Multilingual Search System

Team Untitled - Report

https://github.com/agsimeonov/cse535-final

CSE 535: Information Retrieval

Fall 2015

University at Buffalo

Highlights

A pure multilingual search

Can handle queries in 5 different languages

(English, Russian, German, French and Arabic)

Based on twitter data corpus

with data of around 0.1 million tweets

Spanning more than 150 countries

1. Data Aggregation

Data for this project was collected from Twitter using the Twitter API. In addition a Geocoding API was used in order to derive the country of origin for some tweets.

1.1 Scraper https://github.com/agsimeonov/cse535-final/tree/master/scraper

The scraper is a python script using the tweepy library and the Twitter search API. This script is multithreaded and takes in a configuration file which can have multiple API keys at one time (one for every thread). The reason we chose a multithreaded approach is that we can collect multiple language tweets at a much faster rate (one language per thread). In fact our aggregator is very resistant to API limitation which dictate that once the request limit is reached a connection over a single API key must wait for 15 minutes.

Our threads detect when we have reach such a state and wait the necessary time. Using the search API we are able to collect data for different languages over the course of 7 days. We are limited to 7 days as the search API would only allow us to go that far back in time. We left our scraper to run overnight and collected close to a **100000 tweets**. We stored the tweets in their raw for later processing. We did this so that we can have the flexibility to use more fields if we later decide to add functionality to our search system.

1.2 Geocoder https://github.com/agsimeonov/cse535-final/blob/master/crossdoc/geocoder.py

We wanted to do some interesting things based on tweet location. However determining the tweet location is no easy task. We noticed that less than a 100 of the tweets we gathered have coordinates associated with them. However a huge chunk of the tweets had the user location field filled out. Unfortunately the user location field is user specified and can containing anything including a large variety of languages and invalid locations. Most users do tend to, however, fill out a valid location so we used the Open Street Maps geocoding API in order to both translate the user location and extract the country associated with it. When coordinates were available we used them instead. Most geocoding APIs are paid or have severe limitations. The Open Street Maps limitation was that we can’t send too many requests at one time, 1 per second.

Our geocoder script uses this API and sends requests at the defined speed of 1 per second. In case of a timeout the script has a wait period after which it continues onwards. The script outputs a file containing a map of tweet IDs and country names. We left the geocoder running for over 24 hours in order to figure out the actual locations of our tweets. Over 40000 of our tweets ended up with an associated country, a sizable chunk indeed.

1.3 Post-Processing https://github.com/agsimeonov/cse535-final/blob/master/scripts/JsonBeauty.java

The next step in the process was to process the raw data and extract some of the important fields which we can use for indexing purpose. Since we were building a complete multilingual system and the raw data was not in a format as solr expects it to be, like the format of date, splitting of text based on language, extracting data specific to tweet and user. Moreover the data from Geocoder also need to be merged. So we wrote our own post processing script which does all this processing for us.

We **collected more than 25 fields per tweet** which we finally decided to use for indexing purpose. These fields were tweet_id, tweet_created_at, tweet_lang, tweet_text_en, tweet_text_ru, tweet_text_de, tweet_text_fr, tweet_text_ar, tweet_favorite_count, tweet_retweet_count, tweet_urls, tweet_hashtags, tweet_users_mentioned, user_id, user_created_at, user_name, user_screen_name, user_lang, user_description, user_verified, user_friends_count, user_followers_count, user_statuses_count, user_favourites_count, user_listed_count and user_location.

2. Configuring Solr Backend https://github.com/.../solr-5.3.0/UntitledSolr

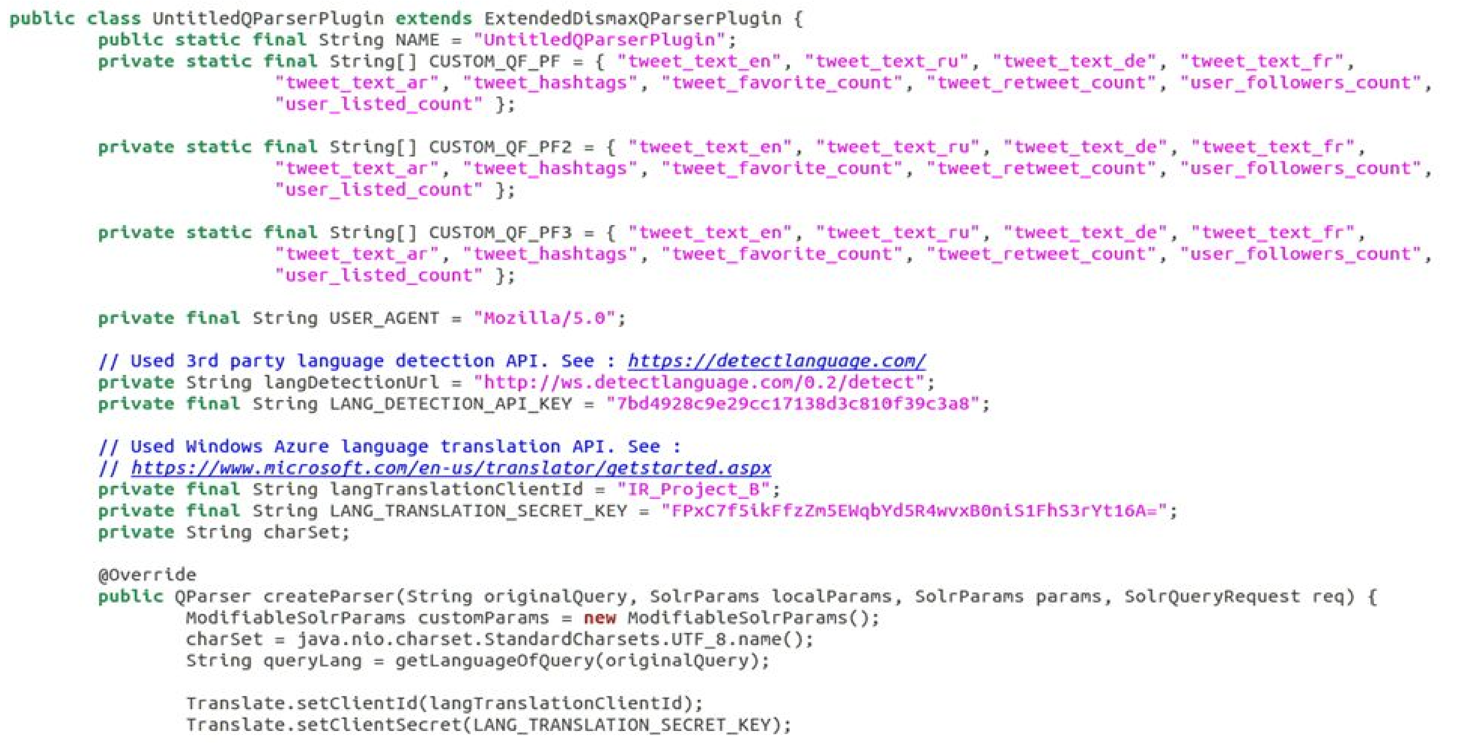
This part is the heart of our Search System. We have implemented 5 components and most them were configured at the backend so as to provide a real solr search functionality to the user.

