Detecting Trending Topic on Chatter

JEAN-BAPTISTE CHAUBET

Master of Science Thesis
Stockholm, Sweden 2011
Detecting Trending Topic on Chatter

JEAN-BAPTISTE CHAUBET

Master’s Thesis in Computer Science (30 ECTS credits) at the Degree Program in Information and Communication Technology
Royal Institute of Technology year 2011
Supervisor at CSC was Dilian Gurov
Examiner was Johan Håstad

TRITA-CSC-E 2011:092
ISRN-KTH/CSC/E--11/092--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc
Abstract

The amount of posts on social network is overwhelming: for example Twitter has more than 50 millions posts a day. It has become crucial to be able to sort them. By detecting trending topics, which are topics the most discussed on a social network, we allow the user to instantly know what is happening in the network and if he is interesting in one topic, he can get access to all the posts related to this topic.

In this work we present and compare different algorithms to detect trending topics. Our approach is to compute similarities between posts and then to find clusters in the graph of “similarities” using clustering algorithms.
Foreword

This thesis was written in accordance with the requirements for the degree of Master of Science at Royal Institute of Technology in Stockholm. All research, implementation and analysis were carried out between September 2010 and March 2011 at the office of Salesforce.com in San Francisco, California and in Stockholm, Sweden. Salesforce.com is a cloud computing company that distributes business software.
Acknowledgment

I am very grateful for the cooperation of everybody who helped me on this thesis and through all my academic life specially my supervisor in Salesforce Jari Koister for his permanent support and for offering the opportunity of doing this interesting project. I would also like to thank all the members of the Salesforce Discovery Team both in San Francisco and in Stockholm for their precious help.

Finally I would also like to thank my KTH supervisor Dilian Gurov and my KTH examiner Johan Håstad.
# Contents

Chapter 1 Introduction

1.1 Chatter ................................................................................................................ 1

1.2 Trending topic ........................................................................................................ 2

1.3 Goal ....................................................................................................................... 3

1.4 Contribution .......................................................................................................... 4

Chapter 2 Background

2.1 Algorithms used today to detect trending topics ................................................. 5

2.1.1 Facebook ......................................................................................................... 5

2.1.2 Twitter: algorithm based on frequency analysis ............................................. 6

2.2 Clustering algorithms ........................................................................................... 14

2.2.1 Hierarchical algorithms .................................................................................. 14

2.2.2 Partitional algorithms ..................................................................................... 15

2.2.3 Markov Clustering (MCL) ............................................................................. 16

2.2.4 DBSCAN ......................................................................................................... 19

Chapter 3 Our algorithms

3.1 Frequency-based algorithm .................................................................................. 22

3.1.1 Based on Twitter algorithm ........................................................................... 22

3.1.2 Inquiry about what make the Swedish people happy ..................................... 22

3.1.3 How will we use it? ....................................................................................... 23

3.1.4 Last step in our algorithm ............................................................................ 23

3.2 Using clustering algorithms ................................................................................ 23

3.2.1 Jaccard Similarity ......................................................................................... 23
3.2.2 Comparison using a global ordering .............................................................. 24
3.2.3 Clustering algorithms ................................................................................ 24
Chapter 4 Experiment .......................................................................................... 25
  4.1 Data collection ............................................................................................... 25
  4.2 Preprocessing of the data ............................................................................. 25
    4.2.1 Time interval ......................................................................................... 25
    4.2.2 Stop words removal ............................................................................. 25
  4.3 Evaluation of the results of the algorithms ..................................................... 26
Chapter 5 Results and analysis .......................................................................... 29
  5.1 Results ........................................................................................................... 29
  5.2 Frequency analysis algorithm ...................................................................... 30
  5.3 DBSCAN vs. MCL ....................................................................................... 32
    5.3.1 Complexity ............................................................................................. 31
    5.3.2 Difference in classification .................................................................... 32
Chapter 6 Conclusion and future work .............................................................. 41
  6.1 DBSCAN chosen for the new version of Chatter .......................................... 41
  6.2 Improvement .................................................................................................. 41
    6.2.1 Similarities computation ...................................................................... 41
    6.2.2 Name of the topic ................................................................................ 42
Bibliography ......................................................................................................... 43
List of Figures

Figure 1.1: An example of a Chatter page ................................................................. 2
Figure 1.2: Twitter trending topics, a way to know what is happening in the world........ 3
Figure 2.1: Booshaka, a website to see Facebook’s trending topics .................................. 6
Figure 2.2: Example of Hashtags .................................................................................. 6
Figure 2.3: Duplicate trending topics .............................................................................. 8
Figure 2.4: Proof that Twitter uses td-idf: Nadal is not a trend even if it appears much more often than Clic ........................................................................................................... 11
Figure 2.5: An usual Friday on Twitter, Friday is a trending topic .................................... 12
Figure 2.6: An usual beginning of month on Twitter, February is trending topic ....................... 13
Figure 2.7: Twitter has some problems to detect trigrams or more as trending topics ........ 14
Figure 2.8: An example of dendrogram ........................................................................ 15
Figure 2.9: An example of column stochastic matrix .................................................. 16
Figure 2.11: Matrix of similarities ............................................................................... 17
Figure 2.10: Graph of similarities ............................................................................... 17
Figure 2.12: Stochastic matrix ..................................................................................... 18
Figure 2.13: After an expansion step ............................................................................ 18
Figure 2.14: After an inflation step .............................................................................. 18
Figure 2.15: Equilibrium state ..................................................................................... 19
Figure 2.16: DBSCAN is started on a random node. .................................................... 20
Figure 2.17: A core node has been found, a new cluster is started. ................................. 20
Figure 2.18: Final cluster ........................................................................................... 20
Figure 4.1: The website we made to test our algorithms............................................... 27
Figure 4.2: Statistics of answers ............................................................................... 28
Figure 5.1: Percentage of correct classification of the different algorithms ....................... 29
Figure 5.2: Frequency analysis algorithm display all the posts containing one word whether it is the same topic or not .................................................................................................................. 31
Figure 5.3: Example of graph where it is better to use MCL ........................................... 33
Figure 5.4: Result of the clustering with MCL .............................................................. 33
Figure 5.5: Result of the clustering with DBSCAN ................................................................. 33
Figure 5.6: DBSCAN can merge different topics .................................................................... 34
Figure 5.7: Example of graph where it is better to use DBSCAN. The node on the left should not
belong to any topic, but with MCL it will be part of the cluster on the right............................. 35
Figure 5.8: Graph obtained after the similarities computation................................................. 37
Figure 5.9: Execution time of MCL with different number of machines................................. 39
Figure 5.10: Execution time of DBSCAN with different number of machines......................... 40
List of Tables

