UNIVERSITAT POLITÈCNICA DE CATALUNYA

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA DE TELECOMUNICACIÓ DE BARCELONA

MASTER OF SCIENCE IN INFORMATION AND COMMUNICATION TECHNOLOGIES (MINT)

Master Thesis:

"Twitter Weighting System"

Student: Gabriel Pollner

July 2012

Contents

Motivation

During the last decade the amount and complexity of information flowing through the World Wide Web has been growing exponentially. New websites, blogs, news feeders, video and image sharing portals and social networks had born and been active for a while and each day more Internet users subscribe to one or more of these online services.

Each one of these services was created with one or more purposes, such as multimedia sharing, exchange of personal information between friends or just content distribution. After some years of activity developers of the main social networks in the industry realized that their product was useful for many more purposes than the ones though in the original idea. For example this was the case of Facebook which was initially conceived for college students as a way to share their college life with other students among the different universities in the United States. Today Facebook is not only being used by millions of people around the world but it is also being used by companies to promote their products or services and also for events organization. This kind of evolution has been observed in many other social networks and it is still going on.

The main objective of this project is to develop an initial idea for an improvement of the use of Twitter as a source of content. Taking into account that major software developers count with multidisciplinary teams and even with that advantage, they take at least one or two year to launch new applications, Artic Group, as a startup company, with only two people working in this project, has no greater ambition than to achieve, in one semester, a prototype which later will require more time and more hands to finally become releasable to the market.

Artic Group, has already developed some Twitter related websites, peoplenews.me and sports.peoplenews.me for example present tweet messages from a selection of Twitter users (authors), using a “news feeder” format. This means that all the content extracted from the different tweets is presented in form of newspaper or magazine articles.

Using the processes behind those two websites as a starting point, we want to begin our development of the new application thinking straight to our main concern which is to create a website similar to the previous ones, but this time with a customized selection of authors. Furthermore we want the selections of authors to stay automatically up to date over time.

Spending just half an hour reading tweets messages from different people in Twitter, we can observe two main different types of use to this social network. Some users just talk about their life and share personal thoughts with friends or fans, among these users are common individuals, famous actors, international athletes, etc. Meanwhile other Twitter users like to post messages related with a particular sport, hobby, recreational or intellectual activity. This last type of user is commonly associated with magazines, newspapers or journalists who really see Twitter as another communication medium like radio or television.

The second type of user previously mentioned, is clearly the one we are looking for to become part of any customized selection of authors related to a particular topic. The reason is because these authors are more likely to seriously stick to one topic and always publish information of interest.

In order to differentiate the Twitter users that will fulfill our interests from the other ones, an evaluation mechanism applied to each user should be done. For example looking the number of followers, the frequency of posting, and the most commonly mentioned words would be an ideal way to see whether a Twitter user is a good candidate for our selection of authors or not. As a measurable result of this evaluation mechanism we would establish a "weight" for each author. The weight will tell us how much is that author related with the chosen topic of interest.

As our main goal for the end of the semester is to have a just prototype, we are going to focus our work in this evaluation mechanism, leaving the addition of new authors for future phases of the project. For this reason we have called this phase "Twitter Weighting System".

State of The Art

From the very beginning of this project, it was clear that different programming tools and APIs would be required to add any kind of Intelligence to the Twitter Search Engine. Starting from using previous programming knowledge to learn the basic syntax of the PHP programming language, followed by getting into some more complex APIs like the one from Twitter, and finally understanding the use of some Semantic Analysis Applications like Zemanta and FreeLing, became part of the time invested in the project. In the following lines, ordered by the time of appearance during the development process, different States of The Art related to each one of these tools are presented.

PHP

The PHP (Hypertext Preprocessor) as described in [1] is a programming language is a non-compiled programming language. It is widely-used combined with HTML to develop Web sites or web-based Internet Applications. It has an easy to use syntax and most of the web developer tools found on the Internet have an API specifically built for PHP. Besides all of the advantages PHP provides, one of its main disadvantages is the very low capacity for highly intensive processing of data in comparison to more powerful (but more complex) programming languages like C++, Python or Ruby.

Drupal

Nowadays due to the complexity of modern websites and Internet Application, a programming language and a database are not enough to get the job done. A content management platform like Drupal is necessary to get a faster development, deal with more data and minimize the number of code lines. According to [2] Drupal is an Open Source platform which provides to the developer a graphical interface over the actual developing website. Drupal allows to create and add new structures, new contents, new appearances as well as other useful elements to the website that otherwise would be harder to create using just

PHP. Another special feature about Drupal is that it provides a large set of useful PHP functions that commonly do something similar to an existing PHP function but with some improvements like debugging properties. Finally once the web application is working and probably in the production phase, thanks to all the monitoring and report generating mechanism it provides, Drupal would be the main source of information to quickly troubleshoot any found issue or bug.

MySQL and PHPMyAdmin

MySQL is an Open-Source relational database management system. As it is mentioned in [3] it provides access to database access for multiple users. In order to easily access MySQL using a web browser, a tool called PHPMyAdmin can be used. As [4] explains, this tool allows the administrator to create, modify and delete databases, tables, rows and fields. It is also possible to use SQL statements from a command line shell or from any programming language to operate with MySQL.

JSON Format

JSON stands for JavaScript Object Notation. According to [5] it is a commonly used format to interchange human-readable data because it is a lightweight text-based open standard. The main reason to use JSON file format is because database tables can’t store array or object structures directly so they must be encoded into text files such as JSON. Fortunately current programming languages come with predefined functions to encode arrays and objects into JSON format and decode JSON files into objects so their data can be retrieved.

Twitter API

The acronym "API" stands for "Application Programming Interface". An API is a defined set of methods for retrieving or modifying data in order to accomplish a task. In Twitter's case the API allows programmers to develop projects with the capacity to interact with Twitter. Using the HTTP protocol any application can talk to the Twitter API, just as a web browser does it when a web page is visited.

Basically in order to interact with Twitter a programmer must add a POST or a GET method (from HTTP protocol) with the right URL into the application source code. The Twitter API [6] defines how this URL must be defined and what parameters to consider depending on the type of request. For example, if the application wants to retrieve the latest tweets from a particular twitter user, the URL must be set in a GET method and parameters to consider are the user's name (E.g. @username) and the Identification Number from the last tweet retrieved from that user. Here is how this request should look like:

GET https://api.twitter.com/1/statuses/user_timeline.json?include_entities=true&include_rts=true&screen_name=username&since_id=12345

The response to this type of requests would be a file containing a set of data fields which associated with each tweet from the user. Each data field would be repeated as many times as tweets get retrieved. There are 4 possible file formats to this response: JSON, XML, RSS and ATOM. In the case of the previous example the file is requested in JSON format.

In the following table there are presented some of the fields that come with each tweet inside the JSON file:

Field	Description	Optional
Title	The tweet message itself	No
User_name	Username of the author of the tweet	No
User_ID	Identification Number for the author of the tweet	No
Language	Tweet's language	No
Tweet_ID	Identification number for the tweet	No
Created	Timestamp of the creation of the tweet	No
URL	URL address included in the tweet.	Yes
URL_title	Title of the destination website from the URL	Yes
Body	Body of the destination website from the URL	Yes

Table 1. Tweet Data Fields

As we can observe, some of the values are optional, because not all the tweets will use all the fields, for example a tweet without an URL link will leave the URL, URL_title and Body fields empty.

Zemanta

Zemanta [7] is an open-source application that analyzes a text and returns some keywords, images and related articles found on the web. As its free version is an online version, the only way to integrate to an external application would require calling the Zemanta's URL from that application. This is very inconvenient for an intensive data processing and for that reason Zemanta was discarded after the performing some tests during the preliminary developing phase.

FreeLing

FreeLing is an open-source language analysis tool. Its main feature is that it can analyze a text and give a full semantic analysis in return.

The result from the FreeLing processing comes as a morphological analysis of Text. Each word, compound word, punctuation mark, number, date, physical magnitude, currency or ratio inside the text is treated as an individual element. Line by line the results show the morphological values of each element using the following format:

Element lemma1 tag1 probability1 lemma2 tag2 probability2 ... lemmaN tagN probabilityN

Depending on what is the element, it can have one or many sets of lemma, tag and probability. This can be easier to understand observing the following example:

Given the following text:

"John Smith likes to write on Twitter every once a week. He is very interested in technology and classical music."

The FreeLing morphological analysis result would be:

```
John_Smith john_smith NP 1
likes like VBZ 1
to to TO 0.999991 to IN 9.09819e-06
write write VB 0.936047 write VBP 0.0639535
on on IN 0.971769
Twitter twitter NP 1
every every DT 1
once once RB 0.754582 once IN 0.245418
a 1 Z 0.99998 a DT 1.01633e-05 a NN 1.01633e-05
week week NN 1
. . Fp 1
He he PRP 1
is be VBZ 1
very very RB 0.954545 very JJ 0.0454545
interested interest VBN 0.96875 interest VBD 0.03125
in in IN 0.986606 in RP 0.0102221 in RB 0.00317237
technology techonology NN 1
and and CC 1
classical classical JJ 1
music music NN
. . Fp 1
```

From the previous example we can realize that whenever FreeLing finds an ambiguous element it shows all its possible meanings but gives a higher probability to the one that fits more in the context. An example of this can be seen with the word “once” which is give two sets of lemma, tag and probability. The first one “once RB 0.754582” suggest that the word “once” works as an adverb with a probability of 0.754582 while the second set “once IN 0.245418” gives the preposition meaning of “once” a probability of 0.245418. The highest probability set will always come closer to the element.

