ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΥΝΑΙΣΘΗΜΑΤΙΚΗ ΑΝΑΛΥΣΗ ΒΑΣΙΣΜΕΝΗ ΣΤΑ ΚΟΙΝΩΝΙΚΑ ΔΙΚΤΥΑ

του

ΖΑΜΠΟΥΝΗ ΦΙΛΙΠΠΟΥ

Επιβλέπων : Βάβαλης Μανόλης, Καθηγητής
Μέλη Επιτροπής : Κατσαρός Δημήτριος, Λέκτορας

Βόλος, Σεπτέμβριος 2013

UNIVERSITY OF THESSALY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

THESIS

SOCIAL MEDIA BASED ON SENTIMENT ANALYSIS

by

ZAMPOUNIS FILIPPOS

Supervisor: Vavalis Manolis, Professor
Committee Members: Katsaros Dimitrios, Lecturer

Volos, September 2013

Filippos Zampounis
University of Thessaly & Skype Communications

ABSTRACT

This thesis concerns the wide area of sentiment analysis that is based on text found in social media on the World Wide Web. Besides the general exposition to the emerging scientific and technological area of sentiment analysis this thesis should be considered also as an initial effort to examine whether sentiment analysis practices and tools will be beneficial to web based Question and Answer (QA) communities. We exclusively focus on web social structures which are "thematic closed". Such structures drastically restrict the question so they truly belong to a relatively restricted topic. We introduce a general framework, examine the various characteristics of QA communities that are related with our sentiment analysis viewpoint, elucidate on those which seem to be the most import ones and draw our conductions which we attempt to validate through an actual implementation concerning the *Stack Overflow* site.

Keywords: Information Retrieval, Sentiment Analysis, Web Science, Q&A Systems, Stack Overflow.

ΠΕΡΊΛΗΨΗ

Σε αυτήν την διατριβή κάνουμε μια αρχική προσπάθεια να εξετάσουμε εάν τεχνικές και εργαλεία συναισθηματικής ανάλυσης μπορούν να είναι ωφέλιμα για ιστοσελίδες ερωτήσεων και απαντήσεων (Q&A). Εστιάζουμε κυρίως σε ηλεκτρονικές κοινότητες με πολύ συγκεκριμένη θεματική ενότητα, αναλύουμε τα ιδιαίτερα χαρακτηριστικά αυτών και φτιάχνουμε ένα γενικό πλαίσιο για την ανάλυση τους με βάση τις ιδιομορφίες τους σε σχέση με την συναισθηματική ανάλυση. Καταλήγουμε υλοποίηση ενός συστήματος το οποίο εκμεταλλεύεται αυτά τα ιδιαίτερα χαρακτηριστικά και τα εφαρμόζει στο Stack Overflow, μια από τις μεγαλύτερες κοινότητες για προγραμματιστές.
Τέλος γίνεται αναφορά για δυνατότητες επέκτασης και σε άλλες κοινότητες του Stack Exchange δικτύου αλλά και και σε άλλες γνωστές Q&A ιστοσελίδες.

Contents

1. Introduction

Information can be obtained from several sources through various paths. Asking questions might, under certain circumstances, be one of the most effective ways. Traditionally question/answering used to be, and to some extent still is, the most common way in the non-electronic area for certain areas and particular social frames and platforms. Nowadays people come back to old fashion question/answering practice mainly for opinion mining and decision making.

In the past years we have witnessed significant progress in developing methods for obtaining pro or anti opinions or feeling sentiment expresses by the authors of a set of documents. Natural Language processing or information extraction methods are commonly employed. Therefore sentiment analysis or opinion mining is a natural language processing or information extraction task that helps extract pro or anti opinions or feelings expressed by a writer in a document collection. As far as the web concerns sentiment analysis is a way data miners can take the legwork out of understanding the meanings and feelings behind statements made in social media and other forums. The sentiments in sentiment analysis can be obtained at document level by classifying the polarity of the expressed opinion in the text of a document, at the sentence or entity feature level to find out if the opinion expressed is positive, negative or neutral.

The rest of this thesis is organized as follows. In the next section we provide the required background, we briefly discuss our motivation and we explicitly state our scientific and practical objectives. Next we present our system design efforts, and comment on implementation issues. The evaluation of our system is given in the next section through the two case studies considered. The last session contains a concluding summary together with a list of potential future research and development objectives.

2. Background, motivation, and objectives

Very often our decisions are partially based on what other people think. In certain cases this might be a crucial, or even sole, decision making factor. Therefore, identifying positive and negative opinions, reviews and emotions could be of great practical importance.

Sentiment analysis is a procedure that analyses social media conversation patterns in order to identify trends in opinion and attitude. It is often also referred as "opinion mining" or "text analytics" and it provides us with means to form an otherwise overwhelming flood of data into valuable information. It is mainly based on tracking word choice and frequency as well as word definitions. For an excellent overall review of the modern sentiment analysis thematic area the reader is referred to (Liu, 2012).

As a field of research, it is closely related to (or can be considered as part of) computational linguistics, natural language processing, artificial intelligence and text mining. Proceeding from the study of affective state (psychology) and judgment (appraisal theory), this field seeks to answer questions long studied in other areas of discourse using new tools provided by data mining and computational linguistics.

Sentiment analysis is a relatively new approach (Cheng, 2009) (Liu, Hu, & Cheng, 2005) and has been rapidly penetrating several of our research and development eras in the past three years. Nevertheless,

It is a rather challenging subject for several reasons (Feldman, 2013) that range from purely theoretical to more commonly highly technical ones. Even humans have often difficulties understanding the sentiment of someone else's saying. Therefore, it is expected machines and software agents to face ambiguity problems that prevent them from accurately identifying the tone and meaning in a statement or set of statements. In other words people express things in different ways and finding the sentiment in a sentence is hard using certain statistical approaches. Proximity analysis is usually of great help but still the problem is challenging, to say the least.

Like opinions, sentiment is inherently subjective from person to person, and can even be outright irrational. For this, it is critical to mine any available large and relevant sample of data when attempting to measure sentiment. No particular data point might be necessarily relevant. It is in fact the aggregation that matters.

An individual's sentiment toward a brand or product or in our specific case a presumably difficult problem or a crucial question may be influenced by one or more indirect causes. With a large enough sample, outliers are diluted in the aggregate. Furthermore, given that sentiment very likely changes over time according to a person's mood, events, and so forth, it's usually important to look at data from the standpoint of time.