2.1 Custom EdismaxQParserPlugin https://github.com/.../UntitledSolr/solr/UntitledSolr/lib

For building a **multilingual search** we faced 2 challenges, 1st **detecting the language** of the query and 2nd **translating it to 4 other languages**. Since most of the language translation api’s were paid including the Google and had limit on the number of requests/day, so we decided to go for open source. For the 1st part we used language detection api of **detectlanguage.com** which provides the result in a form of json. For 2nd part we used **Microsoft Azure** language translation api.

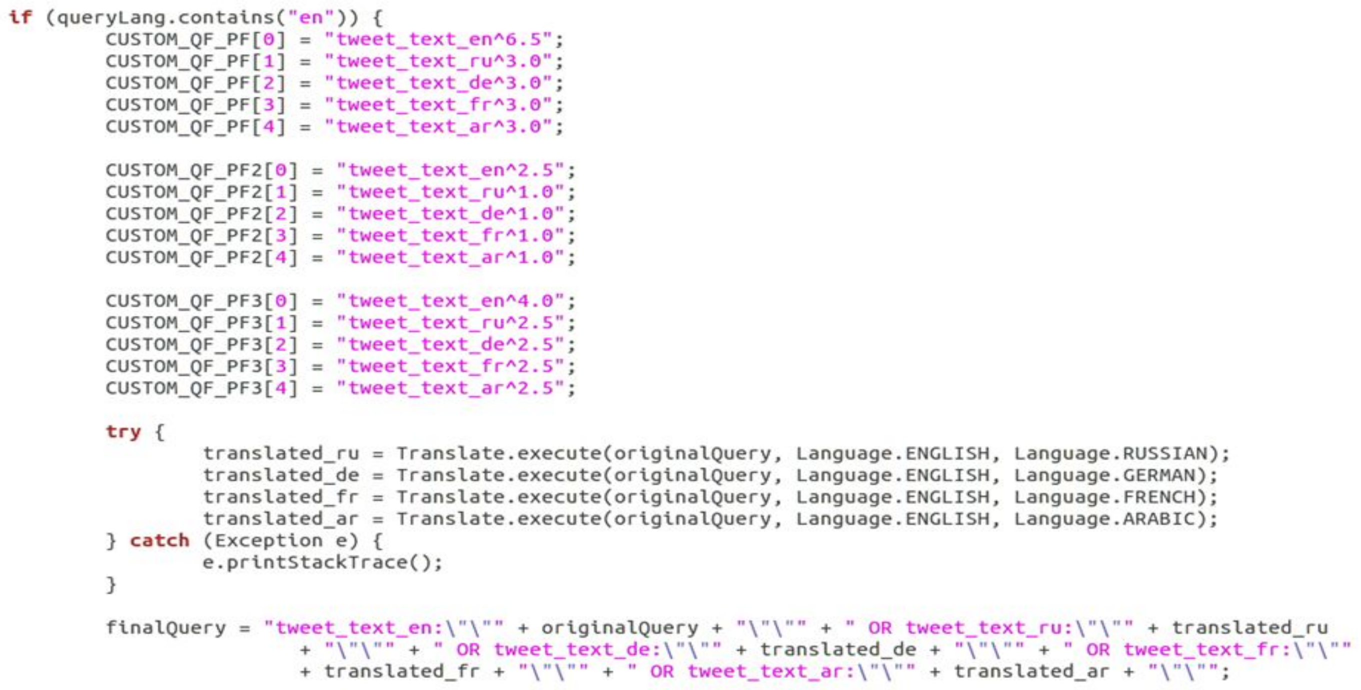
The next challenge was to configure solr such that language detection and translation happens automatically for every query. We extended solr’s inbuilt query parser plugin (ExtendedDismaxQParserPlugin) to provide a smooth integration of our enhanced functionality over the top of solr inbuilt classes. With the custom query parser we also assigned specific weights to each language to be fair with the native language of the query, so that the language in which user typed in the query should get higher score than other languages and get a better chance to appear on the top.

Here is a snippet of our custom query parser plugin-



```java
public class UntitledQParserPlugin extends ExtendedDismaxQParserPlugin {
        public static final String NAME = "UntitledQParserPlugin";
        private static final String[] CUSTOM_QF_PF = { "tweet_text_en", "tweet_text_ru", "tweet_text_de", "tweet_text_fr",
                        "tweet_text_ar", "tweet_hashtags", "tweet_favorite_count", "tweet_retweet_count", "user_followers_count",
                        "user_listed_count" };

        private static final String[] CUSTOM_QF_PF2 = { "tweet_text_en", "tweet_text_ru", "tweet_text_de", "tweet_text_fr",
                        "tweet_text_ar", "tweet_hashtags", "tweet_favorite_count", "tweet_retweet_count", "user_followers_count",
                        "user_listed_count" };

        private static final String[] CUSTOM_QF_PF3 = { "tweet_text_en", "tweet_text_ru", "tweet_text_de", "tweet_text_fr",
                        "tweet_text_ar", "tweet_hashtags", "tweet_favorite_count", "tweet_retweet_count", "user_followers_count",
                        "user_listed_count" };

        private final String USER_AGENT = "Mozilla/5.0";

        // Used 3rd party language detection API. See : https://detectlanguage.com/
        private String langDetectionUrl = "http://ws.detectlanguage.com/0.2/detect";
        private final String LANG_DETECTION_API_KEY = "7bd4928c9e29cc17138d3c810f39c3a8";

        // Used Windows Azure language translation API. See :
        // https://www.microsoft.com/en-us/translator/getstarted.aspx
        private final String langTranslationClientId = "IR_Project_B";
        private final String LANG_TRANSLATION_SECRET_KEY = "FPxC7f5ikFfzZm5EWqbYd5R4wvxB0niS1FhS3rYt16A=";
        private String charSet;

        @Override
        public QParser createParser(String originalQuery, SolrParams localParams, SolrParams params, SolrQueryRequest req) {
                ModifiableSolrParams customParams = new ModifiableSolrParams();
                charSet = java.nio.charset.StandardCharsets.UTF_8.name();
                String queryLang = getLanguageOfQuery(originalQuery);

                Translate.setClientId(langTranslationClientId);
                Translate.setClientSecret(LANG_TRANSLATION_SECRET_KEY);
```

. . . . .



```java
  if (queryLang.contains("en")) {
          CUSTOM_QF_PF[0] = "tweet_text_en^6.5";
          CUSTOM_QF_PF[1] = "tweet_text_ru^3.0";
          CUSTOM_QF_PF[2] = "tweet_text_de^3.0";
          CUSTOM_QF_PF[3] = "tweet_text_fr^3.0";
          CUSTOM_QF_PF[4] = "tweet_text_ar^3.0";

          CUSTOM_QF_PF2[0] = "tweet_text_en^2.5";
          CUSTOM_QF_PF2[1] = "tweet_text_ru^1.0";
          CUSTOM_QF_PF2[2] = "tweet_text_de^1.0";
          CUSTOM_QF_PF2[3] = "tweet_text_fr^1.0";
          CUSTOM_QF_PF2[4] = "tweet_text_ar^1.0";

          CUSTOM_QF_PF3[0] = "tweet_text_en^4.0";
          CUSTOM_QF_PF3[1] = "tweet_text_ru^2.5";
          CUSTOM_QF_PF3[2] = "tweet_text_de^2.5";
          CUSTOM_QF_PF3[3] = "tweet_text_fr^2.5";
          CUSTOM_QF_PF3[4] = "tweet_text_ar^2.5";

          try {
                  translated_ru = Translate.execute(originalQuery, Language.ENGLISH, Language.RUSSIAN);
                  translated_de = Translate.execute(originalQuery, Language.ENGLISH, Language.GERMAN);
                  translated_fr = Translate.execute(originalQuery, Language.ENGLISH, Language.FRENCH);
                  translated_ar = Translate.execute(originalQuery, Language.ENGLISH, Language.ARABIC);
          } catch (Exception e) {
                  e.printStackTrace();
          }

          finalQuery = "tweet_text_en:\"\"" + originalQuery + "\"\"" + " OR tweet_text_ru:\"\"" + translated_ru
                          + "\"\"" + " OR tweet_text_de:\"\"" + translated_de + "\"\"" + " OR tweet_text_fr:\"\""
                          + translated_fr + "\"\"" + " OR tweet_text_ar:\"\"" + translated_ar + "\"\"";
```