More information about FreeLing can be found in [8], [9], [10] and [11].

Regular Expression

One of the most powerful means to detect a particular character, word or pattern of characters inside a string of text, is the Regular Expression. Also known

as Regex and Regexp, this tool can be implemented in most programming languages. The only requirements are:

1) Know what to look for.
2) Build the regular expression.
3) A string matching function. (Predefined in most programming languages)

Features:

A regular expression is built as a string which is going to work as wildcard for matching the desired pattern. The most common features of this string are:

- It starts and ends with the "\" character.
- The patter can have specific characters like "a", "4" or "+" or even a range of numbers or letters like "[0-9]" or "[a-z]".
- It is also possible to indicate a minimum and a maximum length of the pattern using for example the expression "{8-12}" which means that the pattern must have between 8 and 12 characters.

More information regarding Regular Expressions can be found in [12].

The Problem

What is Twitter missing?

Recommendations:

Even before reading an article like [13] in which Twitter's CEO Dick Costolo admitted that Twitter still needs some improvements, this was something very clear for any occasional user. When a person logs into its Twitter account via web, apart from being able to check all the latest tweets from all following users, he or she can get recommendations about other Twitter users that might be of his/her interest. Although some of these recommendations will certainly be taken into account, most of them are not really what the user is looking for.

The recommendations given by Twitter are based on the current list of following users. This means for example that if John is following his friend Julia he will get among his recommendations some of the people Julia is following. Certainly if Julia only follows people related with the fashion world and John is mostly interested in Football, it is very likely that John would not be interested in any of the recommendations based on Julia's interests.

Having an Up-to-date source of information:

Nowadays, most Twitter users aren't just using Twitter to socialize with friends. They already have Facebook for that purpose. Instead they want to use Twitter to get updated information about their topics of interest. The problem here is that each user must build manually his list of authors to follow, adding one by one the names he knows and getting some others with the Twitter Search Engine. So in the end it takes quite a lot of time to have a decent list of Twitter authors related to a particular topic. Furthermore as times goes by, more authors related to that topic could appear while some of the older ones could stop posting or change their interests to another topic. This means that original authors list could get out of date and therefore stop being a good source of information to the user.

In order to have an up to date authors list, a smart application would be required. Smart because it should include enough intelligence to detect for example: an author from the list who has stopped making posts related to the original topic of interest or a new author has started writing about that topic. And therefore the list would get updated as the time goes by.

An idea for a smart application that would improve the Twitter experience would contain the following features:

1) Recommendations based on keywords instead of what other users are following.
2) Updated list of keywords related to a particular topic.
3) Updated list of authors related to a particular topic.
4) New keywords added by detecting them to be continuously repeated in different Tweets.
5) New authors added when top keywords are mentioned.

Solution Development

Taking into account that our main objective is to develop an initial idea for an improvement of the use of Twitter as a source of content, we are going to present a general approach with our proposal of a flexible solution, designed to be improved in the future.

The first idea that we are going to define regarding our solution is how the user would tell his topics of interest. In general a topic is a very abstract concept that could include different amount of keywords, concepts and contexts. Some topics are very generic while others are more specific. Since it would be very hard to classify each topic and define exactly which keywords, concepts and contexts belong to each one of them, we must think of a simplified way for the user to say what his interests are.

In order to indicate his topics of interest the user of our proposed solution would have to specify some keywords and Twitter authors related with these topics. As the application starts running a sequence of processes would begin and then it would be repeated periodically while the application remains active. During the different processes of this sequence the application will retrieve the latest tweets from the chosen Twitter authors and extract new keywords from those tweets, it will then calculate a weight for each keyword and author and finally add the new keywords and authors to the ones the user defined manually.

The Figure 1 shows the flowchart representing the global work flow of the application prototype.

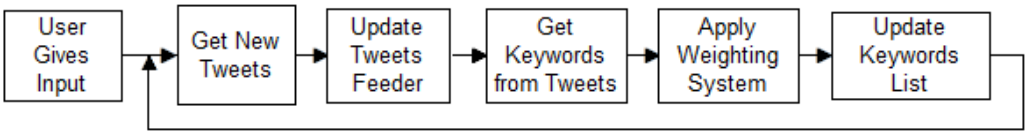

Figure 1. Global Workflow

As we can observe, the user should only give an input once, while all the next steps are sequentially repeated with the minimum user's intervention. This means that the user is allowed to add or remove keywords and authors manually anytime he wants.

User Managed Information

As it can be understand from the previous paragraphs, the user must have an idea of his topics of interests and then look for some keywords and authors related with those topics. Once having this if the user would happen to make use of our proposed application for the first time, he would have to create and account and once he is registered and logged in, he will find this two options:

- Add Keyword
- Add Author

The idea behind offering these two options is that user could start running the application by adding just keywords, just authors or a combination of both. Even though this is a free choice, it is encouraged that the user adds a decent number of keywords and authors (five of each for example). This way the processes would have more information to work with and therefore return more accurate results.

Choosing the "Add keyword" option will open an online form to let the user introduce the data of the new keyword. As some calculations are going to be performed during the different processes, we considered that some additional data must be introduced with each keyword. The Table 2 presents all the data fields to be typed in with every new keyword. These data fields and their default values were chosen following a very personal criterion and therefore they can be modified or removed, also other data fields could be added in the future to make more accurate measures.

Data Field	Description
Keywords	I can be one or multiple words.
Type	Proper noun, common noun, adjective or verb.
Priority	High, medium or low.
Weight	This field cannot be changed, by default it's a 5 and it will be automatically updated.

Table 2. Keyword Data Fields

Choosing the "Add Author" option will open a similar online form; this time will be for introducing the data of the new author. The Table 3 presents all the data fields associated with each new author. Again, there is flexibility in the definition of the additional data fields and default values.

Data Field	Description
Author	It must be a valid Twitter username and it must start with the "@" character.
Priority	High, medium or low.
Weight	This field cannot be changed, by default it's a 5 and it will be automatically updated.

Table 3. Author Data Field

As we observed in tables 2 and 3, both keywords and authors have a data field called "Weight". The value of this field will be automatically updated depending on results given by the Weighting processes. Additionally there are some other relevant values which are not going to be shown to the user and yet will have influence over the weight value. These values were also defined with personal criteria and therefore should be considered as untouchable. The idea behind the selection of these values is that each time a currently listed keyword is extracted from a Tweet, some of these values are going to get updated in the Weighting process and then the user will note the change in the table of keywords presented later in the application's user interface.

On the Table 4 are exposed the keyword proposed internal values.

Value	Description
Keyword ID	Identification number of the keyword.
Frequency	The number of appearances of the keyword since it was first added to the database.
Latency	The time in seconds between the last two appearances of the keyword.
Status	1 = “Published” and 0 = “Not Published” where published means that user can see it in the keywords list. By default all manually added keywords have status 1.

Table 4. Keyword Internal Values

Table 5 shows the author proposed internal values.

Value	Description
Author ID	Identification number of the author.
keywords	A list of the 30 keywords mentioned by the author. Each of them with its internal values and current weight.
Followers	The number of followers the author has on Twitter.
Status	1 = “Published” and 0 = “Not Published” where published means that user can see it in the keywords list. By default all manually added author have status 1.
Last Tweet ID	Identification number of the last tweet retrieved from Twitter.

Table 5. Author Internal Values

Once the user had introduced all the keywords and authors he wants to define his topics of interest, the application will start running the sequence of processes as it is showed in the Figure 1, and after a certain amount of time the first tweets will start to appear (Depending whether the chosen authors have twitted recently or not). In the current prototype the incoming tweets will be from the authors currently included in the list defined by the user. In a more advanced version of the application it would be expected that new authors would be

automatically added, and as the time goes by so will appear the tweets from those new authors.

On the following lines each one of the processes involved in the proposed solution are explained individually starting with the gathering of new tweets, continuing with the extraction of keywords, then weighting of the keywords and the authors, and finally the updating of the keywords and authors lists.

Processes

Get New Tweets

Following the methods described by the Twitter API (check State of The Art section) it is possible to retrieve tweets from particular authors applying certain filters. The proposed solution is to retrieve as many tweets as possible from each author on each request; therefore we must take into account the limitations given by the Twitter API regarding the number of tweets per author per hour.

The Twitter API offers many filters to apply on each request so that we get specifically the tweets we are looking for. In our case, we just want all the latest tweets from each author. To comply with the API's restrictions we established a filter of 15 in the maximum number of tweets per request. This number can be changed anytime if another one is considered more appropriate in the future. A second filter that must be applied on the second and further requests is the ID of the last Tweet received. This will tell the Twitter API that we want to retrieve the tweets that were posted just after the one indicated in the ID, that way if the author had posted more than 15 tweets after the last request there will not be missing tweets, it will just take a few more requests to retrieve them all.

