fan']

These preprocessed tweets are given at word2vec to create user profile and query embedding

**Word Embeddings**

For word embeddings this project use Word2vec as seen as practical lessons.

I choose this approach to incapsulate **user profile** because Word2Vec has several advantages over bag of words and IF-IDF scheme. Word2Vec retains the semantic meaning of different words in a document. The context information is not lost. Another great advantage of Word2Vec approach is that the size of the embedding vector is very small. Each dimension in the embedding vector contains information about one aspect of the word. We do not need huge sparse vectors, unlike the bag of words and TF-IDF approaches.

Wod2vec has 2 architecture CBOW and SkipGram, in the Skip Gram model, the context words are predicted using the base word. For instance, given a sentence "I love to dance in the rain", the skip gram model will predict "love" and "dance" given the word "to" as input.

On the contrary, the CBOW model will predict "to", if the context words "love" and "dance" are fed as input to the model. The model learns these relationships using deep neural networks.

For this application i choose CBOW architecture and i collect all the models in *models* folder in the project. Every user has his model build over his tweets publications. Also for topics\_tweets, this tweets preprocessed are taking into word2vec to create a separate model to compare similarity with query terms.

So **user profile** is represented by (@username.model) a model built with word2vec

taking into account all his preprocessed tweets publicated and in search phase top 10 terms similar to query terms are put in should list in the boolean query.

**Indexing**

Index named **twindex** is built using Elasticsearch mapping that define how tweets are processed and saved into index. For this purpose is defined a custom\_analyzer, with properties that convert a tweet into tokens. This process is make by the followings properties and filterings:

* **whitespace tokenizer**
* **char filtering html strip**
* **classic filter** - removes the English possessive from the end of words and removes dots from acronyms
* **synonym filter:** custom filter to expand query terms with their synonyms
* **lowercase filter**
* **remove on char token**
* **remove puntuaction**
* **remove link**
* **remove digits**
* **stemmer** with porter2

To expand query terms with their synonym in the index phase I’ve added a list of 60 english words with their synonym, if a query term is in the list search results taking in consideration also synonym of this term.

The **similarity** measure that will define similarity beetween query terms and the documents, for this purpose i choose LM Dirichlet Similarity where the scoring formula in the paper assigns negative scores to terms that have fewer occurrences than predicted by the language model, which is illegal to Lucene, so such terms get a score of 0.

After this process we have to map the properties of the documents and decide which field to index. I’ve decide to index **username** , **topic** and **hashtags**, that are needed to retrive documents, others field are stored but not indexed.

**Search**

Search is based on the fact that the tweets more relevant are retweeted several times and have more likes that others (classic social paradigm), so i modified the query’ s function score in order to assign higher RSV (Relevant Status Value) to those with a higher number of retweets and likes.

The user can choose a field between: text, username and hashtags, this choice determine the default\_field for the query. In addition the search can be done over topic field in combination with one of default field, and retrive tweets specific to that topic and that contain in username or in text or in hashtags the query terms.

Query is **boolean** and is split in two set: **must** and **should**.