Additional vital issues are raised in several cases. For example, sarcasm and other types of ironic language are inherently problematic for machines to detect. It's imperative to have a sufficiently sophisticated and

rigorous enough approach that relevant context can be taken into account. Such incredibly difficult issues might not commonly appear in the case of question/answering systems.

At any rate automated sentiment analysis systems are needed. As a practical matter, and regardless the above mentioned problems and general criticism, sentiment analysis has already proved itself as a powerful tool in several business aspects with customer satisfaction, market survey, reputation management/brand perception, advertisement placement, trend prediction and stock exchange method development being the most successful and widely spread ones. As an example it is worth to mention that there already exist a plethora of diverse tools and platforms that assist us to track and assess the number of times a company or a product has been mentioned in the social media channels commonly in real time[1].

As already mentioned, sentiment analysis has many applications in modern enterprises. From consumer research to marketing, to reputation management to monetizing content itself; the possibilities are enormous and ever-growing. In particular, in addition to software tools for collecting and interpreting data drawn from social platforms, sophisticated services are nowadays provided to assist us in understanding this torrent of information, helping to turn raw input into actionable strategies.

The main objective of this study is to examine the effectiveness of sentiment analysis on a particular web based social networking activity that appears in the form of question/answering. Q&A type of social activities have been for ages a prominent tool for decision making on an everyday basis. Such activities often consist the cornerstones of web social networking. Perhaps during the very earliest days of the Web, people enjoyed a range of embryonic social networking platforms without any Q&A activities. This is not the case anymore. People nowadays discuss issues in general and in the form of Q&A in particular.

Specifically, individuals increasingly rely on their distributed peer communities for information, advice, and expertise. Millions of individuals learn from each other on public discussion forums (e.g., Usenet), community-built encyclopedias (e.g., Wikipedia), social networks (e.g., Aardvark), and online question and answer sites (e.g., Yahoo! Answers). Recently, several large Q&A sites have attracted the attention of researchers [References]. In aggregate, these studies suggest that general-purpose Q&A sites have answer rates between 66% and 90%; often attract non-factual, conversational exchanges of limited archival value; and may be poorly suited to provide high quality technical answers.

In contrast, stackOverflow has become, within about two years, one of the most visible venues for expert knowledge sharing around software development. With approximately 300,000 registered users and > 8 million monthly visits, stackOverflow has an answer rate above 90% and a median answer time of only 11 minutes. It has captured significant mindshare among software developers: anecdotally, users report that the site has replaced web search and forums as their primary resource for programming problems; others now consider their portfolio of stackOverflow answers a valuable component of their professional resumes.

stackOverflow is not alone in the Web. ChaCha, a heavily funded website where you can ask about anything and get answers from volunteers or the site's database of 2 billion queries, doesn't have a

[1] See for example http://www.opfine.com/

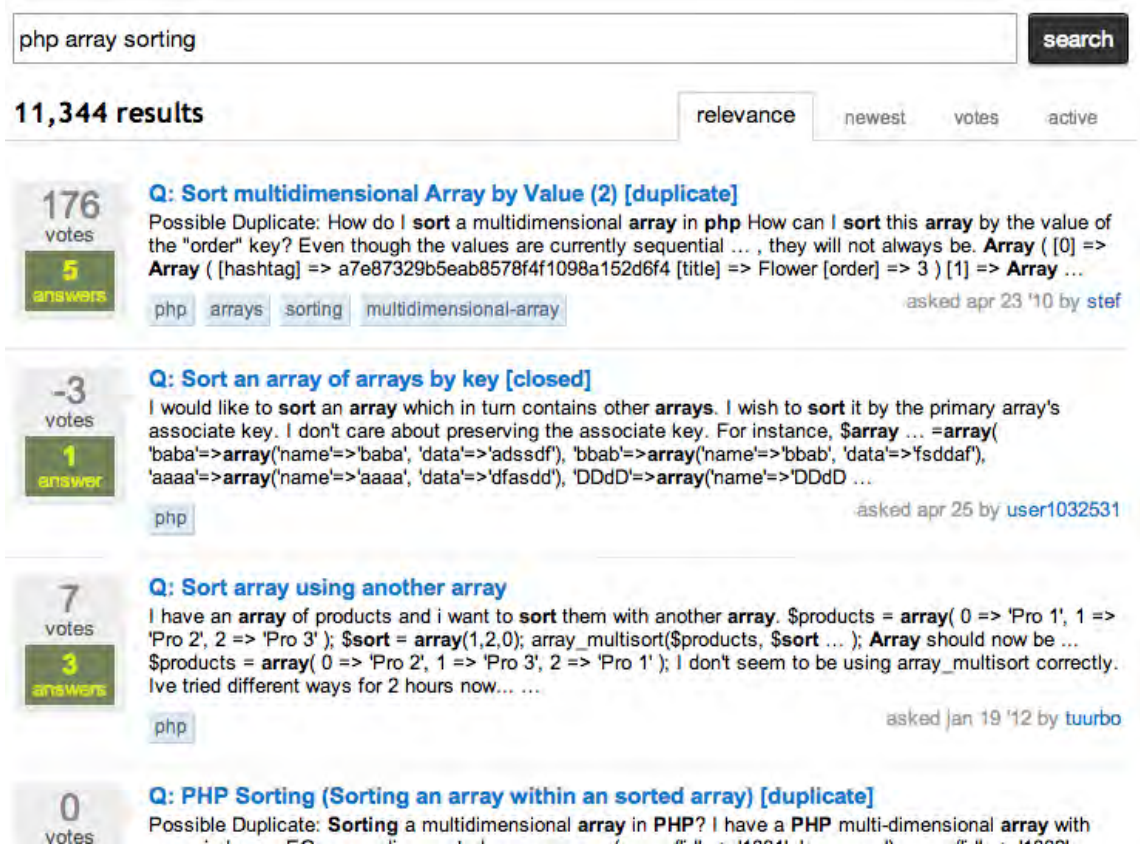
response yet to one big question: Is crowdsourced Q&A a standalone business? Whereas Google proved many years ago that Web search is a serious business, crowdsourced Q&A has yet to find its business success road.

The biggest Q&A site is Yahoo! Answers with 69 million visitors in December, an increase of 11 million compared to the previous year. All the while, Yahoo has made minimal improvements to the site, as it's been overtaken by peculiar questions and often useless answers. Therefore, Q&A sites is a strong business with huge amount of visitors and are here to stay. They nevertheless need additional support!