2.2 Intelligent Ranking https://github.com/.../UntitledSolr/solr/UntitledSolr/conf/solrconfig.xml

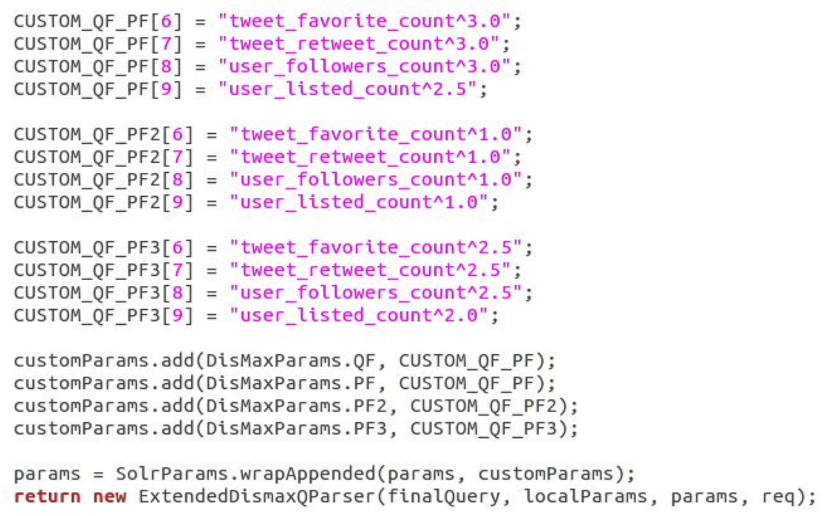
Since tweets couldn't be page ranked, so we tried to enhance our scoring technique based on 2 criteria :-

1. **Boosting documents based on counts:**

   Since the data that we collected for indexing purpose contained lots of counts of various fields we tried to utilize them for building our ranking scheme. We focused on 5 different categories for boosting the scores of documents :-

   a. Boost the score for the tweet with **verified user flag**
   b. Boost the score for the **recent tweet** using bf (boost function)
   c. Boost the score using **retweet count**
   d. Boost the score using **favourite count** of tweet
   e. Boost the score using **followers count** of user who posted the tweet
   f. Boost using the **count of public lists** that the user is a member of

Here is a snippet of our boosting parameters



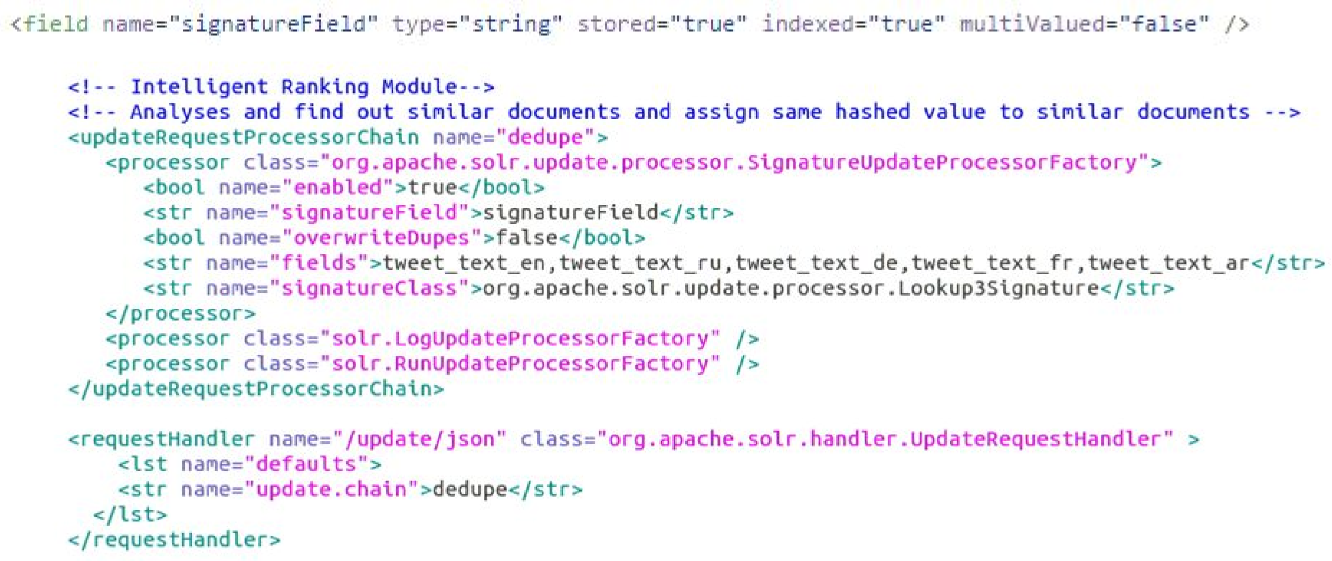
```
CUSTOM_QF_PF[6] = "tweet_favorite_count^3.0";
CUSTOM_QF_PF[7] = "tweet_retweet_count^3.0";
CUSTOM_QF_PF[8] = "user_followers_count^3.0";
CUSTOM_QF_PF[9] = "user_listed_count^2.5";

CUSTOM_QF_PF2[6] = "tweet_favorite_count^1.0";
CUSTOM_QF_PF2[7] = "tweet_retweet_count^1.0";
CUSTOM_QF_PF2[8] = "user_followers_count^1.0";
CUSTOM_QF_PF2[9] = "user_listed_count^1.0";

CUSTOM_QF_PF3[6] = "tweet_favorite_count^2.5";
CUSTOM_QF_PF3[7] = "tweet_retweet_count^2.5";
CUSTOM_QF_PF3[8] = "user_followers_count^2.5";
CUSTOM_QF_PF3[9] = "user_listed_count^2.0";

customParams.add(DisMaxParams.QF, CUSTOM_QF_PF);
customParams.add(DisMaxParams.PF, CUSTOM_QF_PF);
customParams.add(DisMaxParams.PF2, CUSTOM_QF_PF2);
customParams.add(DisMaxParams.PF3, CUSTOM_QF_PF3);

params = SolrParams.wrapAppended(params, customParams);
return new ExtendedDismaxQParser(finalQuery, localParams, params, req);
```

2. **Avoiding similar documents using SignatureUpdateProcessorFactory:**

   Since a tweet can be retweeted many times, so when a query matches with a particular tweet there may be chances that all the top 10 results contain the same tweet text. To avoid this scenario we used Solr SignatureUpdateProcessorFactory, which assigns a unique hash value to the tweet text.