The proposed design of the tweets gathering process was devised to be repeated periodically with sufficient separation in times of execution so that authors would have enough time to post new tweets and at the same time avoid staying too

many tweets behind the latest author's tweet. In order to ease the explanation of this design, the Figure 2 provides the flowchart followed by a detailed description of the relevant steps.

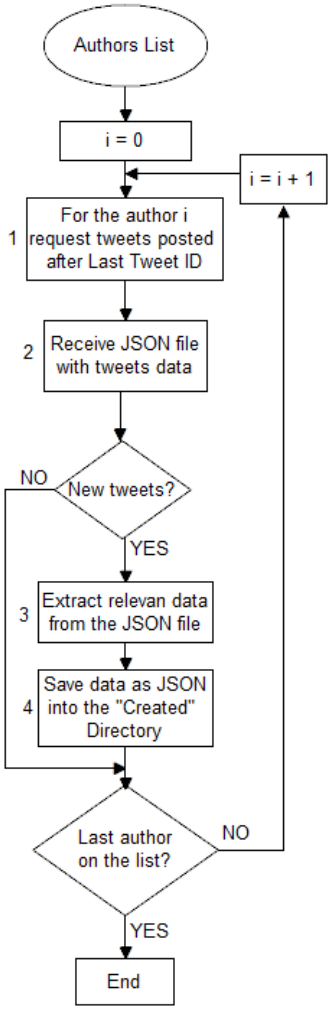

Figure 2. Tweets Gathering Process

As we can observe the process is an iterative loop that will be repeated as many times as the number of registered authors. For each author the process consists of four steps which are numbered from 1 to 4 in the flowchart. Each of these steps is described on the lines below.

1) Taking the author's Twitter Username and the Last Tweet ID values, a URL is built and sent inside an HTTP GET request with Twitter server as its destination.

2) If there are new tweets posted by the current author, the JSON file returned by Twitter will include the maximum of 15 tweets posted after the one matching the Last Tweet ID value. In case there are no new tweets, the JSON file will be empty and therefore the next 2 steps will be skipped.

3) Since not all the data coming from Twitter is relevant for the following processes, just the relevant data from each tweet will be extracted.

4) The last step will be storing the new JSON file into the "Created" directory. Each JSON file will have a different name determined by the Author ID and the time of creation. These files will be later open by the keywords extraction process.

Results:

The result from this process will be a certain number of JSON files containing the data of the new tweets. To get a better idea of what data will be stored with each tweet the following figure shows an example one tweet inside one of the JSON files:


```
- "feed": {
      "nid": "2309",
      "changed": "1340247660",
      "type": "@",
      "text": "sportsguy33",
      "created": 0,
      "lastid": 213724806825254900,
      "theme":"{\"controles\":{\"lastid\":213724806825254913,
      \"ratio_valid\":0.76663939242337,\"grabados\":1,\"latency
      \":55998.455641205}}",
      "uid": "18",
      "lang": "en"
  },
- "data": {
      "title": "fyi RT @Grantland33: Last weekend to take advantage of
      @Thrillist deal on the Grantland Quarterly.
      Issues 1-3: 25% off. ",
      "created": 1341589201,
      "text": "fyi RT @Grantland33: Last weekend to take advantage of
      @Thrillist deal on the Grantland Quarterly. Issues 1-3: 25% off.
      http://t.co/TXnfPvPx",
```

```
      "from_user": "sportsguy33",
      "from_user_id": 32765534,
      "to_user_id": 0,
      "location": "",
      "to_user": "",
      "profile_image_url":
      "http://a0.twimg.com/profile_images/2297975306/image_normal.jpg"
   },

"body": "\n<div id=\"customFieldHead\" readability=\"8\">\n<p>SUMMER
READING, BUT LIKE... GOOD</p>\n</div>\n\n<h6>All 3 issues of Grantland
Quarterly, a sports &amp;amp; pop culture bible written by Bill Simmons &amp;amp;
co.</h6>\n\n\n<div id=\"singleDescript\" class=\"noSelect\"
readability=\"31\">\n<p>Obviously, reading books is for herbs, what with
the existence of the internet, and stuff. But what if we told you
that <em>Grantland Quarterly</em>, while undeniably being a gross book,
is full of the best articles ever published on Bill Simmons' eponymous
sports/pop culture zeitgeist website? <em>What if?!</em> </p><p>Pick your
jaw off the floor and wrap your brain around this offer: for just $62
(rather than $82), you can have the first 3 issues of these gorgeous
hardcover beauties filling your coffee table/bathroom with impossibly
poignant long-form sports/culture-writing about everything from Snooki's
follies, to LeBron's evolving legacy, though comparing her with the King
is a little harsh, considering that her team was comprised solely of
Boozers.</p><p>These tomes also feature original covers and accompanying
art, as well as a bonus reversible poster of "Ice Man" George Gervin for
your wall. Shipping's included in our price, so order this offer now and
prepare to enjoy reading the best writing on the web, off the
web.</p>\n</div>\n",
"titleurl": "$62 for 3 Issues of Grantland Quarterly ($82 Value)",
"url": "http://rewards.thrillist.com/deal/11852/62-dollars-for-3-
hardcover-issues-of-grantland-quarterly",
"domain": "Ver noticia en rewards.thrillist.com",
"media": [ ],
"fix": 1,
"ext": 1
},
```


Extract Keywords from Tweets

Ideally the automatic selection of keywords from a text should be performed by recognizing the context and topic and differentiating the relevant words or group of words from the rest by its meaning, just as an educated human brain would do it. Unfortunately in the current days there is no tool with such intelligence available to the public. For this reason we have been forced to look for a tool that would at least help us to discriminate the type of words so that we get rid of everything that is always irrelevant and keep those words that could possibly become a keyword.

During our research phase, we encountered with various tools but only two of them met our needs. The first one we tried was Zemanta, as it was mentioned in

the State of the Art section this tool does the job but has certain limitations because its free version works with HTTP requests to the Zemanta website and therefore making the keyword extraction process very slow.

The second tool we found was FreeLing. Fortunately this open-source software works locally so we were able to download it and install it inside our server. Once it is installed all the requests are made locally so process is fast enough to meet our needs.

Having FreeLing installed and configured to perform morphological analysis of text we proceeded with the design of a process that would make the most from the FreeLing analysis result.

Since we are going to analyze the text from the JSON files saved in the “Created” directory, the first step of the process should be to look for a new JSON file in that location. In order to make sure that certainly most times there would be a JSON file to process we programmed this process to be executed periodically just 1 minute after the Tweets Gathering process has finished, this way the latest JSON files would be already available in the “Created” directory. The Figure 3 gives a general idea of the proposed process while in the explanations of each step more specific flowcharts are going to be presented.

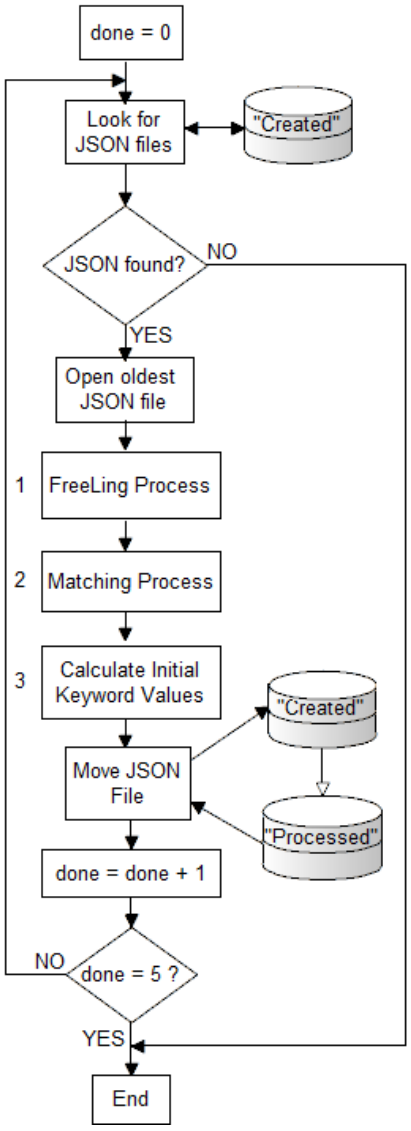

Figure 3. General Flowchart of Keyword Extraction Process

Looking into the previous flowchart we can observe that it starts by initializing the variable “done” to 0. The definition of the variable “done” is proposed to establish the maximum number of JSON files to process every time the keywords extraction process is executed. This is a way to prevent the process from running for a long period of time and collapse the systems memory. After several tests a maximum of 5 JSON files per process execution was determined as reasonable value but this can also be changed in the future.

Following the initialization of the “done” variable, the iterative sequence of steps starts by looking for JSON files in the “Created” directory. If the “Created” directory is empty the keywords extraction process will end without performing the further steps. When one or more JSON files are found, the process continues by opening and extracting the data from the oldest JSON found in the “Created” directory. The oldest JSON will be the one with the earlier time of creation.

Once a JSON has been open and its data has been extracted the 3 steps proposed that follow would be the most important of the whole process, for this reason they have been numbered in the previous flowchart and in the next lines a detailed explanation of each one is given..