The main objective of the present study is to investigate the possibility of supporting Q&A social networks on the web through sentiment analysis. In particular, we examine the characteristics of such networks with respect to the sentiment analysis of the answers offer.

Let's analyze our problem with an example.

I'm a PHP developer trying to find a way to sort an array with items. I'm visiting SO, place my search term "*php array sorting*" and on the screen shot below you can see the results:



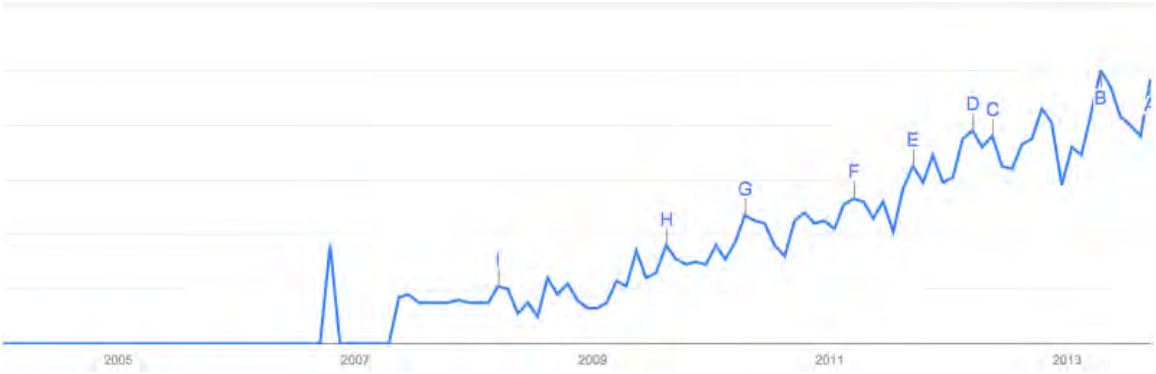
We have **11.344** results and **4** tabs (*relevance, newest, votes, active*) for a simple array sorting.

Which of them shall i chose? Which one is the best and what post answers my question? Those are expected questions, but the problem is that you are in the middle of developing a new product and you want to find the best answer in the shortest time!

The usual process is to open the ones you think are the most relevant, with information you get from SO interface like number of answers and number of votes, but what about the rest of the information that you cannot see on the interface?

How would you feel if you could "**guess**" how users (who already done that process and wrote a comment) found each of those posts interesting or not. Even better how you would feel if you had an **automatic** way to visit each post, analyze the comments users left and give you the ones with the most positive feedback!

Yes I know, that sounds like a "*dream came true*" and its possible using **sentiment analysis**. Despite that this is a convincing answer it doesn't answer the actual question "why to use sentiment analysis"? *Is it because it's trendy*? One can truly say that sentiment analysis has recently gone big. If you will try to search for sentiment analysis on Google Trends, you'll see a huge growth from September 2006 up to the present.



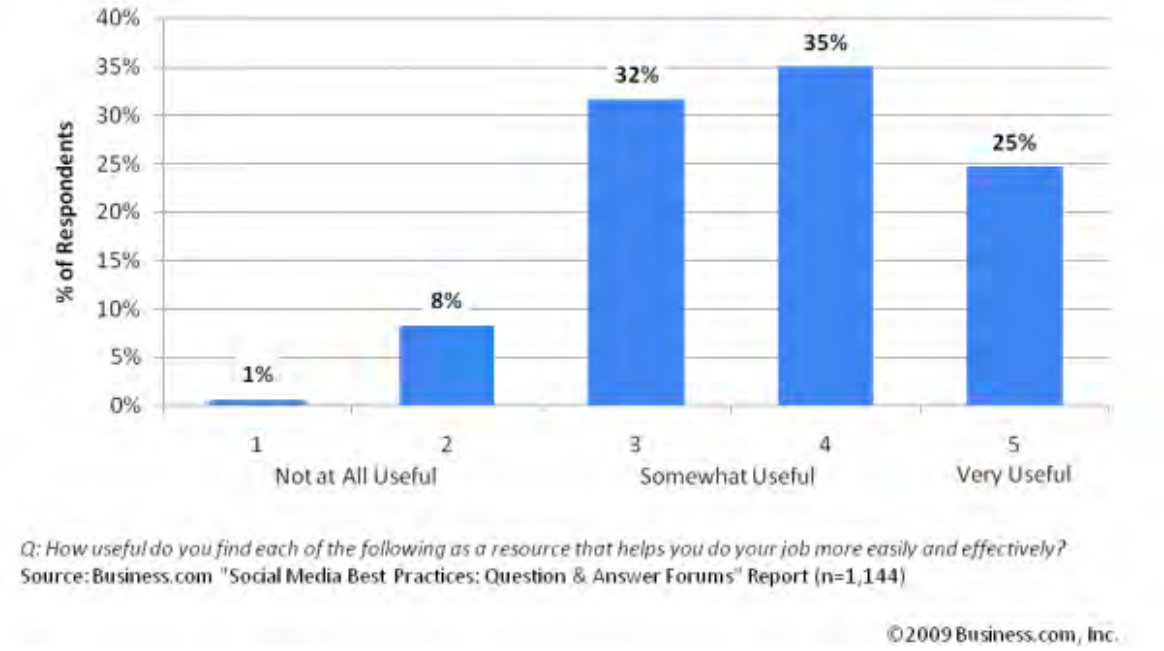
But short answer, is no. I mean sentiment analysis is trendy and major companies try to take advantage of it but that's not the reason we use sentiment analysis. The actual answer to that question is because there is **no other way** to get automatically the information you need from a human text, it's your only option actually.

Another important question that needs to be answered is **why did we select Q&A sites**?

Because they are still famous, useful and they lack of innovation. Whole companies use Q&A sites to promote their products by answering to user's questions using the model:

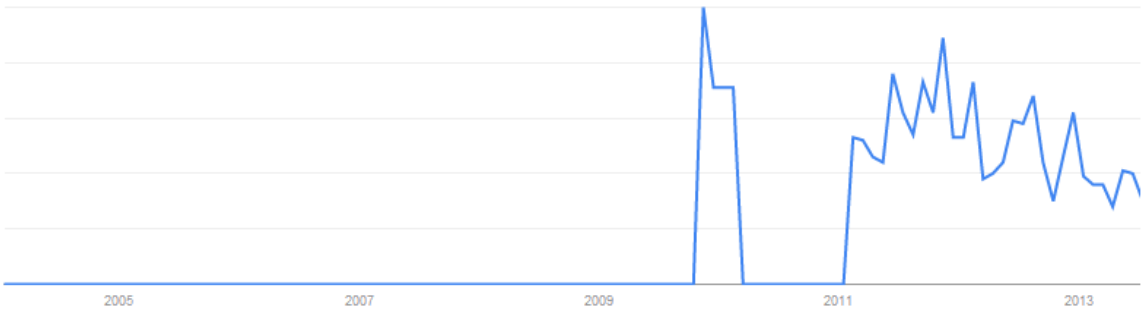
- *Help others.*
- *Build relationships.*
- *Push your products and services when they answer somebody's question or request.*

If you search the web for "*Q&A Sites to Build Your Business & Reputation*" you will find hundreds of presentations and live examples of companies and individuals doing exactly that. Also users take advantage of that and they get some answers for their questions. On the chart below you can see some statistics on usefulness of business in Q&A distribution.