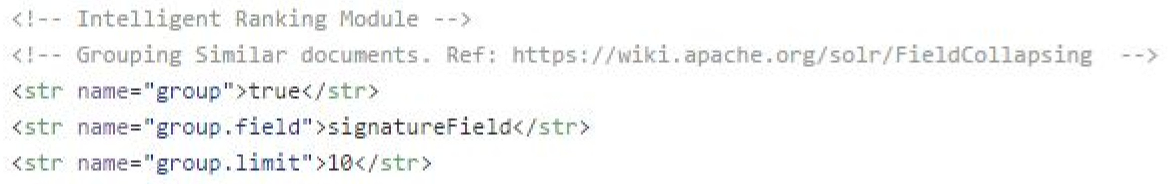
   We then clubbed the documents associated with these hashed values using solr **FieldCollapsing** technique. This helped us to not only analyse similar tweets but also grouped them into a single group to avoid repetition.

Here is a snippet of our SignatureUpdateProcessorFactory code



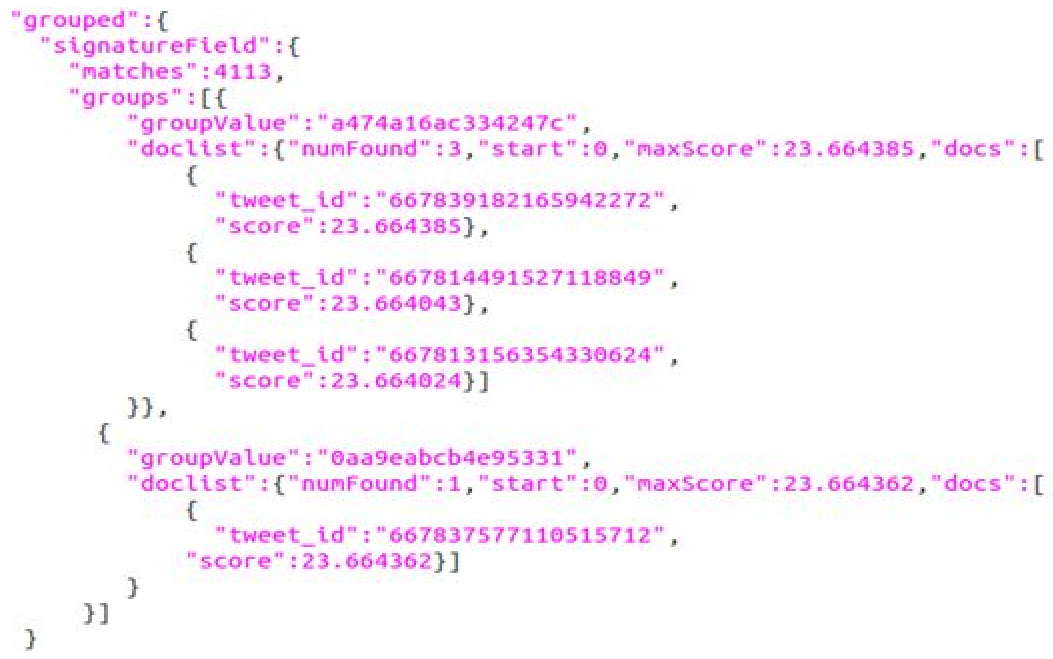
```
<field name="signatureField" type="string" stored="true" indexed="true" multiValued="false" />

    <!-- Intelligent Ranking Module-->
    <!-- Analyses and find out similar documents and assign same hashed value to similar documents -->
    <updateRequestProcessorChain name="dedupe">
      <processor class="org.apache.solr.update.processor.SignatureUpdateProcessorFactory">
        <bool name="enabled">true</bool>
        <str name="signatureField">signatureField</str>
        <bool name="overwriteDupes">false</bool>
        <str name="fields">tweet_text_en,tweet_text_ru,tweet_text_de,tweet_text_fr,tweet_text_ar</str>
        <str name="signatureClass">org.apache.solr.update.processor.Lookup3Signature</str>
      </processor>
      <processor class="solr.LogUpdateProcessorFactory" />
      <processor class="solr.RunUpdateProcessorFactory" />
    </updateRequestProcessorChain>

    <requestHandler name="/update/json" class="org.apache.solr.handler.UpdateRequestHandler" >
        <lst name="defaults">
        <str name="update.chain">dedupe</str>
      </lst>
    </requestHandler>
```

. . .



```
<!-- Intelligent Ranking Module -->
<!-- Grouping Similar documents. Ref: https://wiki.apache.org/solr/FieldCollapsing  -->
<str name="group">true</str>
<str name="group.field">signatureField</str>
<str name="group.limit">10</str>
```

This is how the results look like after grouping



```
"grouped":{
  "signatureField":{
    "matches":4113,
    "groups":[{
        "groupValue":"a474a16ac334247c",
        "doclist":{"numFound":3,"start":0,"maxScore":23.664385,"docs":[
            {
              "tweet_id":"667839182165942272",
              "score":23.664385},
            {
              "tweet_id":"667814491527118849",
              "score":23.664043},
            {
              "tweet_id":"667813156354330624",
              "score":23.664024}]
        }},
      {
        "groupValue":"0aa9eabcb4e95331",
        "doclist":{"numFound":1,"start":0,"maxScore":23.664362,"docs":[
            {
              "tweet_id":"667837577110515712",
            "score":23.664362}]
        }
      }]
 }
```

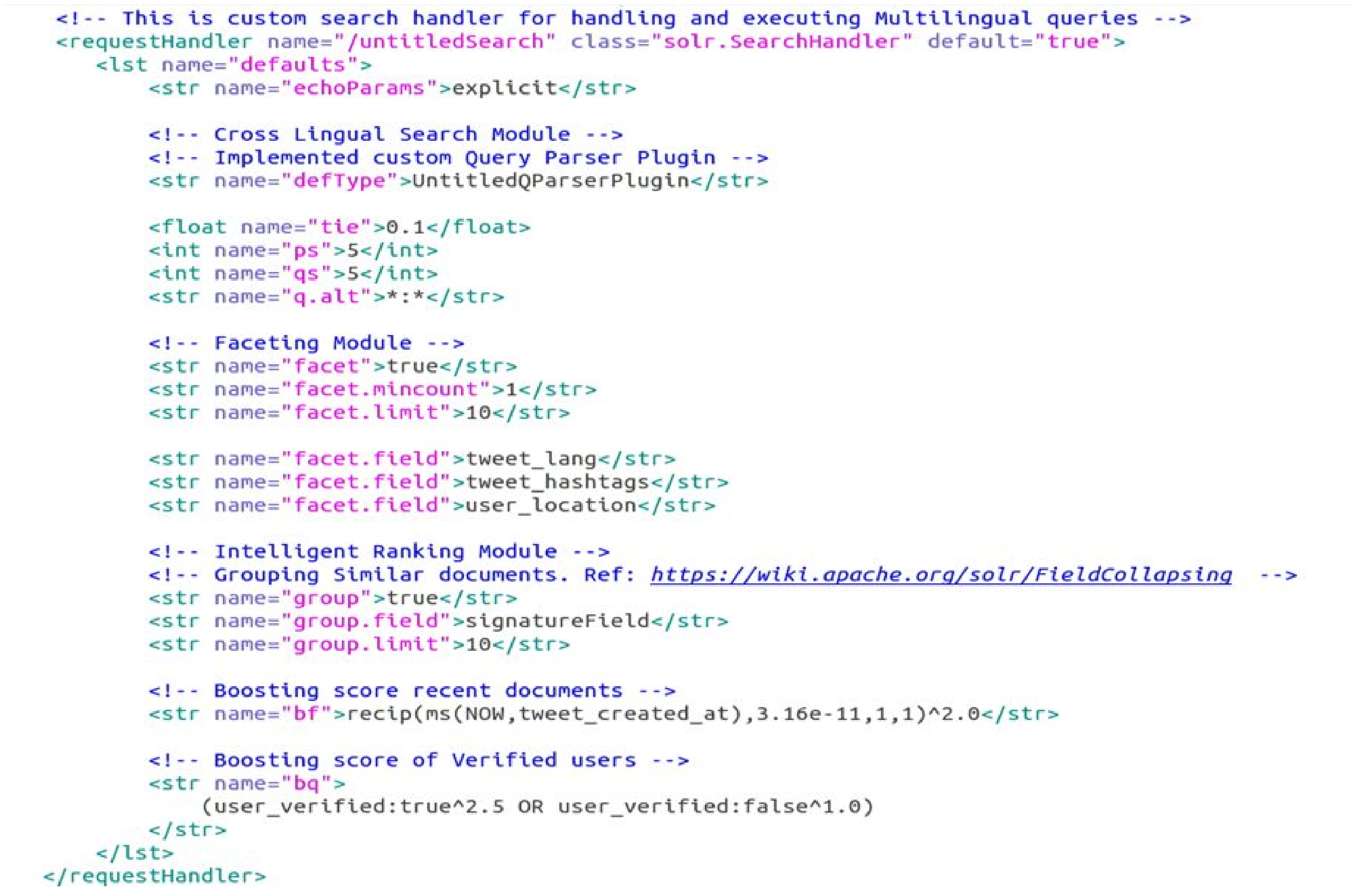
2.3 Faceting https://github.com/.../UntitledSolr/solr/UntitledSolr/conf/solrconfig.xml