Table 5.1: Time of execution for DBSCAN and MCL .............................................................. 32
Table 5.2: Number of neighbors for each node ..................................................................... 37
Chapter 1 Introduction

1.1 Chatter

Chatter (1) was released in June 2010 by Salesforce.com. It was designed to be “the Facebook for enterprise”. The different social networks have changed dramatically our way to stay aware of the world around us. With Facebook, you can easily know what is happening among your friends; with Twitter, you can know what is currently happening in the world, Chatter is meant to allow people to stay aware of what is happening in their work in a secure way. Chatter gives you a real-time feed with all the updates of your coworkers, documents you are working on, group. On Figure 1.1, it can be seen an example of a Chatter page. You can in particular see two examples of the use of Chatter: the user can ask a question and get an answer quickly and he can also share documents and get feedback about it. This is more convenient and more efficient than exchanging many e-mails.
1.2 Trending topic

Chatter has competitors and has to keep innovating. One of the next challenges for Chatter is to implement the detection of trending topics. A trending topic is a topic discussed recently and a lot on a social network; this feature allow a user to know at a glance what is currently happening in the network. On the Figure 1.2, an example of Twitter trending topics can be seen. This screenshot was made only a few minutes after the earthquake in Japan, and just by looking at Twitter trending topics you can learn that something important just happened in Japan.
On Twitter, trending topics allow a user to stay aware of what is happening in the world. On Chatter, it could help an employee to know the top topics in his company. For example he can learn about a conference, a project deadline or a release.

1.3 Goal

As we said, Chatter is a network inside a company. On Chatter, the amount of posts a day is of the order of several thousand which differs a lot from the 50 millions Twitter posts a day. That is why there is hope to develop more complex algorithms than the one used by Twitter.

We suppose we have the Chatter posts from one company, our goal is to find the trending topics and to find the posts corresponding to each of these topics.

We want to design, implement and compare different algorithms which achieve this goal and eventually to keep the best one for the next release of Chatter. In this thesis, even if we take into account the speed of the different algorithms, our main criterion to decide the best algorithm is the quality of its results.
1.4 Contribution

We used two different types of algorithms to achieve the goal mentioned above.

The first one was based on word-frequency. The main tasks in this algorithm were:

- To look and analyze Twitter trending topics to be able to know which algorithm is used on Twitter.
- To implement a similar algorithm with some new features.

The second type of algorithms is completely new to compute trending topics and is the main contribution in this thesis. This type of algorithms is a more semantic way to detect trending topics. It is based on a graph approach and composed of two steps:

- During the first step, we build a “similarities” graph where each vertex corresponds to one post and each edge to the similarity between the two posts.
- During the second step, we use clustering algorithms to find clusters in the graph. The biggest clusters correspond to the trending topics.

In these algorithms, the main tasks were:

- To find a way to compute similarity between two posts.
- To choose among the different clustering algorithms already existing and find the most suitable to our problems and then to implement them.
- To merge these two previous steps together.

We also implemented a website to determine which algorithm was the best for our problem; this website displays two posts we had classified in the same topic and asks the user if they were indeed belonging to the same cluster.
Chapter 2 Background

In this chapter we present several algorithms: the algorithms already used today to detect trending topics and some existing clustering algorithms which have never been used to detect trending topics but which are necessary for our implementation.

2.1 Algorithms used today to detect trending topics

Facebook and Twitter algorithms are not open source; but by looking at the trending topics obtained, we can learn a lot about their algorithms. We cannot, however, be absolutely certain and all facts stated about the trending finding algorithms used by these famous web-sites are only educated guesses.

2.1.1 Facebook

Facebook has not implemented trending topic yet: when you log on facebook, you cannot know what is trending.

However booskaka.com can be used. This website classifies the posts among preexisting topics such as sport, politics, and movies. The algorithm to classify the posts is quite simple: it is assumed that for example Barack Obama will only speak about politics or Roger Federer only about sport and then all the posts posted by Obama are classified as belonging to the topic politics.

This algorithm has two main problems:

- It can only display the posts of the famous people and all those people should have been manually classified into one topic. It can lead to incorrect classification. For example, as you can see on the Figure 2.1, when Ryan Newman (a professional driver) was speaking about his new baby, it was classified as sport.