Q: How useful do you find each of the following as a resource that helps you do your job more easily and effectively?
Source: Business.com "Social Media Best Practices: Question & Answer Forums" Report (n=1,144)

But if we search for "*Q&A sites" on* Google Trends we will get the chart below.



We can see that the chart is opposite to SA chart. But why is this happening?

Q&A sites have huge amount of traffic because they have some important **advantages (for users)** compared to the rest of the World Wide Web innovations, for example:

- *You get fast answer to your question.*
- *Get many opinions from different aspects.*
- *Every subject is categorized.*
- *You don't have geographical limitations.*

But they also have a lot of **disadvantages:**

- *The technology used is ancient.*
- *Ugly design.*
- *Zero innovation.*

In general we can say they don't evolve compared to the rest of World Wide Web. They are static and old fashioned without any innovative technologies. It was believed to be a matter of time until they are buried completely from Faceboook, Twitter and Google massive expansions. But even today some Q&A sites are among the web pages with huge amount of traffic. It's obvious they become more and more less famous but this not because they are not useful but because of lack of innovation! Of course there are some exceptions here, for example ChaCha lunched a free mobile answers service which allows users to call or text questions to ChaCha on mobile phones and receive answers within minutes. The company also provides online access to questions and answers at ChaCha.com, and via other social platforms including Facebook and Twitter. That was innovative and ChaCha reports traffic has increased 1000 percent since the firm's January 2008 launch, supplying text answers to more than 150 million questions via mobile devices and the web.

To the best of our knowledge, the present work seems to be the only result that associates Q&A systems with sentiment analysis practices and tools. No similar efforts have been found neither in the scientific literature nor in the web itself. Our system implementation tries to change that, but it’s not perfect at all.

Currently our implementation consumes the **most voted** posts that match the search term from SO, analyses the comments on each of them and creates a ranking for the original post. An important improvement would be to add options so the user can select one of SO options (*relevance, newest, votes, active).*

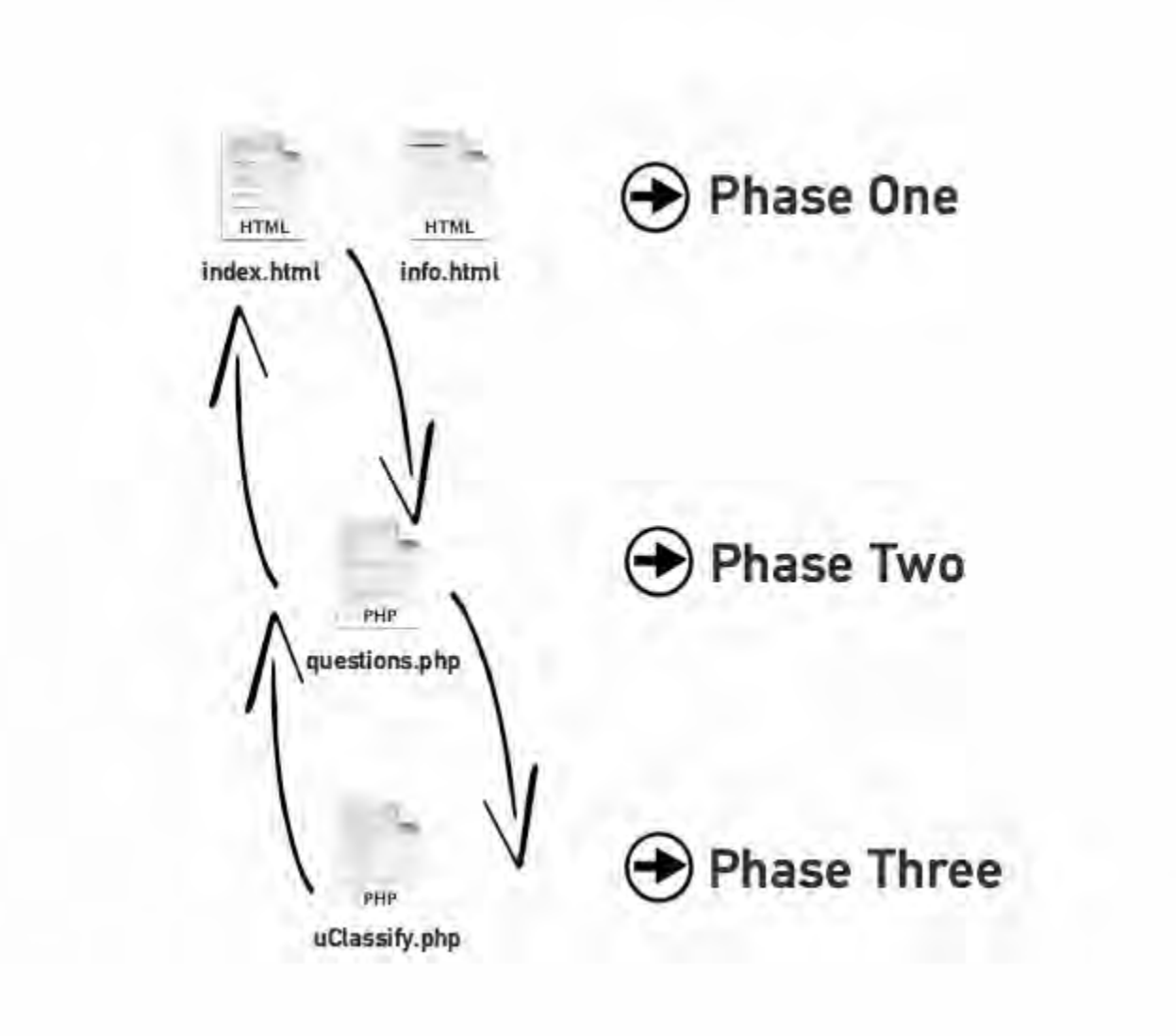
It would be also good addition to have **information about the ranking** of each post in the UI so the user will know what the exact ranking of each post is.