Faceting is one of the important factor when we deal with the advance search systems. It help us refine the results by breaking up the search results into multiple categories, typically showing counts for each, and allows the user to "drill down" or further restrict their search results based on those facets. We provided faceting on 4 different categories :-

a. Language of tweet
b. Location of tweet (to search tweet made from specific country)
c. Hashtags present in the tweet
d. Date on which tweet was posted

This help user fetch information based on need, rather than getting all tweets matching the query.

Here is a snippet of our custom Search handler with Faceting



```
<!-- This is custom search handler for handling and executing Multilingual queries -->
<requestHandler name="/untitledSearch" class="solr.SearchHandler" default="true">
   <lst name="defaults">
       <str name="echoParams">explicit</str>

       <!-- Cross Lingual Search Module -->
       <!-- Implemented custom Query Parser Plugin -->
       <str name="defType">UntitledQParserPlugin</str>

       <float name="tie">0.1</float>
       <int name="ps">5</int>
       <int name="qs">5</int>
       <str name="q.alt">*:*</str>

       <!-- Faceting Module -->
       <str name="facet">true</str>
       <str name="facet.mincount">1</str>
       <str name="facet.limit">10</str>

       <str name="facet.field">tweet_lang</str>
       <str name="facet.field">tweet_hashtags</str>
       <str name="facet.field">user_location</str>

       <!-- Intelligent Ranking Module -->
       <!-- Grouping Similar documents. Ref: https://wiki.apache.org/solr/FieldCollapsing  -->
       <str name="group">true</str>
       <str name="group.field">signatureField</str>
       <str name="group.limit">10</str>

       <!-- Boosting score recent documents -->
       <str name="bf">recip(ms(NOW,tweet_created_at),3.16e-11,1,1)^2.0</str>

       <!-- Boosting score of Verified users -->
       <str name="bq">
           (user_verified:true^2.5 OR user_verified:false^1.0)
       </str>
   </lst>
</requestHandler>
```

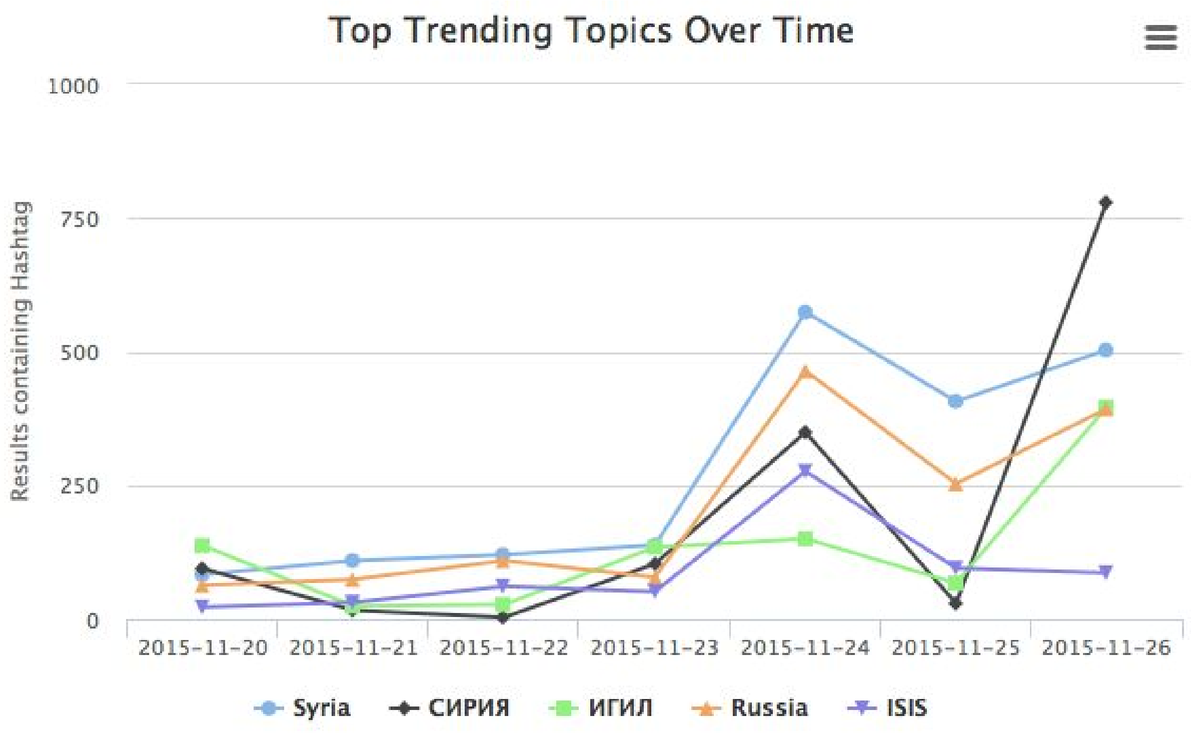
3. Cross-Doc Analytics https://github.com/agsimeonov/cse535-final/tree/master/crossdoc

The cross-document analysis component was completed using highcharts in Python scripts that take as input Solr query results. The scripts then compute their respective analytics and produce a chart as an html document containing the javascript necessary to draw the charts.

3.1 Line Chart Generator https://github.com/agsimeonov/cse535-final/tree/master/crossdoc

There is a single line chart produced per script execution. The script will acquire the set of dates contained in the input and sort them from earliest to latest. It will also count the number of occurrences of each hashtag. After doing this it will pick the top 5, or some other user defined number, of hashtags. It will assign for each date in the sorted date set a count of occurrences per top hashtag. From this we will then generate a line chart displaying the top hashtags (topics) and how they varied across the different dates.