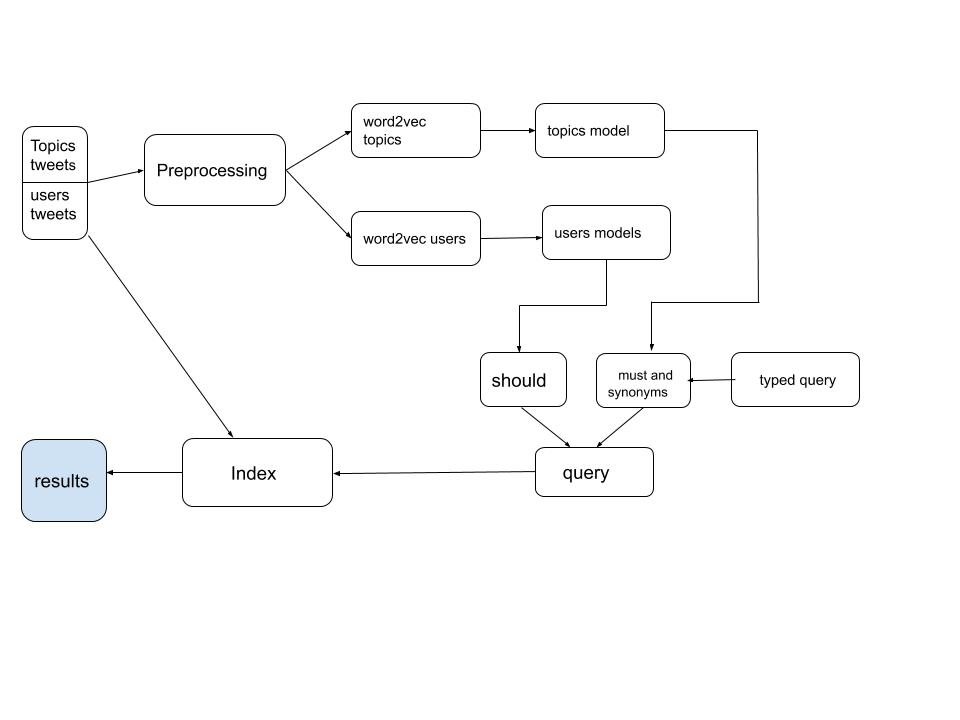
Must contain all the terms digits in the query and indicate that elasticsearch have to retrive tweets that have equal terms with the query, in boolean query must indicate an and clause, so if i have q1,q2,q3 documents retrived must contain all the terms.

But introducing a list of synonyms these terms if are present in the list are **expanded** with their **synonyms** making the search less rigid.

All the documents matching must terms have a very high RSV while taking into account a generic **user profile,** should list contain the 10 terms most similar (contained in a specific user model), to the query terms, this can increase RSV

Should indicate an or in boolean query, therefore tweets retrived can have or not terms specified by this list.

The following image represent data flow.