Another important thing is the search speed, the time you hit the "*search button*" until the time you get the actual results is very important for the end users. If it's faster to open the posts in tabs and take your shoot to find the one that answers their question they will probably use that way! To **increase the speed** there are couple of things we can do. We should host sentiment analysis service in our own servers (*this is costly*) and we need powerful machines (*e.g. amazon cloud servers, microsoft azure, google app engine*). The difference with those services is that they actually overtake machines when needed automatically depending on the load of each one.

Last but not least is an automated way to test your system. All modern software products rely on **automation testing** and it is a must do for our system too.

3. System design and implementation

As we can see from the image below the system is composed by many sub-units. In the first level is our html page which is visible by the end user. That is the place where users can start their interaction with the system by inserting their search term and pressing enter.



The search term is passed to the second level of which is a php file. On that phase we make the actual request to SO service using user's search term and get a response from SO service with a bunch of posts and comments. Then we have to "clean" and organize them into arrays.

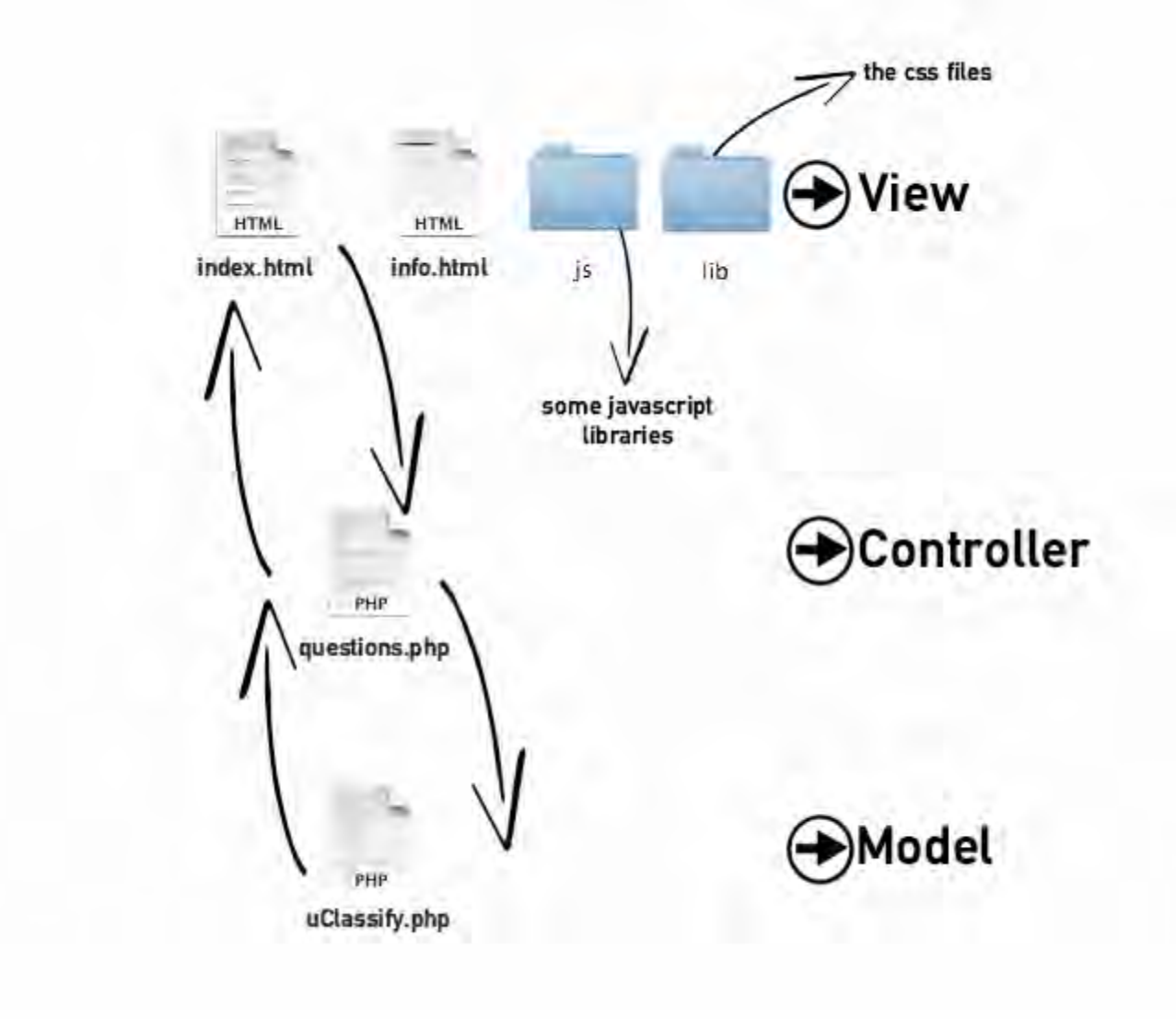
Those arrays are organized again to a single array, that single array goes to another php file. On that phase we actually contact our classifier for classifying that array. When the classification is done we pass back to phase two our results (*again an array with rankings*).

On phase two now we have to separate again the single array to multiple arrays and find the ranking of each post. Then we have to rearrange our posts based on the ranking each post got. The ranking algorithm is simple:

- *+ 1 for positive comment.*
- *- 1 for negative comment.*
- *No action for neutral comment.*

And then again pass the result to phase one and display the results to the end user (*again a rearranged array with posts*).

The system is built using the MVC and you can see the design below for a visual representation.



Below we will explain in more detail each phase. But before we do that we have to analyze some things first.

Our target was to develop a system that would parse user comments (*pure text*) from stackoverflow and "*magically*" would recognize if that text is positive negative or neutral. In other words I had to find a way to recognize pure text and find its meaning.

There are two ways to do that:

- *Using a lexicon with some values for each word (positive negative neutral).*
- *Using a classifier which is more dynamic system that learns on its own.*

In the beginning we implemented the system using lexicon. And that solution had some **advantages**:

- *It was fast enough.*
- *It was simple to implement*

But also had some **disadvantages:**

- *It wasn't accurate enough.*
- *It wasn't dynamic enough (you should update the lexicon manually).*
- *Hard to find the proper lexicon for your language group.*

So I decided to use a classifier instead and not only that but use a service that is dedicated on text analysis classification.