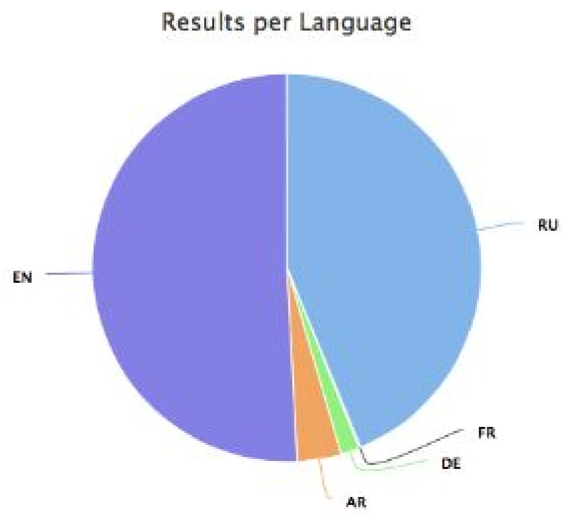
Here is an example output

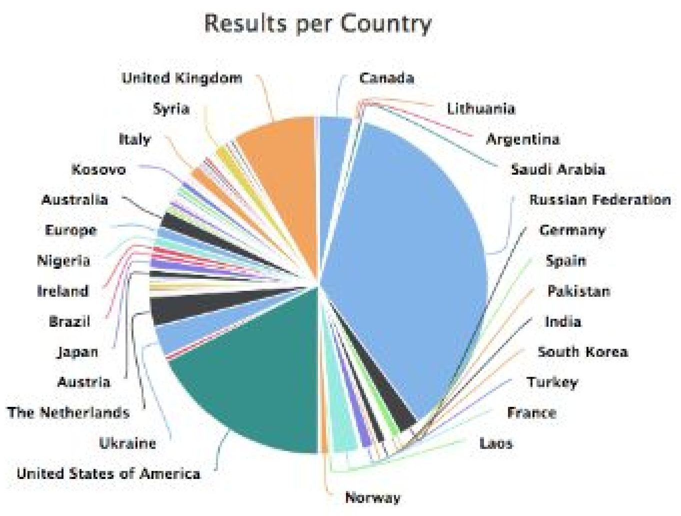


3.2 Pie Chart Generator https://github.com/agsimeonov/cse535-final/blob/master/crossdoc/pie.py

The pie chart generator works in a similar way to the line chart generator, however it generates two different pie charts. The first pie chart it generates counts the number of tweets per language in the provided Solr query results. The second pie chart does a similar thing but instead of counting the number of tweets per language, it counts the number of tweets per country designation, a designation available due to the geocoder as described previously. Both pie charts are output in separate files.

Here is an example of both pie charts



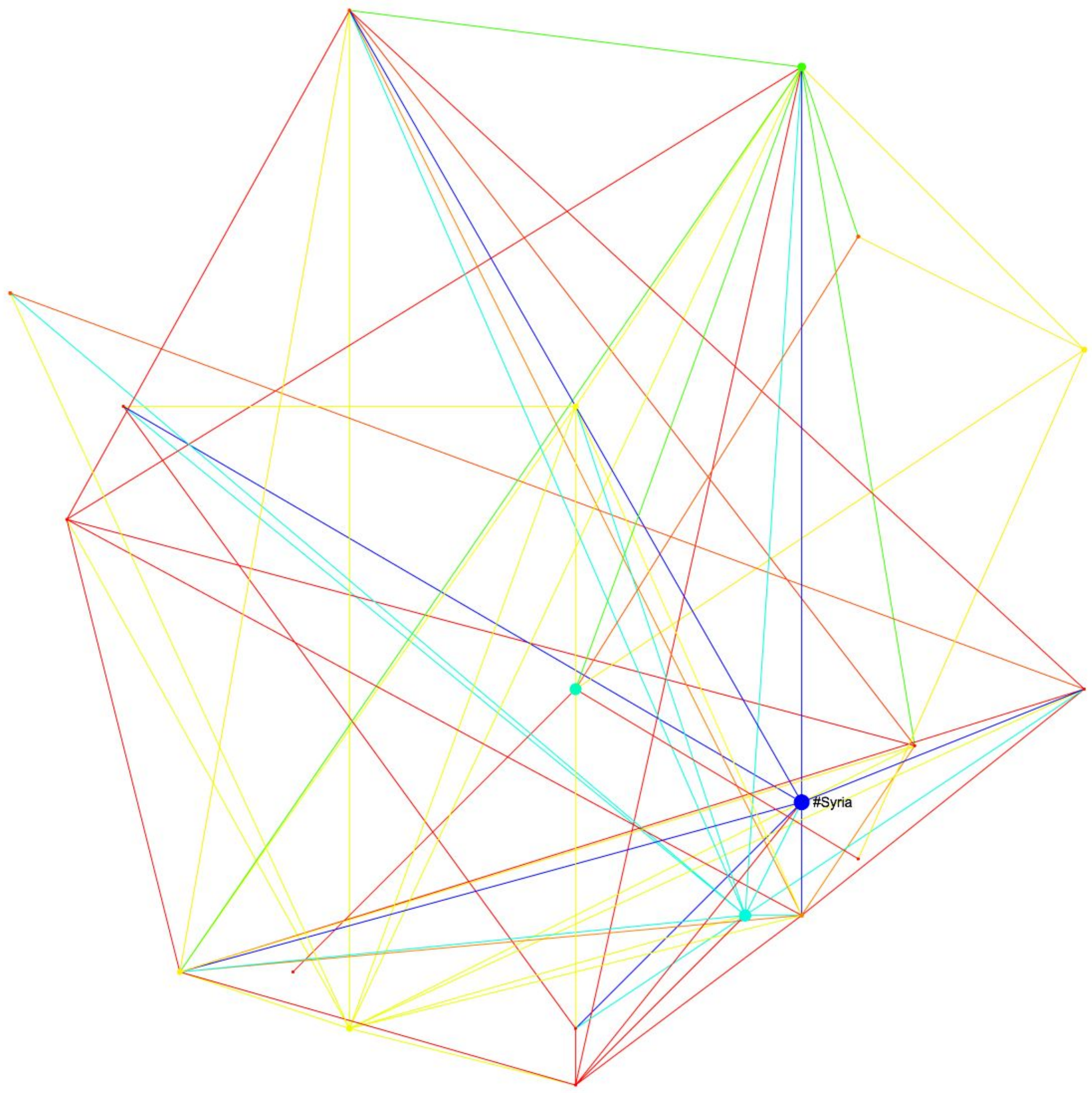


4. Graphical Analysis https://github.com/agsimeonov/cse535-final/tree/master/graph

Graphical Analysis works in a way that is very similar to Cross-Document Analytics. We have a script that takes in JSON Solr query results as input and produces a graph. We decided to graph hashtag co-occurrence. This would display how different topics relate to one another. Since the result graph can get fairly busy we limit the results to only the top 20 hashtags. This still produces a good meaningful graph. The tools we use are once again Python, and sigma.js which is a great library used for graph creation. Our graph is interactive, one can move around and look at what is inside each node. As you zoom in node descriptions (hashtags) would appear.

To distinguish high value nodes we make them bigger depending on the number of occurrences of their respective hashtag. We also color nodes on the RGB spectrum such that more important topics are blue and less important ones red. In our graph nodes represent hashtags (topics) and edges represent the fact that there is a co-occurrence relationship between them. Sigma.js doesn't support edge weights but if it did edge weights would have represented the number of co-occurrences.