1) FreeLing process:

The first one of the major steps proposed is the one that calls the FreeLing Analyzer. As we can observe in the Figure 4, we considered that it should start by checking the language which is included in one of the fields inside the JSON file. As the language must be specified on each request made to FreeLing, depending on the language indicated in the JSON file, the request to launch the FreeLing Analyzer server will be done using a different set of parameters.

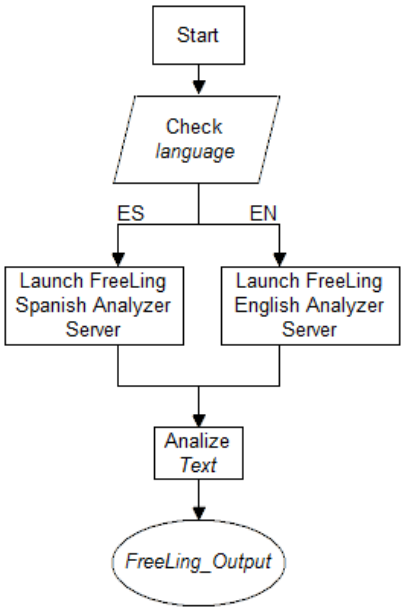

Figure 4. FreeLing Process

Once the server is launched, the application will use the FreeLing client to send the text to be analyzed. The content of the text extracted from the JSON file includes text from all the tweets integrating the file. For each tweet there are three text fields that would be appended using the Formula 1.

Text = Tweet + (space) + URL Title + (space) + URL Body.

Formula 1. Appending of the tweet's text fields

If the tweet comes without URL link then the Text will only be the Tweet itself. Some tweets would only come with a title while others would come also with a URL link and a body. This will only affect the number of keywords that would be extracted from the tweet. Since the URL Body could be a very large text which is going to require a long processing time, we had considered appropriate to establish a limit for the maximum number of characters for this field. This means that without cutting any word or sentence the URL Body should be truncated after certain number of characters has been reached. In our proposed solution we set this limit to 500 characters.

The result from the FreeLing analysis comes as a text and it is formatted as the one presented in the FreeLing introduction given in the State of The Art section. For a matter of simplification we have decided give the name "FreeLing Output" to the analysis result, so from now on we are going to call it by that name.

As we observed in the State of The Art section, the "FreeLing Output" contains all the words and elements from the text classified and separated line by line. In order to extract the relevant words from this result we propose the following step called "Matching Process".

2) Matching process:

The “Matching Process” is our proposed solution to extract the relevant words from the “FreeLing Output”. The idea is to check the “FreeLing Output” line by line and pick up just the words that match certain criteria. As each element of the “FreeLing Output” contains its morphological information which includes a tag to classify it into a particular category (E.g. Adjective, verb, proper noun, ect), we have decided to select some of these tags for our matching criteria. The specific tags chosen as relevant for the matching criteria is presented later in this section.

The flowchart of the proposed design of the Matching Process is presented in the Figure 5.

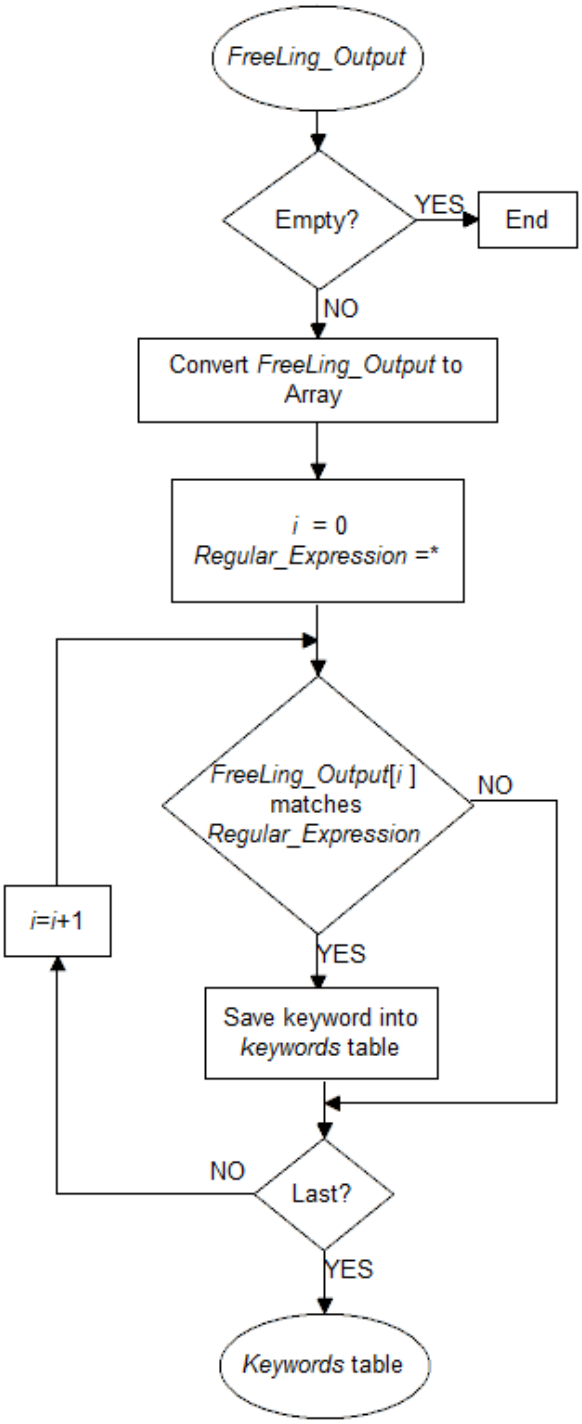

Figure 5. Matching Process

As it can be observed in the Figure 5, the first step is to check if the "FreeLing Output" was empty. The reason is that sometimes FreeLing analysis

could fail for different reasons and therefore the result comes empty. If this is the case, the process ends and no keywords are returned.

After checking whether the “FreeLing Output” is empty or not, if it is the last case then the process will continue by converting the “FreeLing Output” into an array. This means that each line of the “FreeLing Output” text will be a value of the array so that the array will have as many values as lines where in the original “FreeLing Output”. We decided to do this conversion because it was easier for us to program an iterative sequence for reading an array than to program one for reading a text.

Just after converting the “FreeLing Output” into an array the variable “i” is initialized to 0 and it will used as index to go through the array. As it was explained in the State of the Art section the “Regular Expression” is a powerful programming tool for matching a very particular pattern inside a text. In this case the “Regular Expression” is going to look for the pattern presented in the Table 6.

Location	Matching Values	Description
Second word of the text	Alphabet: a-z	Being the second word of every FreeLing_Output’s line, the second word should always be the “lemma”. Only the lemmas that contain just the matching values will be accepted. This way we remove anything that doesn’t look like a word.
	Numbers: 0-9	
	Spanish letters: áéíóúÁÉÍÓÚüÜñÑ	
	Special Characters: “-“ and “_”	
First 2 letters of the third word of the text	For Spanish: “NP”, “NC”, “AQ” and “VM”	The FreeLing_Output result brings every word with a tag indicating the semantic meaning. Every language uses a different dictionary for the tags so we have to use 2 different sets of matching values. Each set contains 4 values according to the 4 type of keywords we are extracting: Proper Nouns, Common Nouns, Adjectives and Verbs.
	For English: “NP”, “NN”, “JJ” and “VB”	

Table 6. Regular Expression for Matching Keywords

After checking each element from the “FreeLing Output” array if the element matches the “Regular Expression” then the word is saved into the keywords table. In order to take the most from the “FreeLing Output” we decided to save each keyword with two additional data fields obtained from the same element. The first one is the type of keyword which as we previously mentioned could have one of four possible values, the second field is the original word which is going to be needed later for some calculations. The Table 7 indicates all the fields contained in the keywords table.

Field	Description	Possible values
keyword	The lemma word from the “FreeLing_Output” element	Could be one or multiple words
type	The type of keyword	Proper noun (4)
		Common noun (3)
		Adjective (2)
		Verb (1)
Original word	The first word from the “FreeLing_Output” element.	Could be one or multiple words

Table 7. Keywords table fields

3) Calculating initial keywords values:

Once we have the keywords table has been built, we found the need to establish some initial values for each keyword and therefore start to distinguish some keywords from the others. This is when we decided to propose the weight and priority values..

As we can observe in the Figure 6, each keyword is assigned a keyword_title which is a combination of the keyword itself and its type. The idea is to avoid the possible mistake of storing 2 or more homonym words as the same word in the database. This is more likely to happen with Spanish words but yet it’s of no harm to have it in consideration for English texts.

Selecting keywords from the “FreeLing Output” would mean that a selection of elements with certain characteristics would become keywords while others

would be discarded. The criteria chosen for selecting keywords, was conceived from the following question:

What words are of more interest in any kind of text?

The simplest answer that came to our minds was: The nouns, the adjectives and the verbs. Why? Because they provide most of the information regarding what is the text about.

Knowing what words we were interested to select from the text and having tags on each word to differentiate them, made it as easy for us to pick up just the words with particular tags. In the Table 8 are all the tags matching the words we are looking for. It is important to note at this point that it was our personal criteria to choose just these four types of word, so if any developer in the future would like to include any other type or remove one of the proposed here to improve it will not make any damage to the rest of the design.