- The set of topics is predefined and is not precise (they can be seen on the left part of the figure 2.1), so it will never allow a user to discover brand new events which are happening right now like the demonstrations in Tunisia or in Egypt.
2.1.2 Twitter: algorithm based on frequency analysis

2.1.2.1 Hashtags

Hashtags have existed since 2007 and were the first way to compute trending topics. To make a hashtag, a user has to add a prefix “#” before the words he considers as relevant (Figure 2.2).

AFP: #Libya rebels reject proposed transition under #Gaddafi son.

Figure 2.2: Example of Hashtags

One who wants to know more about Libya just need to search for #Libya and get all the posts containing this hashtag.

To compute trending topics, an easy algorithm can be used: the frequency of each hashtag is computed and the most frequent ones become trending topics.
However hashtags can only work if users use them, and many users do not like using them. Hence, even if hashtags are still partially used on Twitter to detect trending topics, Twitter has also implemented other algorithms to improve them.

2.1.2.2 Trending WORD

As stated above, the Twitter algorithm is not open source, so we can only guess about it. It seems that to detect trending topics and to name them, instead of only counting the frequencies of hashtags, Twitter counts the occurrence of every word and the most frequent ones become trending topics. So instead of being called “trending topic” in Twitter, it should rather be called “trending WORD”.

Different mistakes in Twitter trending topics made us think that:

- A word which does not make any sense on its own can become a trending topic. For example “Lanta” has been a trend in France. But “Lanta” alone does not mean anything at all. It was just a trending topic because of the French reality TV-show “Koh Lanta”.

- A word which is not a topic can become a trending topic because it appears frequently but in completely different contexts. For example “Etats-Unis” which means “United States” in French is quite often a trending topic in France, because it appears quite often in the news: life expectation which gets lower, Assange, world champion in fussball. So “United States” was a trending topic even if it is not a topic. Another example: once Jean-Pierre (a French first name) was trending in France even it was a completely different Jean-Pierre in each post. It would be exactly the same if Magnus was a trending topic in Sweden.

- Two words (or even more) which refers exactly to the same topic can be a trend simultaneously: for example both Prince William, Kate Middleton (his future wife), royal wedding and royal engagement were trending topics at the same time when Prince William announced he was getting married (Figure 2.3). Another example,
“Jugendmedienschutz-Staatsvertrag” and “JMSTV” were both trending topics in Germany even if “JMSTV” is the acronym for “Jugendmedienschutz-Staatsvertrag”. In some cases the ten Twitter trending topics can correspond to only two different topics. For example in the right part of the Figure 2.3, all the topics correspond either to the cyclone in Australia or to the troubles in Egypt.

![Figure 2.3: Duplicate trending topics](image)

---

**Term frequency - inverse document frequency (Tf-idf)**

**Removal of stop words**

The most frequent words carry often very little information. Hence using raw frequency could lead to have trending topics such as “a”, “is”, “at”. So we need to remove them directly from our count the most common words which can never be trending topics.

**Tf-idf**

However, the problem remains for words which occurs frequently all the time even if they are not a trend or do not correspond to any topic, but they can still correspond to a trending topic when they occur even more frequently than usually.
For example “United States” occurs very frequently and could be a trending topic all the time if Twitter was using raw frequency, however of course “United States” cannot be put in our stop words list because it can still be a trending topic sometimes. In fact, it means that a tool is needed to avoid detecting POPULAR topics as trending topics.

Hence Twitter uses a weight called Term Frequency-inverse document frequency. It is used to evaluate how important a word is to a corpus. In our case, it increases proportionally to the numbers of times the word appears in the recent posts and it decreases if the word appears frequently in the corpus (the corpus can be for example the last 6 months posts).

\[
tf_i = \frac{n_i}{N_i}
\]

Where \(n_i\) is the number of times the word occurs in the recent posts and \(N_i\), the number of times the word occurs in all the last 6 months posts.

Twitter uses the Td-idf in their algorithm to detect trending topics. It can be proved by different ways:

- First, it appears quite often that words which are not in the dictionary and which had never occurred before on Twitter become easily trending topics. For example, “tYp3 LyK tHi5” which stands for “type like this” became a trending topic on Twitter probably because of course these “words” had never occurred before on Twitter.

  Or “Egyption” instead of “Egyptian” became trending when wikileaks did a spelling mistake.

- When there is a game between two teams or players, it often happens that the less famous one becomes a trending topic instead of the famous one. For example on the Figure 2.4, a Twitter screenshot done just after a victory of Nadal against Cilic (less well-known tennis player), you can see that “Cilic” became a trending topic in Spain but not “Nadal” because he occurs really often in spanish Twitter posts. This screenshot was
done just after a game between Island and Spain and you can see that “Islandia” is trending topic but not “España” for exactly the same reason.
Figure 2.4: Proof that Twitter uses td-idf: Nadal is not a trend even if it appears much more often than Cilic.
Using Td-idf to weigh the frequency of each word is a really good way to avoid popular topics to be all the time trending topics, but it still has some problems, for example to avoid REGULARLY popular topics to be the trending ones. For instance Friday is a trending topic almost every Friday on Twitter (Figure 2.5) and it cannot be prevented by using Td-idf because Friday is a really popular topic but only on Friday. It is the same at the beginning of each month (Figure 2.6).