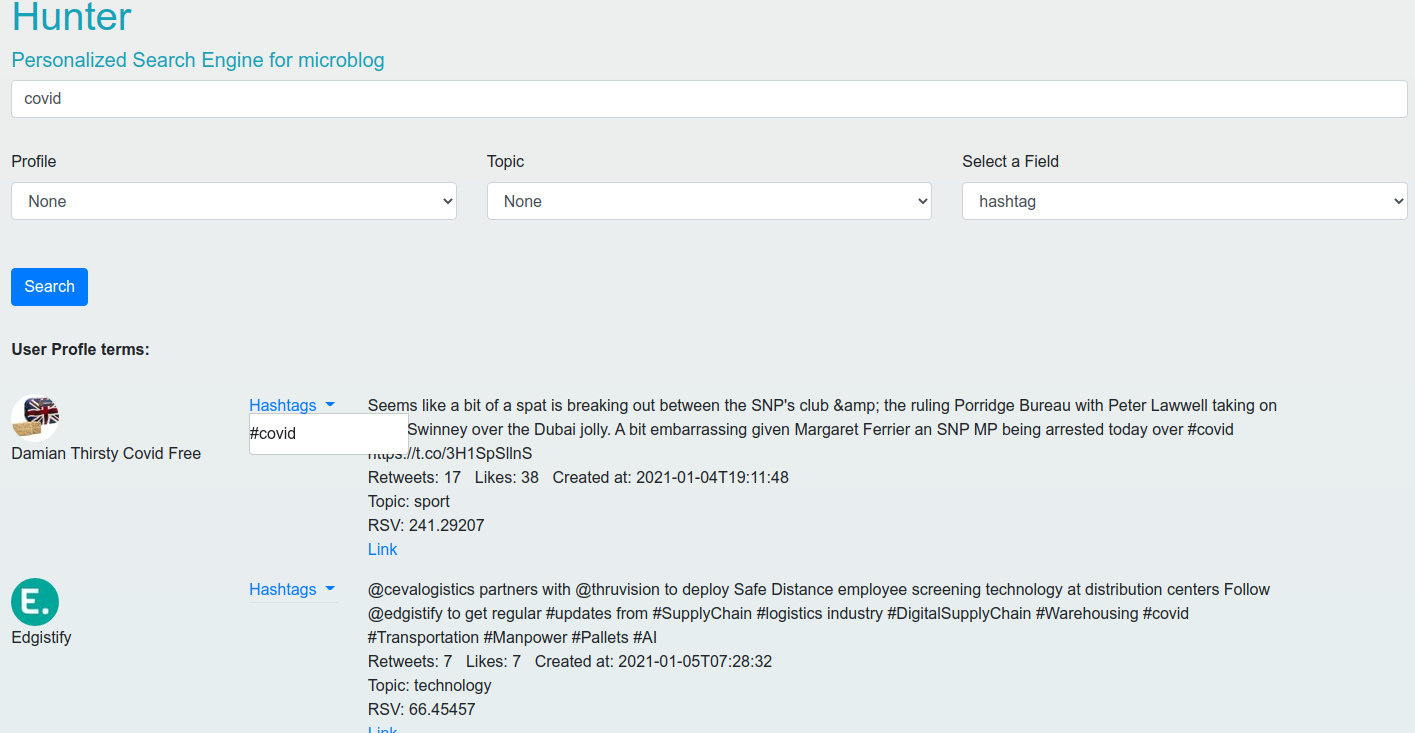
# Demonstration plan

**User cases**

**1-Textual search on a specific field using keywords**

In this project textual search can be done over a specific field such as topic, text, hashtags and username. In the indexing phase fields profile\_image\_url, topic and hashtags are set to type: keyword. The results over a specific field are shown in the following images: i will use user interface for show the results of a query.





For example here i search for hashtag covid in field hashtag.

As you see the documents retrived show their user, text and hashtags if are present.

Under text of tweet you can see number of **retweets** and **likes** that impact the RSV, and lastly topic and a link to original tweet.

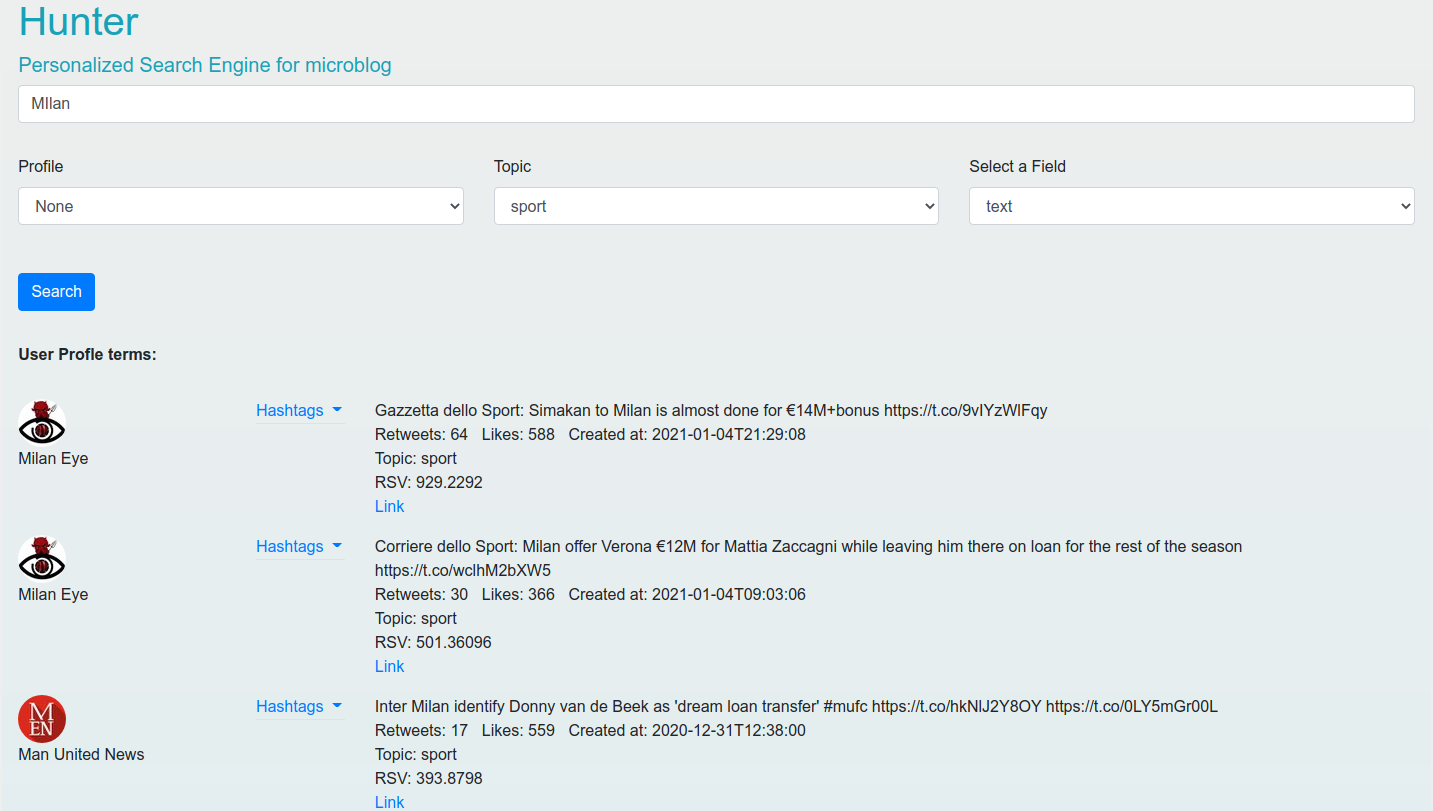
In this case the search is done over the specific field hashtags that is a keyword.