Type of Word	English Language	Spanish Language
Proper Noun	NP	NPXXXXX
Common Noun	NN, NNS, NNP, NNPS	NCXXXXX
Adjective	JJ, JJR, JJS	AQXXXX
Verb	VB, VBZ VBD, VBG, VBN	VMXXXXX

Table 8. FreeLing Relevant Tags.

Since we decided to work with Spanish and English texts, we have to work with 2 different sets of tags, so as it can be observed in the previous table, there can be multiple tags for each type of word in either language. This is not a problem because in both of them what matters the most are the 2 first letters of each tag. The rest of the tag letters are only for indicating additional characteristics of the word like gender, number, tense and other grammar classifications.

In order apply a sort of standardization to the keywords, instead of saving each keyword as it appears on the text we decide to save it in its plain-most form which is its lemma. Fortunately FreeLing provides it, so by saving the lemma as the keyword it will no matter if on a latter Tweet message the same keyword appears with a different form because it will have the same lemma and it will match with the one stored in the database. To get a better understanding of this idea lets observe the following example:

The word “singing” is found on a tweet message A and FreeLing returns as result:

```
singing sing VBG 1
```

From this result the keyword to save would be “sing” which is the lemma of “singing”.

Later in the tweet message B the word “sang” is included and the result from FreeLing comes like this:

```
sang sing VBD 1
```

This time the lemma of the word “sang” is also “sing”.

The words “singing” and “sang” are both different conjugations of the same verb “to sing”. This is something an English speaker would recognize easily, but making an algorithm capable of doing the same recognition would take a lot of time without the help of FreeLing.

Along with the lemma, the keyword type would be required for a later use because there is always a chance that two homonym words appear and because

of their different meanings they can be related to totally different topics, so in order to avoid counting both as the same word the type is going to differentiate them.

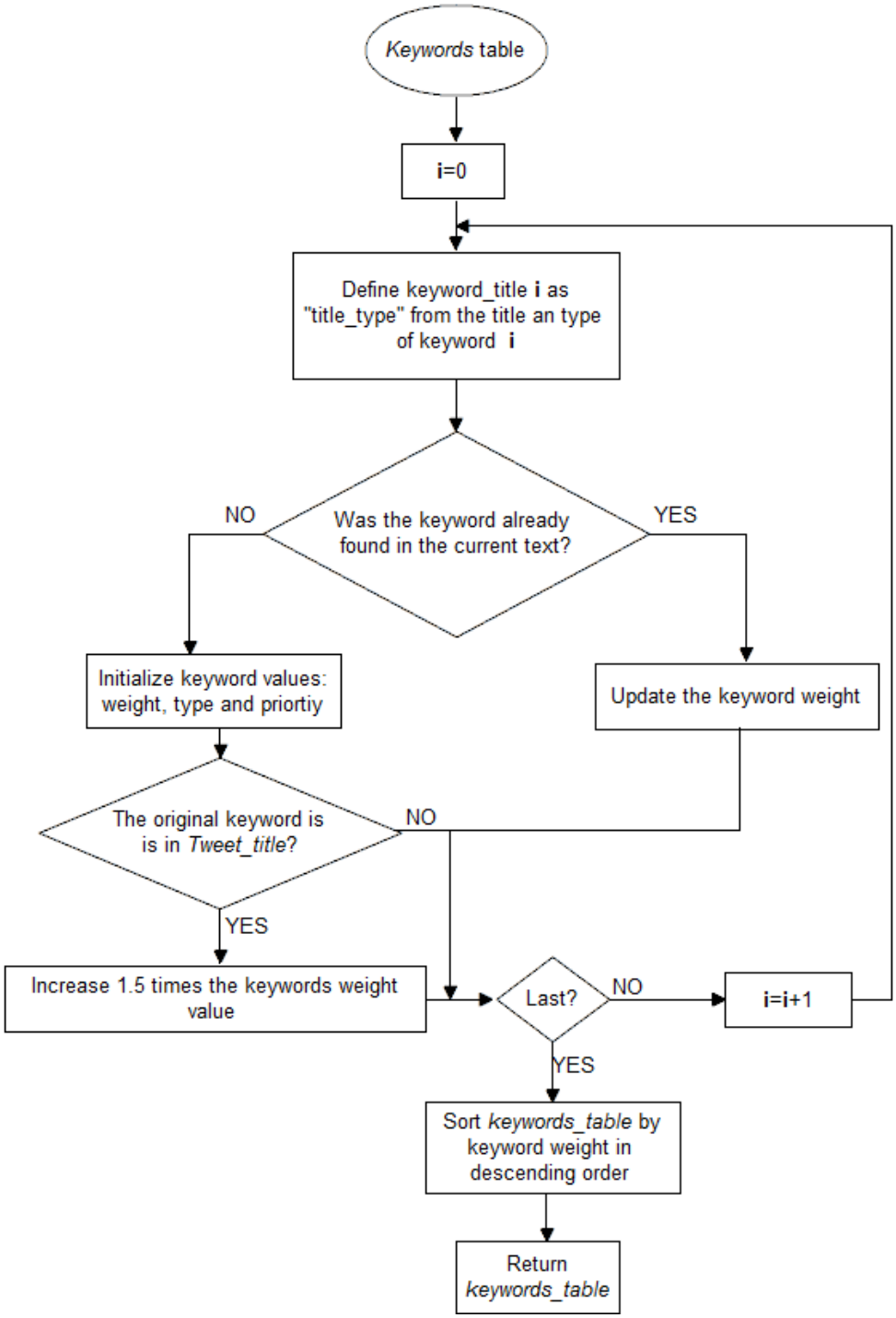

Figure 6. Calculating Initial Keyword Values

As it was mentioned before we want to give each keyword an initial weight that later would be changing over the time while the keyword remains stored in the database. In order to accomplish the calculation of the initial weight, we propose a sub-process of three steps that would be applied to each keyword.

Step 1. The keyword type establishes the initial weight:

Now that there are only four types of keywords there would be four different initial weights. In The Table 9 we propose the weights to be associated to each type of keyword; these values can be changed anytime without disturbing the process behavior.

Type of keyword	Initial weight
Proper Noun	4
Common Noun	3
Adjective	2
Verb	1

Table 9. Initial weights by keyword type.

Step 2. Number of appearances in the current Tweet adds some additional weight:

In order to give a higher weight to the keywords that are mentioned several times in the same text, we propose Formula 2 to increase the weight from the 1st step:

$$W = W * \left(\frac{1}{P} + 1\right)$$

Formula 2. Initial weight increment for repeated keywords

Where:

P is the position of the word inside the text. For example in the text “John is walking in the woods”. The position of the word “walking” is 3 because is the third word from left to right.

The idea behind this formula is to provide a small increment for each time the keyword is repeated in the text. This way if the keyword is repeated many times in the text its weight will grow moderately. Like most elements of this flexible design are, this formula could be changed for a better one in the future.

Step 3. Keyword mentioned on the Tweet’s Title or in the URL Title adds even more weight:

Taking into account that a tweet message itself and the URL title are the first information a user would read before reading the URL body, we decided that an additional increment to the weight should be given to any keyword which happens to be found in any of those two elements of the tweet. In order to keep the maximum initial weight below the value of 10.00, we decided to multiply by 1.5 the resulting weight from steps 1 and 2. This 1.5 value can be changed in case a more appropriate one is considered in the future.

Looking into the next step of the flowchart in Figure 6 we can observe that is conditional element asking if the keyword was already found in the text. If the answer to this question is NO it means that the keyword was not already in keywords_table and therefore it must be added along with the initializations of its keyword values: type, priority and weight.

Final steps:

Just after the last of the three major steps has been finished, if we look back into the flowchart from Figure 3 we will observe that recently processed JSON file gets moved from the directory “Created” to the directory “Processed”, then the variable “done” is increased by 1 and then a conditional block checks if it’s the 5th

JSON processed in the current execution and finish the process if the answer is “yes” or if the answer is “no” it goes back to look for another JSON file to process it.

Results:

The results of the keywords extraction process are obtained after the keyword values initialization step. As the process runs, one multidimensional array with all the keywords and their respective values is created. The Table 10 shows all the values added with each keyword:

Field	Description	Possible values
keyword	String value of the keyword	Could be one or multiple words
type	The type of keyword	Proper noun
		Common noun
		Adjective
		Verb
priority	The priority of the keyword	All keywords are initialized with a medium priority (2) by default
weight	The initial weight of the keyword.	A decimal number from 1 to 10.

Table 10. Initial keyword values.

Keywords Weighting System

Once that we had designed a method to obtain table with the keywords extracted from the tweets, we had to continue the solution with a process that would check which keywords were new and which keywords were already stored in the database. This process should also update the database by adding all those new keywords and updating the values of the old keywords that are being repeated.

We proposed a design for this process and called it “Keywords Weighting System”. The input four our designed process would be the array of keywords obtained from the last execution of the keywords extraction process. This means that the process should be executed every time a keywords extraction has been done. As we can observe in flowchart presented in the Figure 7, the process starts

be initializing the variable “i” at 0 so it can be used as the index for reading the array of keywords.