Figure 2.5: An usual Friday on Twitter, Friday is a trending topic
2.1.2.3 Bigrams, trigrams, and more

In order to avoid the last problem, Twitter merges two words when they appear frequently next to each other: that is why Prince William is only one topic and not separated as one topic Prince and another one William. But Twitter used to do it only for up to 2 words, so we were getting some weird topics, for example when Mark Dailey died, RIP Mark and Dailey were both trending topics, even if in all the posts they appeared as RIP Mark Dailey. Another example can be seen on Figure 2.7 where Twitter split “chevaliers des Arts et des Lettres” into two topics. However it seems that Twitter solved this problem since 2011, because there are now some longer trending topics such as “Happy birthday Nina Dobrev”. But the problem for Prince William, Kate Middleton and royal wedding still remain because these words do not appear next to each other in the posts.
2.2 Clustering algorithms

In this thesis, we want to try a more semantic way to detect trending topics. This approach is based on clustering algorithms. We will now go through different already existing clustering algorithms that we will use in our implementation.

2.2.1 Hierarchical algorithms

Hierarchical algorithms are the simplest algorithms. They choose a distance between clusters and they create a hierarchy of clusters by merging in each step the two clusters which are the closest.

The results of the algorithm will be given through a tree called dendrogram as it can be seen in Figure 2.8.

Figure 2.7: Twitter has some problems to detect trigrams or more as trending topics.
Unfortunately this type of algorithm is not well adapted for our case because there are no hierarchies between the different topics and also it will classify all the posts into one cluster even if they do not belong to any topic.

2.2.2 Partitional algorithms

Partitional algorithms are algorithms who determine all the clusters at the same time. They can be really powerful. The probably most famous one, K-means allows clustering quickly. Unfortunately you need to know a priori the number of clusters and like for hierarchical algorithms all the posts will be classified.

There are other partitional algorithms such as quality threshold clustering algorithm where you do not need to know the number of clusters a priori but you need to define the maximum diameter for a cluster.
2.2.3 Markov Clustering (MCL)

2.2.3.1 Stochastic matrix

A stochastic matrix (also known as Markov matrix) is a matrix used to describe the transitions of a Markov Chain. There are two types of stochastic matrix:

- Row stochastic matrix is a square matrix where each entry is a positive real-number and the sum of each row is equal to one.
- Column stochastic matrix is a square matrix where each entry is a positive real-number and the sum of each column is equal to one.

\[
\begin{pmatrix}
.2 & .6 & .35 \\
0 & .1 & .15 \\
.8 & .3 & .5 \\
\end{pmatrix}
\]

Figure 2.9: An example of column stochastic matrix

In this thesis, only column stochastic matrices are used and they will be called simply stochastic matrix.

2.2.3.2 Description

This algorithm was invented by Stijn van Dongen (2) (3); the idea is really simple and ingenious: a node inside a cluster has much more connections towards nodes located inside the cluster than to nodes located outside. Hence a random walk will have larger chance to stay inside one cluster than to go to another one. That is why we will compute random walk inside the graph. Before starting the algorithm, we normalize the similarities matrix to make it stochastic (Figure 2.12). The column \( j \) of the matrix corresponds to the different probabilities to go from the vertex \( j \) to the other nodes. By doing this, the probability to go from one node to another is proportional to the similarity between them. One step of random walk is called an expansion step (Figure 2.13). Computing an expansion step is done by computing the square of the matrix.

Between each expansion step, an inflation step is done (Figure 2.14): the Hadamard power of the matrix is computed. It means each entry of the matrix is raised at the power \( r \), which is a parameter bigger than one. Then the matrix is normalized again. As \( r \) is bigger than one, this step will increase the high probability and decrease the low probability and thus the algorithm will converge faster. The parameter \( r \) can be any real number greater than one. It influences the result of the algorithm. The bigger \( r \) is, the faster the algorithm will converge and the more granular the clusters will be. The choice of the parameter depends of course of the dataset. The
value has to be chosen through experimentation. In his paper (3), Stijn van Dongen recommends to start with a set of value 1.4, 2 and 6. In our work, we decide to use the value 2 for \( r \).

At the end of each inflation step, each value of the matrix smaller than a given threshold \( \text{min} \) is set to zero. \( \text{min} \) is another parameter; in our work we used \( 10^{-3} \) as threshold.

Expansion and inflation steps are alternated until reaching an equilibrium state (Figure 2.15).

### 2.2.3.3 Pseudo-code

<table>
<thead>
<tr>
<th>Algorithm 1: MCL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INPUT:</strong> ( M ) //( M ) is the matrix of similarities</td>
</tr>
<tr>
<td>( M ).normalize()</td>
</tr>
<tr>
<td>while ( (M \text{ is changed}) ) do</td>
</tr>
<tr>
<td>( M=M*M ) //we compute one step of random walk</td>
</tr>
<tr>
<td>( M).inflate() //we compute Hadamard power of the matrix</td>
</tr>
<tr>
<td>( M).normalize() //make ( M ) stochastic</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

### 2.2.3.4 Example

![Graph of similarities](image1)

![Matrix of similarities](image2)

MCL algorithm uses the matrix of similarities. A similarity between a node and itself (diagonal element) is equal to one.
Each column is **normalized**, to obtain a stochastic matrix; thus the element \( m_{ij} \) of the matrix corresponds to the probability of walking from the vertex \( j \) to the vertex \( i \).

To simulate a **random walk**, the square of the matrix is computed.

The Hadamard power of the matrix is computed (here \( r=1.5 \)) and the values below the threshold (here 0.05) are set to 0; then the matrix is normalized again.
After a few iterations, an equilibrium state is reached. Here it can be seen that there are 2 clusters: one with the points 1, 2 and 3 and one with 4, 5 and 6.