Here is an example graph



5. Analytics Script Server https://github.com/agsimeonov/cse535-final/tree/master/pyserv

We designed our analytics in a way that it would be feasible to incorporate them within our website in a modular way. The easiest way a website can interact with them would be through some sort of a CGI server. We implemented a server that accepts the JSON Solr Query Output as the data element in a POST request. It would then run our scripts in a separate thread to produce the charts and graph. The server also accepts GET requests so that the graphs can be displayed and embedded without our system.

The script server makes sure the input format is correct. If the format is in the form of groups it would extract the tweets and format them into the more expected standard query result format which is required in the scripts. Once that is done it would save the JSON results to a file which is used as input to each script. Scripts are ran as separate threads so that charts and graphs can be produced in a quick and robust manner.

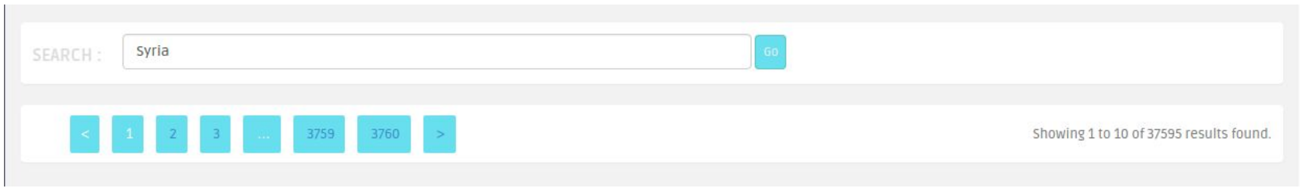
6. Untitled UI

All the features mentioned above are executed through the Untitled UI Component. We have a designed a un-cluttered user-interface to enable smooth searching functionality with intuitive design and some advanced features to get the best possible search experience. Untitled presently indexes the recent tweets on some of the most trending topics globally and run analytics on these information to get useful insights into what is not explicit in the tweets in their original from. The Analytics section classifies the results based on the Hashtag Co-occurrence , Distribution by Language, Country and topics.The search also integrates part if your twitter search environment as part of the application and you have access to all the tweets and notifications from Untitled. In order to fetch information from you twitter handle, untitled requires you to sign-in using Twitter, and give the application the rights to some of your twitter information using oauth.

Some of the major components of the search system are:

6.1 Search

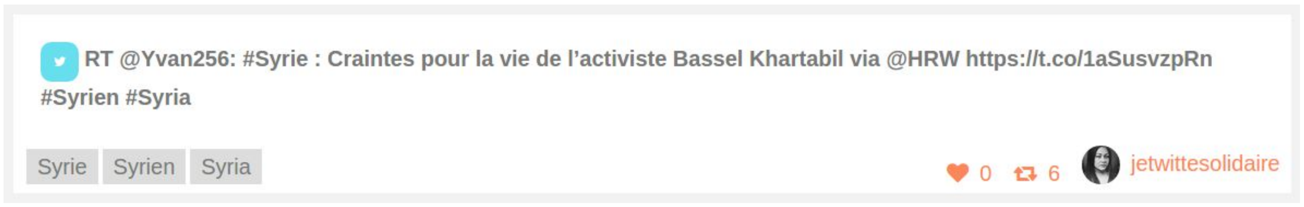
This refers to the main search box that enables a user to search for a query.Since we are using solr to index the information , the query supported is a full-text search and solr handles the query in its original form and retrieves the results instantly. Since Untitled is a multilingual search system, based on the query entered by the user, the search results may be in any of the five languages that is currently supported by untilted namely en, de, ar, ru and fr. Since the results retrieved are in large numbers we have implemented a results pager to organize results into a discrete paging size, and navigate to pages interactively from the paging panel. Along with the paging panel, to the right of the pager you will find the total amounts of results found against the query which is used for the cross-document analysis.



Showing 1 to 10 of 37595 results found.

6.2 Results

The results are populated based on the search query in the region below the search panel which is scrollable and retrieves all the information related to the present page. The results are displaying based on intelligent ranking and other ranking constraints that were applied into the twitter corpus that we have collected. Every result that is retrieved, containstetweet_text, retweet_count, user_name, favorited, hashtags and many more. The twitter icon, before the tweeted text lands you into the twitter where this post was originally from. The hashtags from these results are used to build the co-occurrence matrix and largely into faceting which will be discussed in the next section.



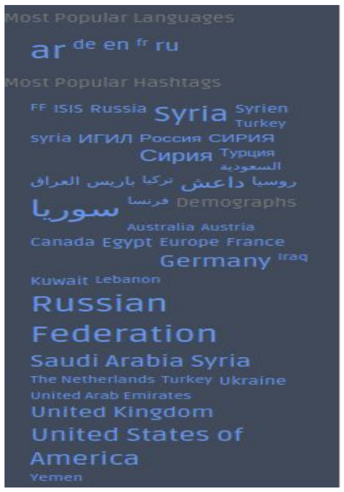
RT @Yvan256: #Syrie : Craintes pour la vie de l'activiste Bassel Khartabil via @HRW https://t.co/1aSusvzpRn #Syrien #Syria

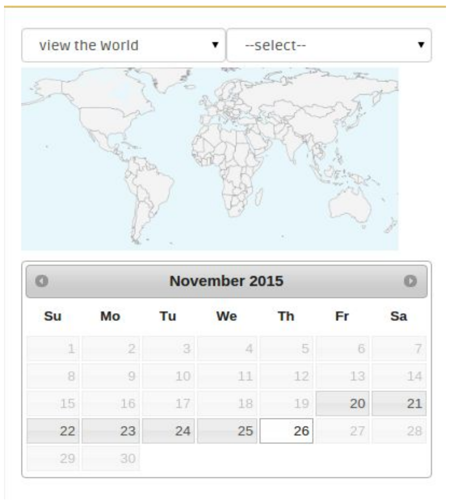
Syrie Syrien Syria 0 6 jetwittesolidaire

6.3 Faceting

Faceting is one of the most important feature of the untitled search system and lead to classified search experience. We have implemented faceting based on language, country, hashtags and time. We have implemented various UI components to reflect the above facets like Tag Clouds, World Map and Calendar to augment your search results with these additional selections and get a more specific information about what you are searching for. Without faceting it would be fairly complex and cumbersome to get access to all the information that you originally wanted but faceting making it simple to retrieve more specific information.