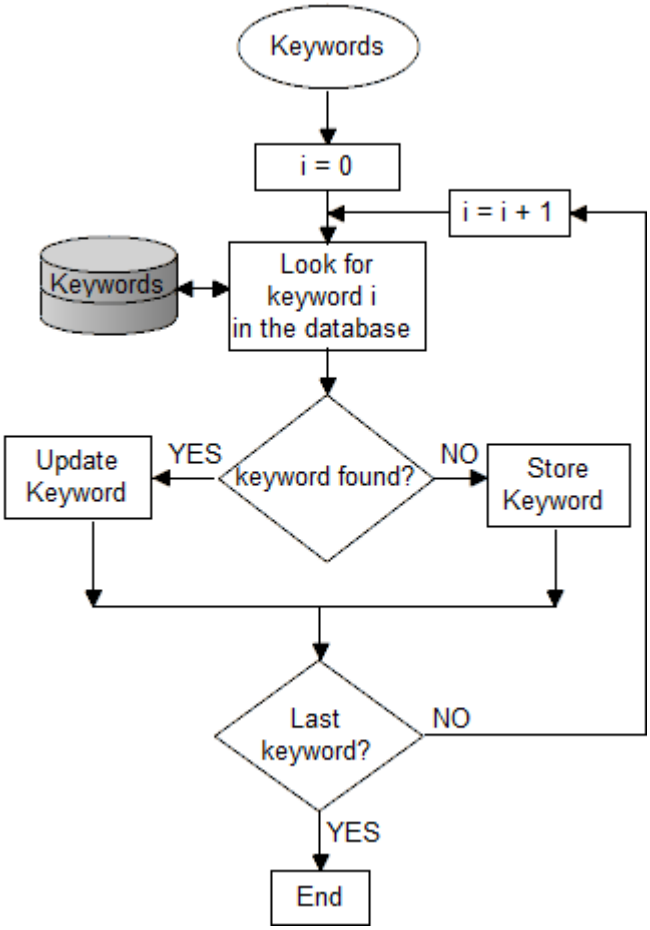

Figure 7. Keyword Weighting System General Flowchart

After the variable “i” is initialized at 0, the iterative sequence of steps starts with reading the keyword in the position “i” of the array and look if that keyword is already stored in the database. Just after making this database query comes a conditional step which ask if the keyword was found or not in the database. Depending on the result of that conditional step, the sequence will continue with the Store Keyword step or the Update Keyword step. In the following lines are the descriptions of these two steps.

Store Keyword:

When the keyword “i” is not found in the database, the sequence continues with the Store Keyword step. In this very simple step, the keyword gets some additional values and then it’s stored into the data base. A more detailed description is presented in the lines below.

In the proposed design every extracted keyword comes initially with the fields presented in the Table 11.

Keyword	Type	Weight	Priority

Table 11. Initial keyword data fields

Once a keyword has been selected to be stored in the database three new fields are going to added to the previous ones. These new fields are show in Table 12.

Changed	Frequency	Latency

Table 12. Additional keyword data fields

These new fields where projected to be always initialized with the following values:

Changed = current time.

Frequency = 1.

Latency = 0.

All the sub-steps proposed for the Store Keywords step are presented in the Figure 8:

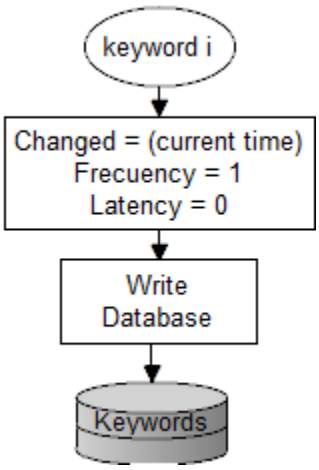

Figure 8. Write New Keyword

The Table 13 shows all the fields that are going to be stored in the database for each keyword:

Keyword	Type	Weight	Priority	Changed	Frequency	Latency

Table 13. Stored keyword data fields

Update Keyword:

When the keyword “i” is found to be already stored in the database the Update Keyword step is proposed as the next in the sequence. In this step, some of the previous values of the keyword such as the frequency, the latency and the weight get updated. A more extended description is presented in the lines below.

Looking into the Figure 9 a flowchart with all the sub-steps from the Update Keyword step can be observed:

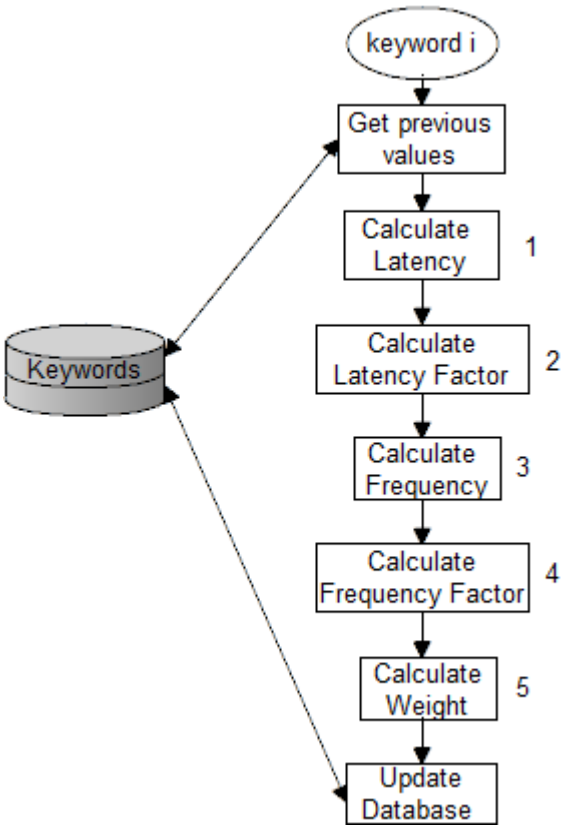

Figure 9. Update Keyword Values

As we can observe in the flowchart, the database is called two times in the sequence, the first time to get the previous values of the keyword "i", and the second time to update the database with the new values. In the next lines all the numbered steps are going to be explained.

1) Calculate Latency:

The definition of the latency of a keyword is proposed as the time difference in seconds between the current and the last time of appearance of the keyword. It can be modeled with the Formula 3.

$$L = CTA - PTA$$

Formula 3. Latency Calculation

Where:

CTA: Current Time of Appearance

PTA: Previous Time of Appearance

2) Calculate Latency Factor:

The latency factor is planned as a measure related to the variation of the latency between the actual keyword and its previous appearance. The Formula 4 is proposed to calculate this value.

$$LF = \left(\frac{L' - L}{L' + L}\right) * \frac{1}{2} + \frac{1}{2}$$

Formula 4. Latency Factor

Where:

L': Current Latency

L: Previous Latency

3) Calculate Frequency:

The frequency value is projected as a counter of the number of appearances of a keyword since the moment it was added to the database. The Formula 5 is proposed as a simplified definition.

$$F = PF + 1$$

Formula 5. Frequency counter

Where:

F: Current Frequency

F: Previous Frequency

4) Calculate Frequency Factor:

The frequency factor is proposed as a scaled 1 to 10 measure of how many times the keyword has appeared since the first time. The resulting frequency factor is determined by the criterion presented in Table 14.

Times of Appearance	Frequency Factor
1 to 10	1
11 to 25	2
26 to 50	3
51 to 100	4
101 to 200	5
201 to 400	6
401 to 700	7
701 to 1000	8
1001 to 1500	9
1501 or more	10

Table 14. Frequency Factor Selection Criterion.

5) Calculate Weight:

The Formula 6 us proposed for the recalculation of the weight. As we can observe the new weight would be composed by a 90% of the previous weight value and a 10% of the New Results value which is going to be explained on the following paragraph.

$$NW = PW * 0.9 + NR * 0.1$$

Formula 6. Weight Update

Where:
NW: New Weight
PW: Previous Weight
NR: New Results

Every time a keyword extracted from a tweet matches with one stored in the Keyword Database there is an update of the Keyword Weight, to obtain the New Weight it is necessary to calculate the proposed New Results value. The Formula 7 was projected for the calculation of this value.

$$NR = [(LF * 0.8) + (FF * 0.2)] * 0.5 + [(TV * 0.7) + (PV * 0.3)] * 0.5$$

Formula 7. New Results Value Calculation

Where:
LF: Latency Factor
FF: Frequency Factor
TV: Type Value
PV: Priority Value

Here is a description of the last two values:

Type Value:

The type value is determined by the criterion presented in Table 15.

Keyword Type	Type Value
Proper Noun	10
Common Noun	7.5
Adjective	5
Verb	2.5

Table 15. Type Value by Keyword Type.

Priority Value:

The priority value is determined by the criterion presented in Table 16

Keyword Priority	Priority Value
High	10
Medium	20/3
Low	10/3

Table 16. Priority Value by Keyword Priority.

Author Weighting System

The Author Weighting System is proposed as a process that stores some of the keywords recently mentioned by the author, applies them the same weighing criteria as the one used in the Keywords Weighting System but instead of storing the new keyword or updating its values in the keywords table of the database, it makes the update in the author's own keywords list.

The maximum number of keywords projected for the author's keywords list is 30 keywords. This means than once the list is fully occupied, the new incoming keyword with highest weights will substitute the old ones if and only if the weight of each new keyword is higher than the weight of an old one.