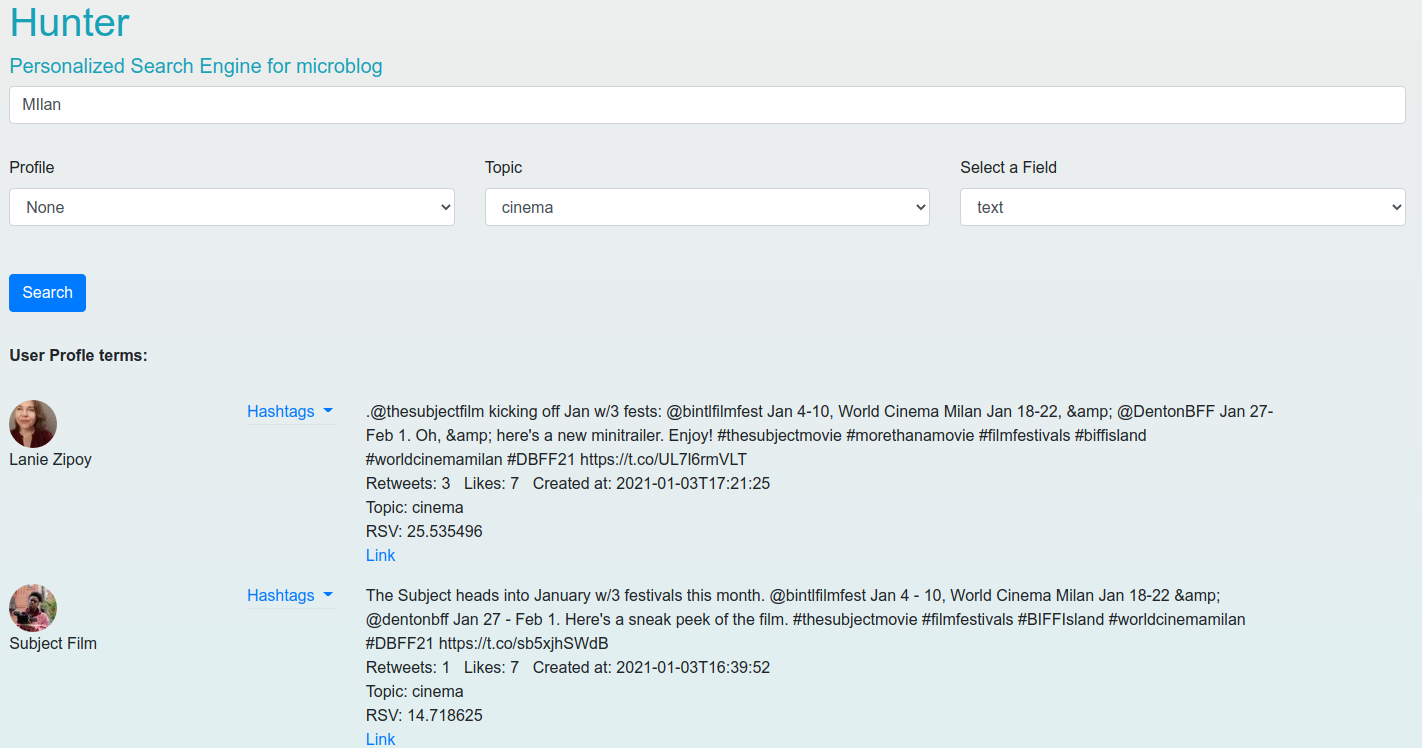
**2- Textual search on a combination of fields**

In this case a combination of fields can be produced by a choice of specific topic ( music, cinema, sport, technology) in the user interface and one of fields between hashtags, username and text.