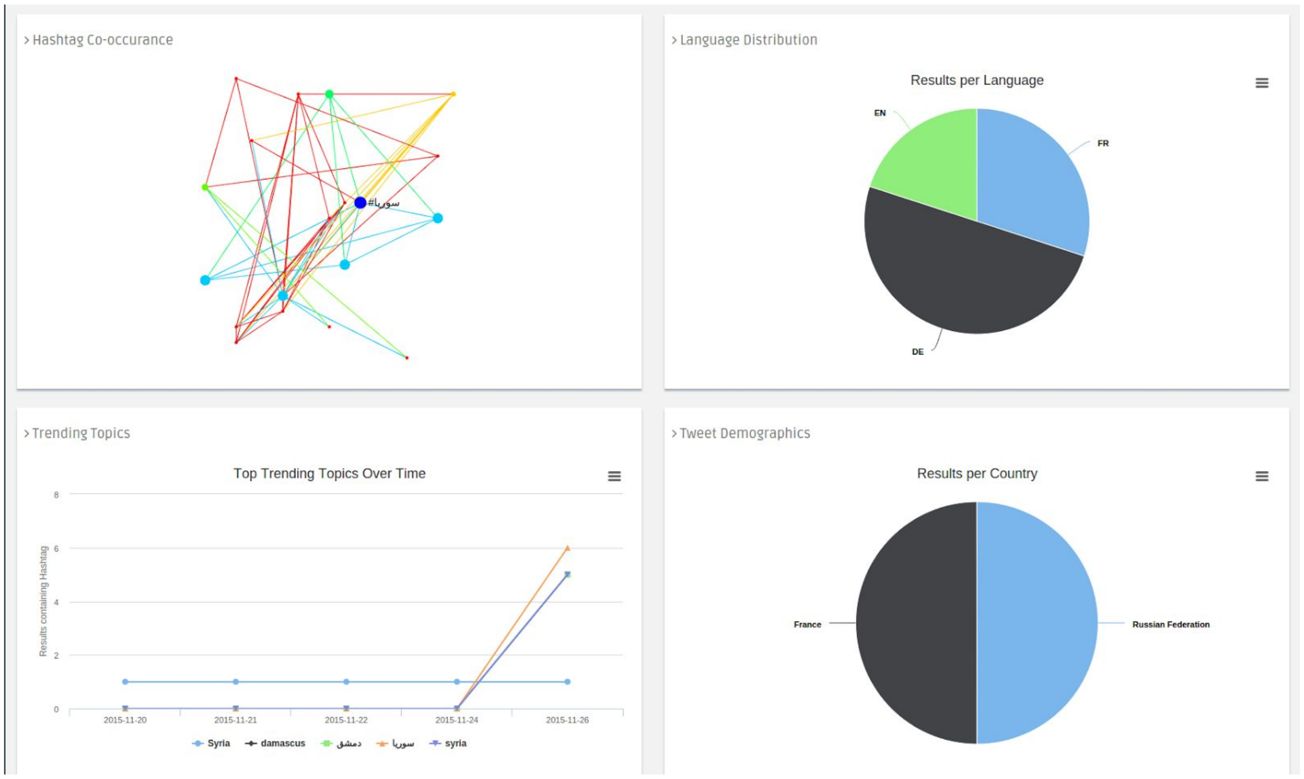
Tag Cloud World Map and Calendar



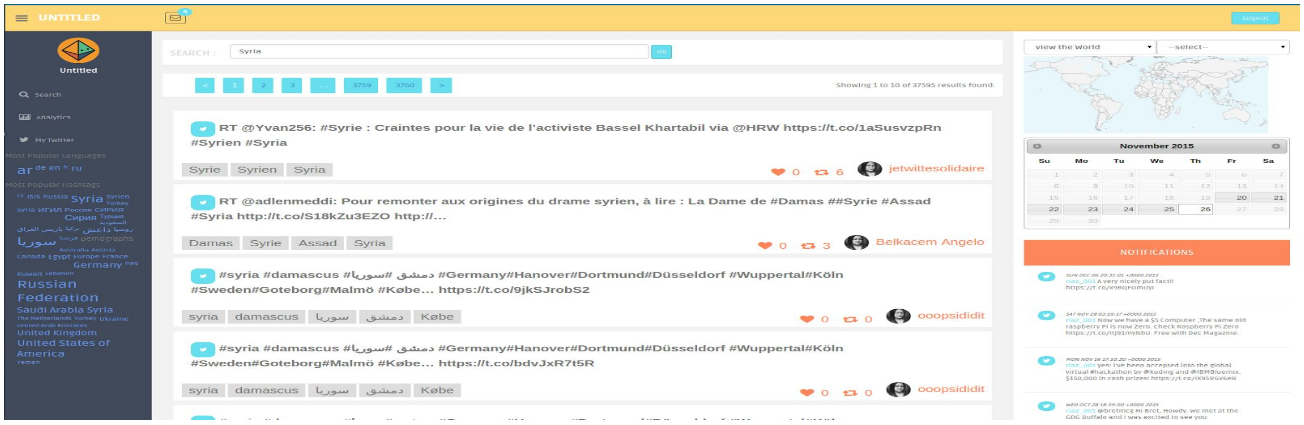


6.4 Analytics

You can run and visualize the analytics on the query results by hitting the analytics tab in the Untitled. The Analytics run analytics on present query results and was well as results based on cross-document analytics. The analytics are based on Hashtag Co-occurrence, Language Distribution, Trending Topics and Tweet Demographics. The chats are based on the highcharts library and can be exported to various formats and downloaded. The analytics are important to get a broader view of the search results and further information not easily perceived from plain text query results.



6.5 Untitled Search



UNTITLED
Untitled
Search
Analytics
My Twitter
Most Popular Languages
ar de en fr ru
Most Popular Hashtags
FF ISIS Russia Syria Syrien Turkey syria ИГИЛ Россия СИРИЯ Сирия Турция السعودية العراق باريس تركيا داعش روسيا سوريا فرنسا Demographs
Australia Austria Canada Egypt Europe France Germany Iraq Kuwait Lebanon Russian Federation Saudi Arabia Syria The Netherlands Turkey Ukraine United Arab Emirates United Kingdom United States of America Yemen
SEARCH :
Syria
Go
< 1 2 3 ... 3759 3760 >
Logout
view the world
--select--
November 2015
Su Mo Tu We Th Fr Sa
NOTIFICATIONS

Showing 1 to 10 of 37595 results found.

RT @Yvan256: #Syrie : Craintes pour la vie de l'activiste Bassel Khartabil via @HRW https://t.co/1aSusvzpRn #Syrien #Syria
Syrie | Syrien | Syria — 0 | 6 — jetwittesolidaire

RT @adlenmeddi: Pour remonter aux origines du drame syrien, à lire : La Dame de #Damas ##Syrie #Assad #Syria http://t.co/S18kZu3EZO http://...
Damas | Syrie | Assad | Syria — 0 | 3 — Belkacem Angelo

#syria #damascus #سوريا# دمشق #Germany#Hanover#Dortmund#Düsseldorf #Wuppertal#Köln #Sweden#Goteborg#Malmö #Købe... https://t.co/9jkSJrobS2
syria | damascus | سوريا | دمشق | Købe — 0 | 0 — ooopsididit

#syria #damascus #سوريا# دمشق #Germany#Hanover#Dortmund#Düsseldorf #Wuppertal#Köln #Sweden#Goteborg#Malmö #Købe... https://t.co/bdvJxR7t5R
syria | damascus | سوريا | دمشق | Købe — 0 | 0 — ooopsididit

SUN DEC 06 20:31:01 +0000 2015
riaz_001 A very nicely put fact!! https://t.co/x9BQFOmUyI

SAT NOV 28 03:19:17 +0000 2015
riaz_001 Now we have a $5 Computer ,The same old raspberry Pi is now Zero. Check Raspberry Pi Zero https://t.co/itj91myNbU. Free with Dec Magazine.

MON NOV 16 17:50:20 +0000 2015
riaz_001 yes! i've been accepted into the global virtual #hackathon by @koding and @IBMBluemix. $150,000 in cash prizes! https://t.co/IX95RGV6eR

WED OCT 28 18:59:00 +0000 2015
riaz_001 @bretmcg Hi Bret, Howdy. We met at the GDG Buffalo and I was excited to see you