Every time the author's keyword list gets updated so does the author's weight. The reason for this is because the author's weight is the average weight of all the keywords registered in his own list. This will be better explained in the lines below.

The inputs for the Author Weighting System are the array of keywords, the author's ID and table of his previously mentioned keywords. The Formula 8 is proposed for the recalculating the author's weight every time new keywords are added to his keywords list.

$$Author\ Weight = \frac{\sum_{i=1}^{30} keyword\ weight_i}{30}$$

Formula 8. Author Weight Calculation

When the Authors Weighting System finish its execution, all the processed authors have a new weight. This new weight will play the main role in the Update Authors List process. This process is not included in the prototype so it remains pending for a more developed version of the current solution.

Update Keywords List

As it was explain in the beginning of the section, the keywords stored in the database can have two possible states: published or not published. The published state means that the keyword is visible to the application's user at the keywords list while in the not published state the keyword remains hidden.

Since all the new keywords are stored in the database as not published by default, it is necessary to have a process for promoting some of the newer keywords to the publish state while removing some of the older published ones.

The first aspect to take into account is that the number of keywords per user must be limited to avoid intensive calls to the database. Also to have a certain level of organization, a maximum number of published keywords must be set to a reasonable amount so that user can check all of them quickly. The Table 17 shows the chosen limits for our prototype.

Maximum published keywords	100
Maximum unpublished keywords	50
Total keywords per user	150

Table 17. Keyword amount limits.

With those limits clearly established we can now proceed with the process execution. Ideally this process should be running periodically with enough time apart from the keywords weighting processes. The reason is basically because while the weighting process adds new keywords to the database, this process removes keywords from the database and therefore generates free spaces for the new keywords. If there were no free spaces in the database the new keywords would have to be discarded.

The Figure 10 shows the first 4 steps proposed in the design of the Update Keyword List process:

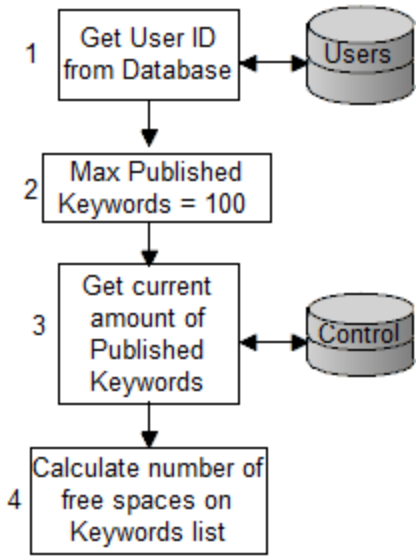

Figure 10. Update Keywords List (part 1)

Here is a description of each one of the steps shown in Figure 10:

1) Since each user of the application would have his own list of keywords then it's crucial to get the user's ID before anything else. In order to do this the application performs a query to the "users" table of the server database, it will look for the user with more time since being treated. The rest of the process will work only with the keywords belonging to that user.

2) As it was previously mentioned a maximum of 100 published keywords per user was chosen for the prototype, but since this value is assigned to a global variable, it can be changed anytime in the future.

3) The "control" table of the server's database has registered the number of published keywords and authors for each user. Making a simple query to this table, the current amount of published keywords is retrieved.

4) Having the maximum number of published keywords and the actual number of keywords on the list the application calculates the number of free spaces by performing subtraction shown in Formula 9.

$$Free\ Spaces = Max.\#\ of\ Keywords - Current\ \#\ of\ Keywords$$

Formula 9. Free Spaces Calculation

Having the amount of free spaces on the keywords list, the process continues with the steps proposed in the Figure 11.

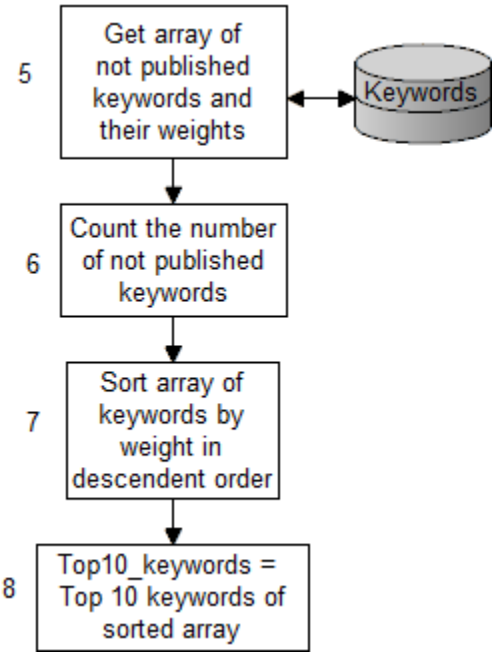

Figure 11. Update Keywords List (part 2)

5) This is a query to the “Keywords” table of the server’s database. It retrieves an array with all the user’s unpublished keywords and their respective weight.

6) This step just performs the count of elements in the array obtained in the previous step.

7) The array gets sorted by the weights of the keywords so that keywords with higher weights will be on the top of the array while the keywords with lower weights will be on the bottom of the array.

8) Having the array sorted, this steps takes the top 10 keywords from that array and stores them in a separate array called “Top10_keyords”.

Once the array with the top 10 keywords is gotten, the following part of the process will fill free spaces of the keywords list with the ones in the array. If the keywords list is already full (no free spaces), then this part of the process will be skipped. In the Figure 12 are presented all the steps involved in this part of the process.

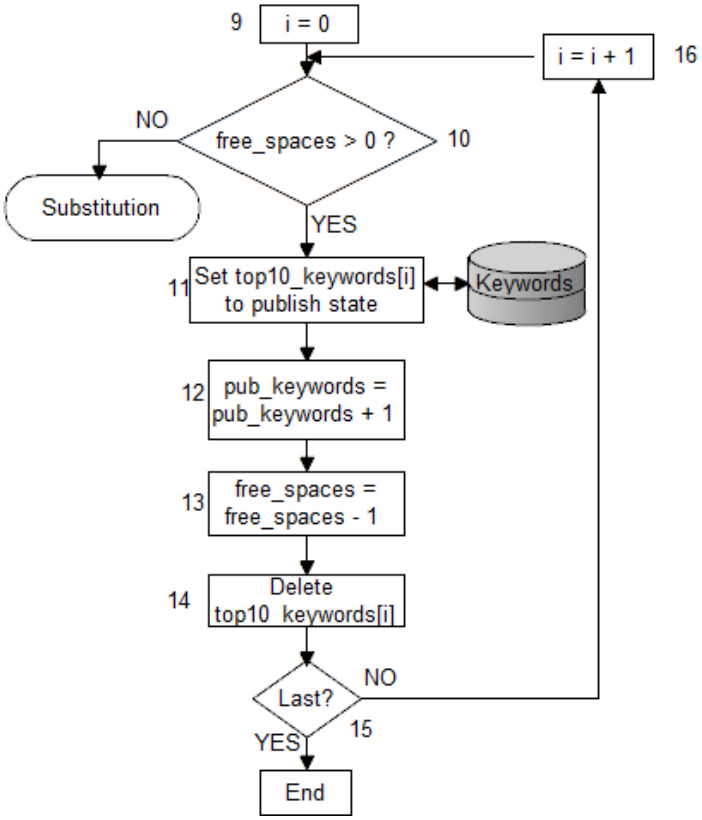

Figure 12. Update Keywords List (part 3)

Following the numbers of the steps as shown in the flowchart, a description of each step is presented next:

9) The variable “i” is initialized at zero so it will be used as the index to process element by element the array with the top 10 keywords.

10) The variable “free_spaces” is checked. If its value is above zero it means there are free spaces in the keywords list and therefore the process will continue to step 11. If there are no more free spaces the rest of the steps of this part of the process will be skipped and go straight to the “Substitution” part which is going to be explained later in this section.

11) Taking the keyword value inside the position “i” of the “top10_keywords” array, the application looks for this keyword in the database and changes its status from 0 (not published) to 1 (published). Now the keyword will appear on the keywords list.

12) The variable “pub_keywords” is incremented by 1. This variable represents the current number of published keywords.

13) The variable “free_spaces” is decremented by 1. This logical since each new published keyword occupies one of the free spaces on the keywords list.

14) Delete the element “i” if the “top10_keywords” array.

15) If this is the last keyword on the “top10_keywords” array then the process will end, otherwise it will go to step 15.

16) The variable “i” is incremented by 1 so that when the loop starts over it will process the next keyword of the “top10_keywords” array.

The final part of the Update Keywords List process design is the “substitution” part. This part will take place only if there are keywords remaining in the

"top10_keywords" array while the keywords list is already filled. The Figure 13 shows all the steps involved in the "substitution" part.