I was searching for a service with the criteria below:

- *Free.*
- *Easy to use.*
- *Good API.*
- *Some good wrappers around it.*
- *With support of huge amount of requests*
- *Open source.*

After two days of googling and registering to different types of services I ended up with uClassify. A descent sentiment analysis that is almost free (*you need to pay if you want to host it on your own server*), easy to use, the API is acceptable, the already developed wrappers are descent, supports a good amount of requests and of course it's not open source (*but that's life deal with it*).

Next step was to study the API and the wrappers I was about to use. The API was straight forward but I had to make the right choice of a wrapper. So I had three options:

- *Use a javascript API for stackoverflow and uClassify and move all logic to the client.*
- *Use php and load with my server.*
- *Use both and load both of them.*

So I made a decision to move all the logic to the server. That decision was made because of two terms basically:

- *Javascript is slow compared to php.*
- *The product should be responsive on slow device too (smartphones and tablets).*

When finished with those critical decision (*right or wrong, that is not the point*) I register to stackoverflow and uClassify services as a developer and obtained my key and I was ready to proceed with my system.

Phase one:

In the index.html file there is an input field where the user can type the search term. When enter pressed the search term goes to a script in the same file. On that script we parse the search term, if empty we alert the user with an error else we forward the input to the questions.php for further analysis. If we get an error from that script we again present the error message to the end user. If we get our response correctly (*an array with our answers*) then we loop through that array and place every element on the DOM. Our answers array is an arranged array with links of each post that has the highest number of positive comments. There are also some fancy scripts there that present some dots when the search is in progress.

Below you can see the javascript code that handles the search term:

```
<!–find function that handles the search input term on click or enter keypress !>

$('#find').click(function(){
                    var amount = $('#amount').val();

                    if(amount==''){
                           // search term is empty
                           alert("Give your search terms, and then Enter!");
                           return;
                    }
                    document.getElementById("results").style.display="block";

                    $.ajax({
                           // post request to php script passes the search term
                           type: "POST",
                           contentType: "application/x-www-form-urlencoded;charset=utf-
8",
                           url: "questions.php",
                           dataType: "json",
                           data: {'amount': amount},
                           beforeSend: function(){
                                  // fancy animation while waiting for results
                                  $("#results").Loadingdotdotdot({
                                         "speed": 400,
                                           "maxDots": 5
                                         });
                                  $('#results').height(18);
                           },
                           success: function(data){
                                  // getting the result after rearranging
                                  // this code is ommited
                           }
                    });
             });
```

Phase two:

Step one: The search term goes to the questions.php. First we have to encode by changing the spaces with the corresponding URL character which is "*%20*". Then we have to merge it with the rest of URL and send it to stackoverflow API. The rest of the URL consists from the developer key, the name of the particular service and some other information described with more detail in case studies. Then we send the request to SO and when we get the answer we decompress it and convert the JSON response to array. On that step we have some checks and if we got zero results or not enough we split the terms and search again for more results. In the end of that step we have an array with our relevant posts and we loop through that array and call the getAnswers() functions passing the link of each post.

Step Two: In the getAnswers() function we need to send a request again to SO API and get the comments for each of those posts. We again get a compressed response that we need to decompress and convert it to array from JSON. Then we have to loop through that array, extract the comments and store them in a new array. In the end of that step we loop through the comments and call the function removeTags() which will "*clean*" the comments from unwanted characters like <code> blocks etc. When done we check if the array of comments is empty or not (*some posts might not have any comments*) and forward that array to classMe() function.

Below is the source code of getAnswers() function:

```
/**
        *     Batch operation of getanswers method
        *
        *     @param $question_link string with a link of a post

**/
function getanswers($question_link){

       $serial = explode("/",$question_link);  //we need to extract the question id

       $question_link
="compress.zlib://https://api.stackexchange.com/2.0/questions/".$serial[4]."/answers?orde
r=desc&sort=activity&site=stackoverflow&filter=!*MpApC8gG3bbJ733&key=q0NPALjE0r1MKsMeQV";

       $result = file_get_contents($question_link);
       //need to be decoded before we can use it
       $decoded_result = json_decode($result);

       //decoded_result has all the answers from a question, comments so we can get them
and parse them later on!

       // store the text from answers
       $answer_body = array();
       // store the ids from answers
       $answers_ids = array();
       // store the text from comments
       $comments_body = array();
       // store the ids from comments
       $comments_ids = array();

       // loop through the answers of the post and save comment's ids and body!
       for($l=0;$l<sizeof($decoded_result->items);$l++){
              array_push($answer_body, $decoded_result->items[$l]->body);
              array_push($answers_ids, $decoded_result->items[$l]->answer_id);
              for($k=0;$k<sizeof($decoded_result->items[$l]->comments);$k++){
                     array_push($comments_body, str_replace("&hellip;","",
$decoded_result->items[$l]->comments[$k]->body));
                     array_push($comments_ids, $decoded_result->items[$l]->comments[$k]-
>comment_id);
              }
       }

       //now it's classification time for every comment gathered.

       //some questions might not have answers or comments yet so no need to parse them
       if(sizeof($comments_body) !=NULL){
              //we call classime to find wich are pos and neg!
              return(classime($comments_body));
       }
}
```

Step Three: In classMe() we create an instance of the class uClassify.php and set the developer keys we got by registering to uClassify web service. Then we pass the array of posts for classification. When we get

the response from the service we forward the result (*an array with positive, negative and neutral percent for each comment*) to calculator() function. In case we get an error we show the error message to the end user.