For example in the image below the search is “Milan” for field text and i selelct **sport** topic to retrive only tweets that have topic equal to sport.

And in the second image same query “Milan” but with cinema topic, let’s see.



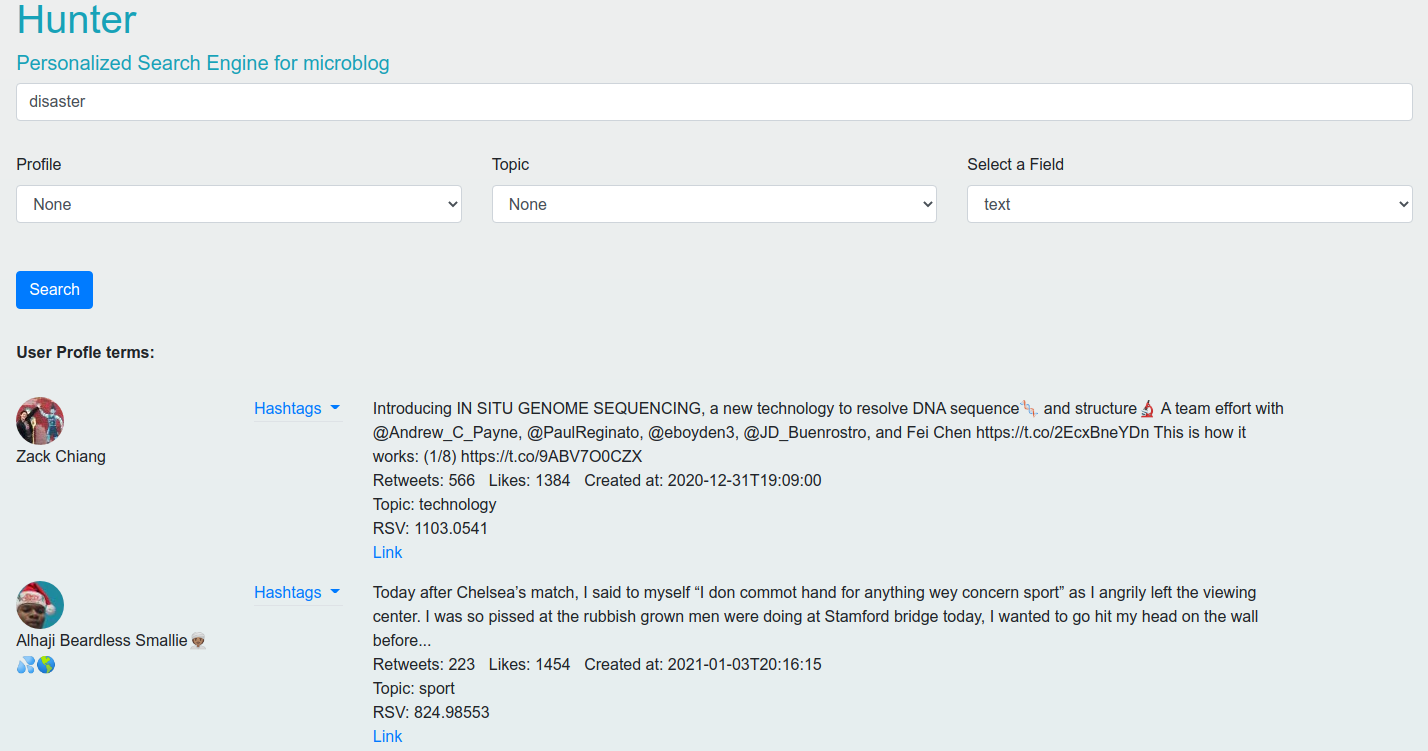


Changing topic field prduce very different result in first case Milan is focused on A.C. Milan and Inter Milano that are football teams, while in the second case Milan is interpreted as the city of Milan.