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

*Figure 2.15: Equilibrium state*

### 2.2.4 DBSCAN

#### 2.2.4.1 Description

DBSCAN which stands for Density-Based Spatial Clustering of Applications with Noise is a density-based clustering algorithm (4) (5).

The idea is that we need to have a high enough “density” of nodes to get a cluster. For that the notion of a core node is defined using two parameters $K$ and $sim$: a node is a core node if it has more than $K$ neighbors more similar than $sim$.

If a core node is found, one cluster is created and is expanded in two steps:

- All the neighbors more similar than $sim$ of the core node are added to the cluster (Figure 2.17).
- All the neighbors of the core node which are also core nodes are expanded (Figure 2.18).

DBSCAN is finished when all the nodes have been visited.

DBSCAN seems to have a lot of advantages: it can detect clusters of arbitrary shape and it deals with the problem of noise which is essential in our case: on social network there are a lot of posts unrelated to any other posts but which would be classified anyway into one topic by
the classic clustering algorithms. While using DBSCAN if a node is not a core node and not more similar than \( sim \) to one core node, it would be classified as NOISE.

2.2.4.2 Example with \( K=3 \) and \( sim=0.3 \)

![Diagram of a network with nodes and edges connecting them](image)

A node is picked randomly (here the node 5) and we check if it is a core node.

![Diagram of the network after node 5 is found](image)

5 has 3 neighbors closer than 0.3 so it is a **core node**. A new cluster is started.

![Diagram of the network after the core node is found](image)

2 and 4 are not core nodes, their neighbors are not added to the cluster. 6 is one core node hence 7 and 9 are added to the clusters.

There are no other core nodes, the clustering is finished.
### 2.2.4.3 Pseudo-code

**Algorithm 2: DBSCAN**

<table>
<thead>
<tr>
<th>INPUT: $G$, $sim$, $K$</th>
<th>// $G$ is the graph of similarities.</th>
</tr>
</thead>
</table>

while (there is an unvisited node) do
  
  pick randomly one unvisited node $N$
  
  mark $N$ as visited
  
  expand($G, N, sim, K$)

end

expand($G, N, sim, K$)

  $neighbor$=get_neighbor($N, sim$)

  if (size_of($neighbor$)<$K$) then
    
    cluster($N$)=noise

  else

    for ($n$ in $neighbor$)
      
      cluster($n$)=cluster($N$)
      
      expand($G, n, sim, K$)

  end

end
Chapter 3 Our algorithms

Based on these different algorithms, we will build our own algorithms to compute trending topics.

3.1 Frequency-based algorithm

3.1.1 Based on Twitter algorithm

We will use the schema of the algorithm used by Twitter for the two first steps:

- We compute the frequencies of all the words which were posted on Chatter during the last 6 months to be able to compute the td-idf of each word.
- We compute all the frequencies weighted by the td-idf on a small time interval (in our experiment we chose a time interval of one day).

We would like merge two words when they appear almost all the time in the same posts. Like in the example of the Figure 2.3, how could we manage to detect that Prince William and Kate Middleton correspond to the same topic? We will try to solve this problem based on an inquiry about happiness in Sweden.

3.1.2 Inquiry about what make the Swedish people happy

An inquiry about happiness (6) was made in Sweden by Sverker Sikström and the text-analysis company Saplo. The goal of the inquiry was to determine what makes the Swedish people happy. The results are not really interesting for us, the technique they used is however good to analyze: they counted all the words which were written in the same sentence as the word “lycka” (which means happiness in Swedish) in Swedish media in 2010.

Basically their problem is: they have one topic “happiness” and want to know all the words referring to this topic. For us it is exactly the same problem, we have one topic: one of the most frequent words (for example Prince William) and in order to avoid having duplicated topics we want to know what are the other words that belong to this topic.
3.1.3 How will we use it?
Hence we had the idea while counting the frequencies of each word to keep track of each post number where a word is occurring and at the end to compute the correlation between the top 50 words. By correlation we mean the number of times they appear in the same post. We decide that if two of the most frequent words appear more than half of the time in the same post, we will merge them into one topic. By doing this, Prince William and Kate Middleton would be classified into the same topic.

It is pretty simple to do that and the question that one can ask is why Twitter does not follow this rule. The reason is most probably because of the scalability, there are more than 50 millions posts a day on Twitter. Fortunately for us, Chatter is just used inside a company so Chatter does not need to scale as well as Twitter and can therefore use a more complex algorithm.

3.1.4 Last step in our algorithm
Hence after the two first steps of our algorithms, we add a last one: for each of the top words, if they appear more than half the time in the same post as another top word, they are both merged into one topic, thus we manage to avoid a lot of duplicated trending topics.

We will now get into details of how we used the clustering algorithms to detect trending topics.

3.2 Using clustering algorithms

3.2.1 Jaccard Similarity
To be able to use clustering algorithms, we need a way to compute the distance between two posts. We will use the Jaccard similarity coefficient: if we have two posts $P_1$ and $P_2$, the Jaccard similarity coefficient is equal to:

$$ J(P_1, P_2) = \frac{|P_1 \cap P_2|}{|P_1 \cup P_2|} $$

Equation 3.1: Jaccard similarity coefficient.
where $P_1 \cap P_2$ is the number of words in common between the 2 posts and $P_1 \cup P_2$ is the total number of different words in the 2 posts.

### 3.2.2 Comparison using a global ordering

To avoid having to compare each word of the first post with each word of the second post which would be a complexity of $O(n^2)$ where $n$ is the average number of words in one post, we sort all the posts according to lexical order. We just need to do that once for each post and then the complexity to compute the similarity between two posts becomes $O(n)$.