Below is the source code of classMe() function:

```
/**
        *     Batch operation of classime method
        *
        *     @param $text Array with text in every cell and stores the pos and neg of
every text on every cell in $resp

**/

function classime($text){
       $uclassify = new uClassify();

       //those are the uclassify keys we need them so server doesn't kill us!
       $uclassify->setReadApiKey('lOHDZf9vIDqsDqP98Q9bPsasx');
       $uclassify->setWriteApiKey('fqFVKgDoALknk1ozyrte1d9BTjU');

       try {
              $resp = $uclassify->classifyMany($text, 'Sentiment', 'uClassify');
              return(calculator($resp));
       } catch (uClassifyException $e) {
              die($e->getMessage());
       }
}
```

Step four: In calculator() function we loop through the array with the percent of each post and we create the ranking for each post! This is done with the algorithm below:

- *+ 1 for positive comment.*
- *- 1 for negative comment.*
- *No action for neutral comment.*

Below is the source code of calculator() function:

```
/**
        *     Batch operation of calculator method
        *
        *     @param $posneg Array of positive and negative values and makes the sum of
them
**/

function calculator($posneg){
       $temp_pos=0;
       $temp_neg=0;

       // one counter and inc dec on every case
       $total_pos_neg=0;

       for($cn;$cn<sizeof($posneg);$cn++){
              //we get the negative value
              $temp_pos = $posneg[$cn]['classification'][0]['p'];

              //we get the positive value
              $temp_neg = $posneg[$cn]['classification'][1]['p'];

              if($temp_neg<$temp_pos){
                     $total_pos_neg++;
              }else if($temp_pos<$temp_neg){
                     $total_pos_neg--;
              }
       }

       //return the rank calculated above using all the comments of a question!
       return $total_pos_neg;
}
```

Phase three:

From phase two, step three we create an instance of uClasiffy.php class and we are using the clasiffyMany() function. In the beginning we have some checks for empty strings and then we encode the text to base64 format before sending for classification. Then we create and new xml request and attach some information on that request:

- *Username for login in the system.*
- *Password for login in the system.*
- *Classifier name.*
- *Array for classification.*

Then we send the request to the service using post request. When we get our response we check for error messages and if everything is okay we return our result.

Below is the classifyMany() method source code.

```
/**
        *     Batch operation of Classify method
        *
        *     @param $texts Acutal Array of texts that needs to be classified
        *     @param $classifierName      Name of the classifier against which the array
of texts needs to be classified
        *     @param $username     Name of the user, under whom the classifier exist. Use
this option if you need to access other's published classifiers
        **/
       public function classifyMany($texts = array(), $classifierName = null, $username =
null) {
              if(count($texts) < 1) throw new uClassifyException("What should be
classified? No text seems to be specified!");
              if(empty($classifierName)) throw new uClassifyException("How should the
text be classified? No ClassiferName seems to be specified!");

              $this->buildXMLRequest();

              $_id = 0;
              foreach($texts as $text) {
                     $this->texts[] = base64_encode($text);
                     // Setting the Ids for the text
                     $this->textIds[] = 'Text' . ($_id++);
              }
              $texts = $this->xmlRequest->createElement('texts');
              $readCalls = $this->xmlRequest->createElement('readCalls');
              if(empty($this->read_key) || !isset($this->read_key)) throw new
uClassifyException("Read API Key is not specified.");
              $readCalls->setAttribute('readApiKey' , $this->read_key);
              $this->uclassify->appendChild($texts);
              $this->uclassify->appendChild($readCalls);
              $_counter = 0;
              foreach($this->texts as $textBase64) {
                     // Creating the textBase64 element tags
                     $textb = $this->xmlRequest->createElement('textBase64',$textBase64);
                     $texts->appendChild($textb);
                     $textb->setAttribute('id', $this->textIds[$_counter]);

                     // Creating the classify tags for the same textBase64 elements
                     $classify = $this->xmlRequest->createElement('classify');
                     $readCalls->appendChild($classify);
                     $classify->setAttribute('id','Classify' . rand(0, getrandmax()) .
time());
                     $classify->setAttribute('classifierName',$classifierName);
                     $classify->setAttribute('textId',$this->textIds[$_counter]);
                     if(!empty($username)) $classify->setAttribute('username',$username);
                     $_counter++;
              }

              $xr = $this->xmlRequest->saveXML();
              $resp = $this->postRequest($xr);

              if(!$resp) {
                     throw new uClassifyException("Invalid data sent by the server!");
              } else {
                     return $this->parseClassifyResponse($resp);
              }
       }
```

After all phases are executed we go back to phase two and rearrange the items based on the final score of each post.

Below you can see the sorting part code:

```
/**

/**      Arranging the items from high to low ranking values

/**

array_multisort($final_answers, SORT_DESC,$term_asnwers);
```

Finally we go back to phase one recursively and display the results to the end user.

That is "*who things work*" in detail but that doesn't mean our system is ready at all. When finished the system implementation it was time for validation. There are two ways to test your sentiment analysis system and actually your best option is to use both of them.

- **The manual way:** At least in the beginning you need to train your system and monitor how it responds to your input. You need to create specific classifiers according to your target group (e.g. technology, movies, music etc.). You're working with natural language, with material you can understand directly, and it'll be pretty clear whether the tools you're trailing are doing a good job. You also need to do that while you are building / improving your system and when you finish it. You should also improve it sentiment analysis is a new field so if you want to be updated you **SHOULD update and evolve your system.** That is a painful process and you need to repeat the manual testing process to prevent regressions on your system.

But the manual way is not enough...

- The automatic way: The high-level idea is to use a language classification framework to do two classification tasks: separating subjective from objective sentences, and separating positive from negative parts of text. You can use two data sets one with **polarity** text parts (positive, negative and neutral) already categorized and ready to apply on your system for testing purposes. Usually that data set is pretty huge (*1.000 or more text parts*). The second data set is the **subjective** data set, you are verifying your system using two different services as input text, for example if you want to sentiment analyze movies reviews you will test your system with input from two services like *imdb* and *rotten tomatoes* and compare the results between each other for the same movies. That's why the second one is called subjective because the actual reviews are subjective and the two systems might have different results for the exact same movies. That's why the latter is harder to apply on every system because you might not have two services with the same subject to test them.

We used only the manual way for verifying our system and would be a must addition automation testing too. Most common testing techniques are the manual way and the polarity data set for automation systems!

Usually online sentiment analysis services have their own data sets to train and test your system according to your sentiment target. But it's also good idea to use external and third party tools to improve and verify that your system is working as expected.

When all the processes had finally finished (*design, development and testing*) it was deployment time (*the fun part*). I had already a server but decided to use a dedicated one for that particular project. So again I was looking for something with the criteria below:

- *Free.*
- *Configurable.*
- *With huge amount of traffic support (just in case).*
- *Without ads.*

So I decided to use GRNET. You can set up your own dedicated server and configure your system from scratch (*operating system, ram, cpu power, hard disk space etc.*) and it’s free if you are a student (*in Greek university*). So when finished with account creation and setup of the system, I installed php and apache and was ready to upload my creation.