**3-Rank tweets taking into account the user profile**

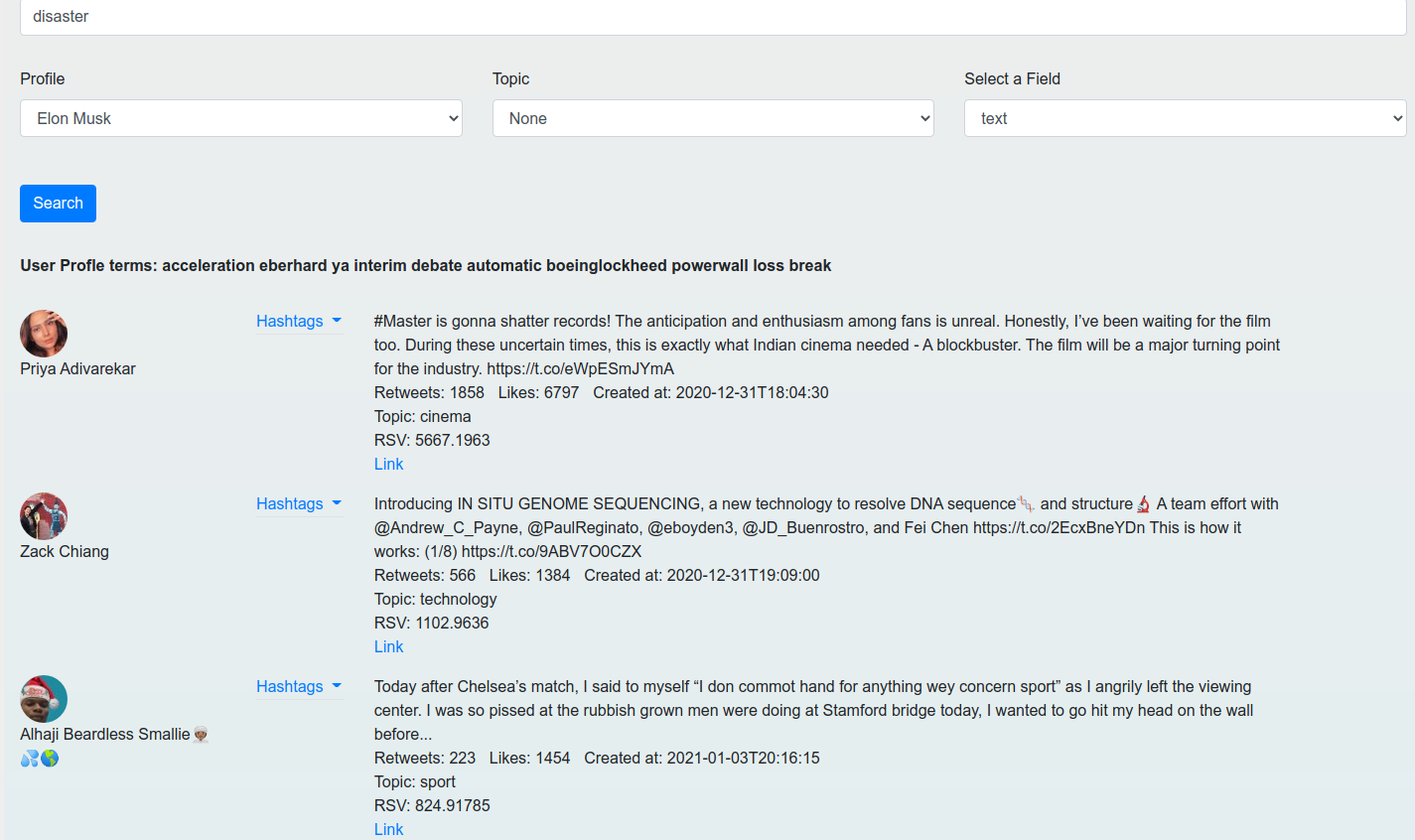
On the interface there is a possibility to choose a profile between users that i choose described in the previous sections. When a user is selected, this application taking in to account the user selected model created with word2vec and compare the similarity between query term and his model, the most similar ten are choose and puts into should list inside the query.