### 3.2.3 Clustering algorithms

Now that we have a distance between posts, we can build the graph of similarities. We can apply some clustering algorithms; for that we need to choose the most suitable to our problem. The clustering algorithm should be able:

- To detect clusters of all shapes and where it is not needed to give information about the clusters a priori.
- Not to classify posts which do not belong to any clusters, simply classify them as noise.

MCL and DBSCAN both fulfilled these requirements; hence we decided to try one algorithm using DBSCAN and one other using MCL.
Chapter 4 Experiment

In this section we explain on which data we tested our algorithm and how we used the outputs of our algorithms to determine which one was the best.

4.1 Data collection

Chatter was released in 2010. For our experiment we got all the posts which were posted between Mach 2010 and October 2010 on the Chatter inside the company salesforces.com itself, which represents an amount of 30 000 posts.

4.2 Preprocessing of the data

4.2.1 Time interval

We have to decide a time interval on which we will compute the trending topics. As the amount of posts on Chatter is not really important (about 500 posts a day), a time interval of one day was decided. Thirty days were chosen regularly between March and October 2010. For each of these days, the different algorithms were applied on all the posts posted during this day.

4.2.2 Stop words removal

In the algorithm based on frequency analysis we have to remove the stop words otherwise the only trending topics will be “a”, “is”, “at”. In the second type of algorithm it is less essential to remove them, but it is still better otherwise two posts can be similar only because they share stop words.

Hence we use a list of English stop words and we remove all the stop words from the different posts.

4.2.3 Parameters choice

In both MCL and DBSCAN there are many parameters to determine. Different values of the parameters were tried to find the values the most suitable to our data.
4.2.3.1 MCL

In MCL the parameters are the threshold parameters \( \text{min} \) and the inflation parameter \( r \). For \( \text{min} \), values between \( 10^{-1} \) and \( 10^{-5} \) were tried. This does not influence significantly the clusters but the number of iterations. We chose \( 10^{-3} \) for our experiments. As Stijn van Dongen recommends, values between 1.4 and 6 were tried for \( r \). When \( r \) was equal to 6, all the clusters were composed of a single post. The value between 1.4 and 2.5 gave acceptable results. However when 1.4 was used, sometimes clusters corresponding to different topics were obtained. Hence for our experiments \( r \) is chosen equal to 2.

4.2.3.2 DBSCAN

In DBSCAN the parameters are \( \text{sim} \) and \( K \). Values between 0.1 and 0.4 were tried for \( \text{sim} \) and between 2 and 5 for \( K \). When \( \text{sim} \) is equal to 0.1 and \( K \) to 2, all the graph is classified into a single post. When \( \text{sim} \) is equal to 0.4 and \( K \) to 5, only really few core nodes are found and hence only almost all the posts are classified as noise.

With \( \text{sim} \) equal to 0.3 and \( K \) equal to 3, we obtained good results on our data; hence these values were chosen for the experiments.

4.3 Evaluation of the results of the algorithms

The problem that we have now is that it is difficult to evaluate ourselves which algorithm is the best; we have to find a way to be able to evaluate our different algorithms. To do so, human interaction is needed. We use a database of Chatter posts which have been classified using our different algorithms: the word-frequency approach, DBSCAN, Markov-Clustering.

Then we ask some volunteers to go on a simple website [http://perso.telecom-paristech.fr/~chaubet/](http://perso.telecom-paristech.fr/~chaubet/) : where 2 posts which have been classified in the same topic will be displayed, and the user has to decide whether the 2 posts belong to the same topic. We also, to check the seriousness of the evaluators, suggest random pairs not classified by any algorithm. This website can be seen on the figure 4.1.
As the posts we used were extracted from the Chatter inside Salesforce, a lot of posts contain specific vocabularies such as “Gus”, “Sprint”, “Dreamforce” or “Cloudburst” it can be hard for people outside of Salesforce to understand some posts. For this reason and also because posts inside Chatter are supposed to remain secret, only people inside Salesforce were asked to answer.

With this website we get an estimate for each algorithm the percentage of clustering which was indeed correct and we are able to compare them. The statistics of answers can be seen directly on the website (figure 4.2).
<table>
<thead>
<tr>
<th>Method</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbscan</td>
<td>84</td>
<td>45</td>
</tr>
<tr>
<td>mcl</td>
<td>78</td>
<td>43</td>
</tr>
<tr>
<td>frequency analysis</td>
<td>48</td>
<td>58</td>
</tr>
<tr>
<td>random</td>
<td>3</td>
<td>107</td>
</tr>
</tbody>
</table>

I want to answer again!

Figure 4.2: Statistics of answers
Chapter 5 Results and analysis

5.1 Results

![Percentage of correct classification](image)

*Figure 5.1: Percentage of correct classification of the different algorithms*

The random algorithm was just used to order to check the honesty of the evaluator and to avoid that he/she could answer yes all the time. We were glad to see that the random algorithm was said to give correct classifications only 3 percent of the time (Figure 5.1), it proves that the users answered carefully.

Frequency analysis algorithm with 30% of correct classification has better results than the random algorithm but the algorithms using DBSCAN or MCL seem to be much better. We will do now a deeper analysis of these results.
5.2 Frequency analysis algorithm

As we said, the frequency analysis algorithm is better than the random algorithm but has pretty bad results compared to the algorithms using DBSCAN or MCL.

We could simply conclude that the second type of algorithms is much better but the situation is more complex than that.