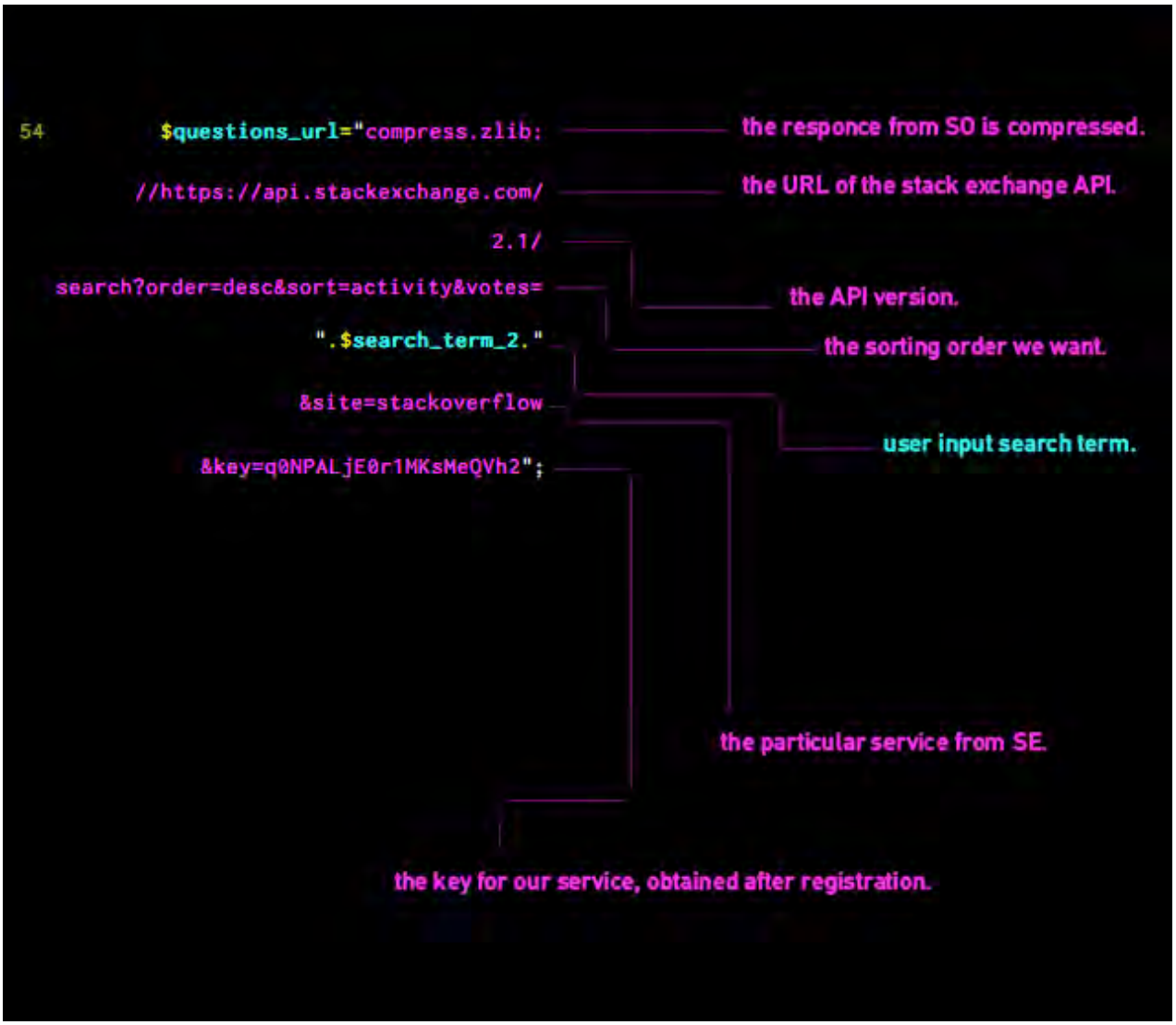
Last step was to try the service online and luckily the deployment was successful and the service was up and running.

4. Case studies

On that part I want to explain how we can extend the system to the rest stack exchange services. This part is very simple because all stack exchange sites have the same API and actually the only thing we need to change is one line of code.

compress.zlib://https://api.stackexchange.com/2.0/search?order=desc&sort=votes&intitle=".$search_term."&site=stackoverflow&key=q0NPALjE0r1MKsMeQVh2

For more details what that line of code does you can see the image below.



From that line of code we need to change the word "*stackoverflow*" with the name of stack exchange service we want to use. It's that simple...

To extend the service to consume other services is not so simple (*one line*) because they have different API and different JSON structure, but it's not that difficult too except from that line of code we will need first of all to register for o key on the rest services so we can have increased number of requests to the service. Except from that we will need to map our local answers arrays to match the format of the JSON answer of each service. The rest of the code will stay intact and our service will be ready to use!

ChaCha has recently launched a free sms system to put their API into the hands of the developer community, and they are supposed to launch a web-based answer service too. Currently (*October 2013*) the only option is the sms system. If you are a developer, you have to wait until they publish some more information on developer.chacha.com.

You can still use the sms system if you want to build a basic application for mobile phones but the limitations compared to web-based service are huge!

Other thoughts are to extend the system using two or more ways for validation. It would be good to verify somehow that our sentiment analysis system is working correctly and the validation results are as expected to be. That can be achieved by using a classifier and a lexicon so you can validate the in between results with each other before presenting to the end user!

5. Concluding remarks

In this paper we briefly reviewed the relatively new sentiment analysis concept focusing on its practical considerations. We then described the current on-line Q&A systems identifying their characteristics and in particular those that are may be associated with sentiment analysis efforts. We finally, proposed and implemented a system that has the potential to increase the precision of Q/A activities. We focused on one of the most popular Q/A system and developed a simple sentiment engine. Our initial experimentation proofs that our objectives are realistic and valid.

The problem of supporting web Q&A systems with sentiment analysis has by no means solved. This paper contains initial results of an ongoing effort. These results should pave the way towards a more comprehensive and more convincingly efforts and perhaps commercial products.

6. References

Cheng, H. T. S. T. X. (2009). A survey on sentiment detection of reviews. *Expert Systems with Applications*, *36*(7), 10760–10773. doi:http://dx.doi.org/10.1016/j.eswa.2009.02.063

Feldman, R. (2013). Techniques and applications for sentiment analysis. *Communications of the ACM*, *56*(4), 82. Retrieved from http://dl.acm.org/ft_gateway.cfm?id=2436274&type=html

Liu, B. (2012). *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.216.5533

Liu, B., Hu, M., & Cheng, J. (2005). Opinion observer: analyzing and comparing opinions on the Web. In *Proceedings of the 14th international conference on World Wide Web - WWW '05* (p. 342). New York, New York, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?id=1060745.1060797