An example can be query “disaster” that, without user profile selected has this results:



While when an user is selected for example Elon Musk, the results list is changed maybe only in the order, maybe also the list of documents retrived, depends on the query typed, in this case with elon musk profile, the top documents have a different RSV because the words retrived by user model are inside should list and affect the query:





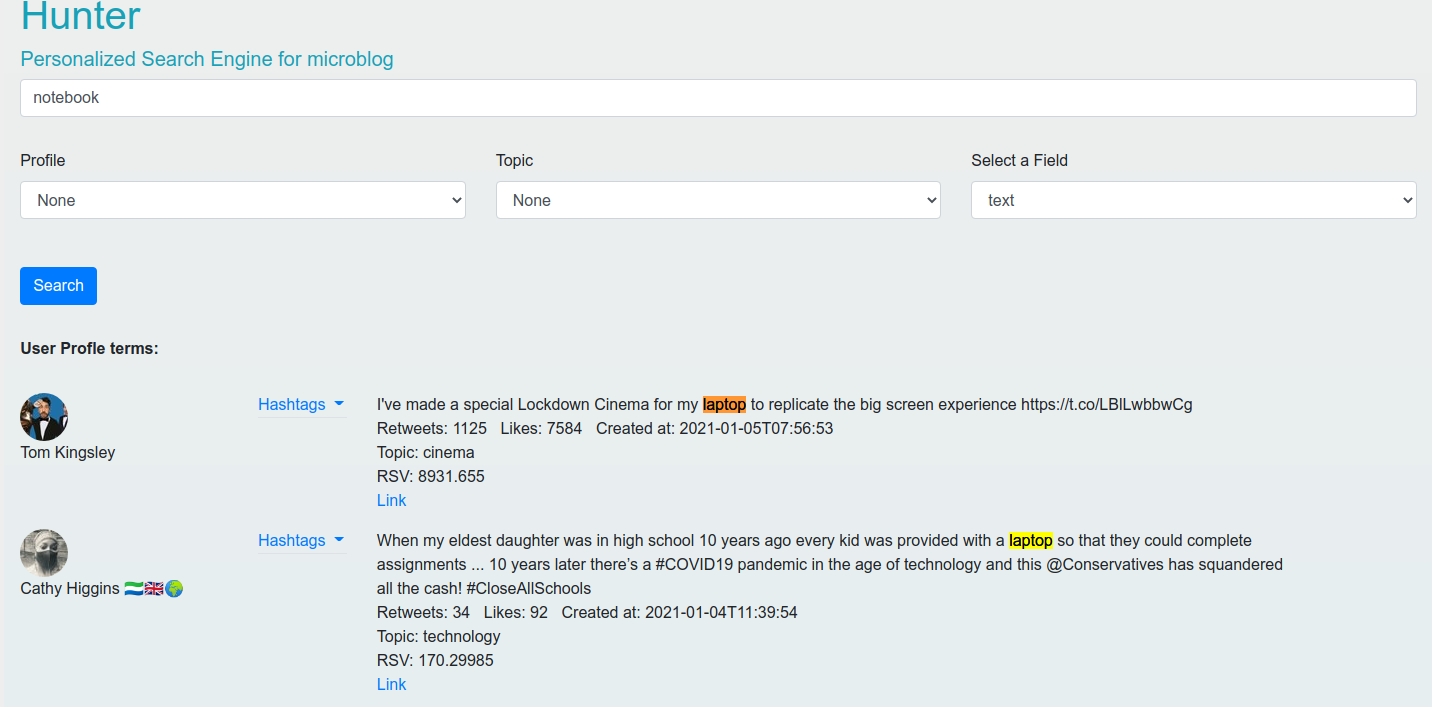
**4- Expand the search adding synonymous of the words in the query**

For this purpose i choose the approach at index time and i have mapped some of the most used words in english and their synonym.

For example:



Infact the results taking in consideration the query and if is present in this little dictionary, his synonyms. If we type a query with “notebook” word inside, the documents are retrived also with a matching with word “laptop”:



# Future developments

This is a simple search engine, to improve this project it needed to collect more tweets for more efficient results and edit the search in such a way to allow more complicate queries, another aspect is to create a very huge dictionary of synonyms and compare with the query terms at query time.