If we just look at the trending topics and not at the posts included in it, the results are not so far from the DBSCAN and MCL algorithms. The main problem in this algorithm is actually to determine which posts belong to one topic. We can find that one word or a set of words are trending but the only way to know which posts belong to that topic is to display all the posts with that word in it, which is really approximated and which explains the bad result for this algorithm.

Twitter is facing exactly this problem. When you click on one trending topic, all the posts containing this word are displayed. It is ok for the words which can refer only to one topic, for example right now Gaddafi is trending and all the posts containing Gaddafi will refer to the same topic: what is currently happening in Libya. However really often the trending words are more ambiguous and it is often difficult to figure out what the topic exactly is because a lot of posts displayed contained the trending word but do not belong to this topic and are just noise. For example Leigh was trending on Twitter because Leigh Sales was presenting a TV-show in Australia. Leigh was actually a topic and it was not a Twitter mistake. But unfortunately Leigh is also a first name and a city in the US. With its algorithm Twitter cannot distinguish between these different meanings. Hence as you can see on the Figure 5.2 most posts displayed in the topic “Leigh Sales” are indeed just noise.

To sum up we can say that algorithms based on frequency analysis can be ok to detect trending topics and it is the only way to do so when there is a big amount of posts. However these algorithms do not cluster, they can just say that a word is trending but are not efficient to detect which posts belong to one topic.
Figure 5.2: Frequency analysis algorithm display all the posts containing one word whether it is the same topic or not.
5.3 DBSCAN vs. MCL

DBSCAN and MCL have about the same percentage of correct classification. But there are a lot of differences between them. We will compare them in a deeper analysis to be able to decide which one we will use.

5.3.1 Complexity

MCL implemented straightforward has a time complexity of $O(n^3)$ (where $n$ is the number of posts) which differs from the time complexity of DBSCAN which is in $O(n \log(n))$.

It can be seen in the Table 5.1 that DBSCAN is indeed much faster and scales much better.

<table>
<thead>
<tr>
<th>Number of posts</th>
<th>Time for DBSCAN (s)</th>
<th>Time for MCL (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>0.0040</td>
<td>2.472</td>
</tr>
<tr>
<td>500</td>
<td>0.0070</td>
<td>13.803</td>
</tr>
<tr>
<td>1000</td>
<td>0.011</td>
<td>349.654</td>
</tr>
<tr>
<td>2000</td>
<td>0.028</td>
<td>3542.276</td>
</tr>
</tbody>
</table>

Table 5.1: Time of execution for DBSCAN and MCL

5.3.2 Difference in classification

Even if MCL and DBSCAN have almost the same results, their classification mistakes are not on the same part of the graph.

5.3.2.1 High-density part of the graph

DBSCAN uses only a threshold for the similarities and the difference between the threshold and the similarity between two nodes does not influence the result of the algorithm. For example if the threshold is 0.3 the clusters will be exactly the same if the similarity between two posts is 0.3 or 1. It can create too big clusters in the high-density part of the graph.

For example in the graph of Figure 5.3, with a threshold of 0.3, DBSCAN will classify all the nodes in the same cluster (Figure 5.5). However MCL (Figure 5.4) splits the left and right part into two different clusters which is better for this data.
Figure 5.3: Example of graph where it is better to use MCL

Figure 5.4: Result of the clustering with MCL

Figure 5.5: Result of the clustering with DBSCAN
On the Figure 5.6, an example on our data can be seen: the two posts in the frame are the posts which make merging the two different clusters when DBSCAN is used.

Figure 5.6: DBSCAN can merge different topics

Hence “Gus is Chatterized” and “is this supposed to be your bug?” are in the same cluster. This problem explains most of the DBSCAN mistakes and there are cases where more than 2 clusters are merged to form one unique cluster. For example in our data we have some posts which are classified into only one cluster with DBSCAN and seven different clusters using MCL.

Luckily this can only happen in the part of the graph with high-density which is not so frequent.

5.3.2.2 Low-density part of the graph

However DBSCAN is better in the low-density part of the graph, because it is designed to handle noise. In DBSCAN a node which does not have enough neighbors is classified as noise. In
MCL to classify the posts, random walk is simulated and all the posts are classified into one cluster. Even if the similarity from one post to all the other posts is really low, it is classified in the less dissimilar cluster.

Of course a way to handle noise is needed in MCL, and to handle that problem in the similarity matrix, all the diagonal elements are set to one (which corresponds to the Jaccard similarity between a post and itself). If a post is not similar to any posts, it will after a few iterations of MCL loop on itself and form a cluster of only one post and eventually the clusters of only one node are classified as noise.

Hence we have a way to handle noise with MCL but it is not perfect. A post that does not belong to any topic can end up in the less dissimilar cluster. An example can be seen on Figure 5.7.

![Figure 5.7: Example of graph where it is better to use DBSCAN. The node on the left should not belong to any topic, but with MCL it will be part of the cluster on the right.](image)
5.3.2.3 Computing similarities and clustering simultaneously

In our implementations of the second type of algorithm the similarities computation and the clustering part were done independently. One can ask two questions: would it be possible to do both at the same time and would it improve the execution time?

The first step of MCL is to normalize the matrix in order to have each column summed up to one. As long as we do not know all the values of the matrix it is not possible to normalize, so we cannot start MCL. Hence when we use MCL we have to perform the two steps independently. However DBSCAN is much more suited to do the two steps at the same time.

5.3.2.4 Idea

In DBSCAN to be part of cluster, a node has to have more than $K$ nodes closer than a given parameter $sim$. While computing the similarities we can count for each node the number of nodes closer than $sim$.