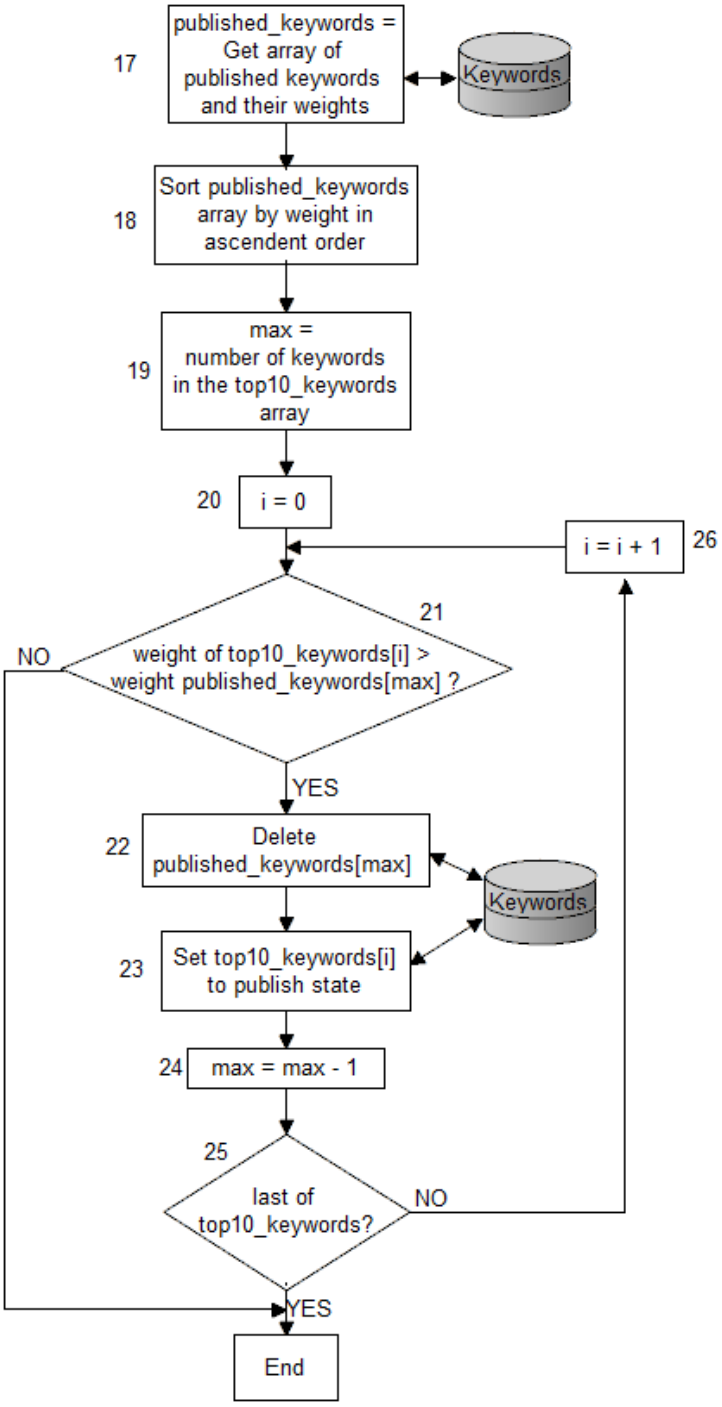

Figure 13. Update Keywords List (part 4)

17) This is a query to the “Keywords” table of the server’s database. It retrieves an array with all the user’s unpublished keywords and their respective weight. The array is saved as “published_keywords”

18) The “published_keywords” array gets sorted by the weights of the keywords so that keywords with lower weights will be on the top of the array while the keywords with higher weights will be on the bottom of the array.

19) The number of elements in the “top10_keywords” array is counted and the result is saved in the variable “max”.

20) The variable “i” is initialized at zero so it will be used as the index to process element by element the “top10_keywords” array.

21) The weight of the keyword in the position “i” of the “top10_keywords” array is compared with the weight of the keyword in the position “max” of the “published_keywords” array. If the first one is greater than the second then the process continues to step 22, otherwise the process ends.}

22) The keyword in the position “max” of the “published_keywords” array is deleted from the database. Now this keyword will not appear anymore in the keywords list.

23) Taking the keyword value inside the position “i” of the “top10_keywords” array, the application looks for this keyword in the database and changes its status from 0 (not published) to 1 (published). Now the keyword will appear on the keywords list.

24) The variable “max” is decremented by 1. This means that next time the process is in step 21, a different keyword with a lower weight will be revised.

25) If this is the last keyword of the “top10_keywords” array then the process will end, otherwise it will continue with step 26.

26) The variable “i” is incremented by 1 so that when the loop starts over it will process the next keyword of the “top10_keywords” array.

Result:

As a result of this process new keywords will appear on the keywords list while the ones with lowest weights will be removed.

Parametric Data

As it can be observed in most parts of the proposed solution, many of the processes contain parametric data which could be changed anytime and would not break the right functioning of the application. The Table 18 presents a summary of all the parametric data.

Location	Data	Proposed value
New Tweets Gathering	Maximum Number of Tweets per request	15
Keywords Extraction Process	Maximum URL Body characters	500
	Initial weight per keyword type	Table 9
	Formula for increment initial weight on keywords repeated in the same text	Formula 2
	Multiplicative factor applied to the initial weight on keywords mentioned in the Tweet or URL title	1.5
Keyword Weighting System	Latency	Formula 3
	Latency Factor	Formula 4
	Frequency	Formula 5
	Frequency Factor	Table 14
	Type Value	Table 15
	Priority Value	Table 16
	Keyword Weight Update	Formula 6
	New Results Value	Formula 7
Author Weighting System	Maximum keywords per author	30
	Author Weight Update	Formula 8
Update Keywords List	Keywords amount limits	Table 17
	Maximum number of keywords to promote per execution	10

Table 18. Parametric Data

Conclusion and Future Lines

As it was expected from the beginning, this project resulted just in a prototype to launch an initial idea of how to improve the use of Twitter as a customized source of content.

After performing several tests with the prototype, some inaccuracies were detected when analyzing the results of the processes. On the first hours after the user had introduced the initial keywords and authors, the application adds to the lists, some keywords and authors that are not related with the initial ones. This was expected because due to the nature of the application, the time of convergence of the authors and keywords lists to an appropriate reflection of the user's interest will depend mostly on how frequently the initially selected authors post new tweets.

Even though FreeLing software was the most appropriate tool found to perform the keywords extraction process, it still has one issue regarding the assignment of the type for each keyword. If the keyword appears on the text with a first letter in uppercase then FreeLing always considers it a proper noun, sometimes this is true but most of the times it is not, so this is something that will require to be fixed in the future.

Another limitation found with FreeLing was that long texts took too much time to get analyzed and often got stuck in the middle of processing. To prevent that, a limitation in the number of characters of the URL body was established so that long texts would have to be truncated to a certain length before being analyzed.

Regarding the formulas and values established to calculate the weights for the keywords and authors, as they were all chosen using personal criteria, it is expected that with proper research works and future innovative ideas those formulas and values get substituted with improved ones, and therefore get better accuracy in the weights calculations.

Another aspect that would require some revision is the time between executions of tweets gathering process. As the new tweets will available as soon as the authors post them, the frequency in which the application should ask Twitter for new tweets should be similar to the average posting frequency of all the authors. This way the tweets will we be gathered without mayor delays and also there will be a minimum of empty responses from Twitter.

The mechanism we designed to calculate the keyword weights takes quite a lot of time to determine which keywords are really related to a certain topic. The reason for this is that some of the measures (E.g. frequency and latency) we included on the calculations will take many repetitions of the processes to make some keyword weights excel the others.

An ideal solution to get faster and more accurate calculations for the weights in the future should include a measure of how close is a new keyword related to the ones that are already in the keywords list. This measure would probably require a sort of smart dictionary which can group all the registered keywords into particular topics and then check if the new keyword fits in one of these topics.

Finally, although the work done so far could be considered just as the first stage of a major project, it can also be seen as an initial set of ideas for other innovative ways to improve the use of social networks . Even if all the algorithms proposed here are completely modified or deprecated in the future, at least the ideas of extracting keywords from the text or applying a weighting system could still make sense.

References

[1] http://www.php.net/manual/en/preface.php

[2] http://drupal.org/

[3] http://dev.mysql.com/doc/refman/5.0/es/index.html

[4] http://www.phpmyadmin.net/home_page/index.php

[5] http://www.json.org/json-es.html

[6] https://dev.twitter.com/

[7] http://www.zemanta.com

[8] Analizadores Multilingües en FreeLing. Lluís Padró. Linguamatica, vol. 3, n. 2, pg. 13--20. December, 2011.

[9] FreeLing 2.1: Five Years of Open-Source Language Processing Tools. Lluís Padró and Miquel Collado and Samuel Reese and Marina Lloberes and Irene Castellón. *Proceedings of 7th Language Resources and Evaluation Conference (LREC 2010), ELRA*. La Valletta, Malta. May, 2010. http://nlp.lsi.upc.edu/freeling

[10] FreeLing 1.3: Syntactic and semantic services in an open-source NLP library. Jordi Atserias and Bernardino Casas and Elisabet Comelles and Meritxell González and Lluís Padró and Muntsa Padró. *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC 2006)*, ELRA. Genoa, Italy. May, 2006. http://nlp.lsi.upc.edu/freeling

[11] FreeLing: An Open-Source Suite of Language Analyzers. Xavier Carreras and Isaac Chao and Lluís Padró and Muntsa Padró. *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, 2004.

[12] http://www.zytrax.com/tech/web/regex.htm

[13] http://gigaom.com/2012/05/01/twitters-big-problem-it-still-needs-better-filters/

[14] http://peoplenews.me