5.3.2.5 New algorithm

We compute the similarities exactly as before, but each time we compute one similarity:

-If it is lower than $sim$ we prune the edge.

-If it is higher than $sim$ we keep it and in the chart $neighbor$ the cell of the 2 nodes are incremented ($neighbor$ counts the number of nodes in the “$sim$-neighborhood” of each node)

When all the similarities between the nodes have been computed, we already have the “$sim$-neighborhood” of each node without any extra cost.

5.3.2.6 Example

Let’s suppose we have 14 posts. In this example $sim$ is 0.3, and $K=3$.

The similarity between $a$ and $b$ is 0.45, hence we keep the edge $a$-$b$ and $neighbor[a]$ and $neighbor[b]$ becomes 1.
The similarity between a and c is 0.17, the edge a-c is pruned.

And so on and so forth for the $n^2$ similarity computations. At the end we have the graph of the Figure 5.8 and for each node its number of neighbors (Table 5.2).

![Graph](image)

**Figure 5.8: Graph obtained after the similarities computation**

<table>
<thead>
<tr>
<th>Node</th>
<th>Number of neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>4</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>2</td>
</tr>
<tr>
<td>j</td>
<td>3</td>
</tr>
<tr>
<td>k</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>2</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 5.2: neighbor counts the number of nodes in the “sim-neighborhood” of each node while computing the similarities*

Hence we know that we have three core nodes c, f and j.

And we have kept in memory the neighbors of each core node:
c: b, d, e, f
f: c, g, h, i
j:k, l, m

As a result when we have finished computing all the similarities, we already know without any extra cost all the core nodes and their neighbors, hence the clustering is almost done: to have the first cluster we just have to expand one core node, which means to add all its neighbor to the cluster and then expand again if one of them was also a core node. We do that as long as there are non-extended core nodes. We will pass through each node at most once hence we have a linear cost for clustering.

5.3.2.7 Updating the trending topics

In this thesis, our goal was to compute trending topics, assuming we have a lot of posts, but once we have done that, if someone posts a new post how do we update the trending topics? Should we start again from the beginning? Or is there a fast way to update the trending topics?

For MCL it is the same problem as in the previous part: when a new post is added we add a new line in the similarity matrix, but then we have to make each column summed up to one, so we will have to change each value in the similarity matrix. Hence MCL has to be started again from the beginning each time we add a new post or that we delete a post which has become too old.

However with DBSCAN it is really easy to update the graph when a node appears or is deleted, we just need to update the table neighbor and then we get the list of core nodes and exactly as before we can have the different clusters in a linear time.

5.3.2.8 Hadoop implementation

We did all our implementations on a single machine; it is not what is done in real life, algorithms run simultaneously on different machines to make them faster. The different machines are called slave machines.

Hadoop is a free-license software framework (7) similar to the Google’s MapReduce. It allows distributing the work between several machines.
MCL consists mostly on matrix multiplications which is easy to implement on Hadoop by using block multiplication: the matrix can be partitioned in small blocks and each slave machine receive only four blocks of the matrix (8).

However with DBSCAN it is more difficult, because DBSCAN is a sequential algorithm, and to be able to know if a node is a core node you need to know all its neighbors, so it is harder to assign parallel tasks to the different slave nodes without giving the entire graph to each slave node.

It was the goal of Nan Gong’s thesis carried out also at Salesforce to determine between MCL and DBSCAN which one was the more scalable using Hadoop.

You can see his results on Figure 5.9 and on Figure 5.10. More detailed results can be seen in Nan Gong thesis (8). Even if MCL execution time improves a bit better using Hadoop, DBSCAN is still much faster.

![Figure 5.9: Execution time of MCL with different number of machines](image-url)
Figure 5.10: Execution time of DBSCAN with different number of machines
Chapter 6 Conclusion and future work

6.1 DBSCAN chosen for the new version of Chatter

In this report, we evaluated different ways of creating trending topics such as for example the traditional frequency based approach used by Twitter. We also explored more semantics based approaches that we thought would be more appropriate for an enterprise service such as Chatter. We also implemented a service for collecting human inputs to evaluate the quality of the different algorithms. It clearly shows that our semantic approach produced significantly better results than frequency based approaches.

Through the results of this thesis, Salesforce.com decided to use the semantic approach to compute trending topics on the next release of Chatter. The future algorithm will be basically the same as the one we presented in this thesis. Based on the differences between MCL and DBSCAN that we explained in the last part, it was decided to use DBSCAN as the clustering algorithm.

Chatter will use the algorithm describe in this thesis to detect trending topics but a few improvement will be made before.

6.2 Improvement

6.2.1 Similarities computation

Our way to measure similarity is quite simple and can be improved. Before computing the Jaccard similarity between the posts, the keywords of each post will be extracted and the similarity between two posts will be the Jaccard similarity coefficient between their two sets of keywords. This should improve considerably the quality of the similarity computation and hence the quality of the trending topics.
6.2.2 Name of the topic

With the algorithm using DBSCAN, we do not detect any trending topics. A step is missing: we only detect clusters which are trending, but we still need to name the topic contain by this cluster.

The easiest way would be to choose the most frequent word as name for our topic, but Salesforce decided to use another algorithm. For the algorithm used in Chatter, it was decided to use TextRank (9): it is an algorithm which builds a graph with the words contained in the different posts of the cluster and will determine the importance of each word in the graph and thus will extract a keyword or a key phrase which will become the trending topic.
Bibliography


