



LUND UNIVERSITY

Master's Thesis

Extraction of trends in SMS text

Christofer Bach, dt05cb6@student.lth.se
Johan Gunnarsson, dt05jg2@student.lth.se

June 7, 2010

Abstract

This thesis investigates the possibilities to extract trends from SMS text that is sent at this moment. With no publicly available corpus of SMS text available we use messages Twitter to aid the development of algorithms and evaluation of the result. Due to the large volume of messages sent using SMS, good scaling characteristics are important. We present a scalable implementation of the algorithm we found most suitable. This implementation is based on MapReduce paradigm using the Hadoop open source framework.

Acknowledgement

Firstly we want to thank Sony Ericsson and Håkan Jonsson for letting us do this Master thesis and supplying us with a pleasant work environment. We would also like to thank our supervisor, Pierre Nugues, for the help and guidance during this project. Furthermore would we like to thank Tobias Ek and Camilla Kirkegaard for their help and valuable comments during this Master thesis.

Contents

1	Introduction	5
2	Background	6
2.1	Trends	6
2.2	SMS	6
2.3	Twitter	7
2.4	SMS and Twitter language	8
2.5	MapReduce and Hadoop	8
3	Purpose and method	10
3.1	Purpose	10
3.2	Evaluation	10
3.3	Test Setup	11
4	Models and term consideration	12
4.1	Classes	12
4.2	TF-IDF	14
4.3	Delta	14
4.3.1	Weighting	15
4.4	Regression analysis	16
4.4.1	Our model	16
4.4.2	Weighting	18
4.5	Cutoffs, filters, and bigrams	18
4.5.1	Filters	19
4.5.2	Bigram	20
5	Privacy issues	21
6	Scalability	22
6.1	Benchmarks	22
6.2	Bill of material	23
7	Result	24
7.1	Measurement and evaluation	24
7.2	Benchmark results	25
7.3	SMS and Twitter corpus	27
8	Discussion	29
9	Conclusions	30
10	Future work	31
A	Prototype	32
A.1	Introduction	32
A.2	Phone	32
A.3	Use-cases	33
A.3.1	Application	33
A.3.2	Widget	34

A.3.3 Service	34
A.4 Android SDK	35
A.5 Front end	35

1 Introduction

With today's information flow on the Internet and through social media, it is hard to stay updated with what is happening. Going through all the information is practically impossible. Therefore are we looking into techniques to handle large amount of information by extracting trends.

There are many techniques and services that can be used to send short messages to each other. Two of the more popular are Short Messaging Service (SMS) and Twitter. These services are an interesting subject for keyword and trend extraction due to there fast reply time. It is possible to get fresh information within minutes or even seconds. This can almost be considered real-time compared to slower media such as newspapers, which can take up to several days to react to new information.

This Master's thesis aims at finding techniques to extract trending information from SMS, but with no available corpus to use we needed to build one ourself. This is a daunting task and the result will not be sufficient for development or scalability testing. Therefore have we decided to build a corpus from publicly available Twitter messages, for use during development and evaluation, which have about the same maximum length. Although the usage of Twitter is slightly different, it is a good approximation of SMS messages.

This Master's thesis is done for Corporate Technology Office at Sony Ericsson. The objectives to solve was firstly to build a SMS corpus and secondly find a technique to see trends in SMS collection, to see what people were talking about.

The goals with the project are to:

- Develop an algorithm for finding trends;
- Develop a scalable implementation of this algorithm;
- Develop an Android application to collect SMS corpus and present the calculated trends to the user.

2 Background

This part of the report describes background information about the techniques and concepts related to this project and report.

2.1 Trends

A trend can be defined as “The popular taste at a given time” (WordNet, 2010). Depending on the context, this time span can vary a lot, from seconds to centuries. In our case, we would like the time span to be as close to now as possible, because we want to be near as real-time as possible and be able to react fast to ongoing events. This is not possible for us to do as we can not access all messages that are sent this very moment and it takes time to do calculations. This forces us to use a time span of few minutes to few hours. We have chosen to have the time span of one hour as this gives us a buffer time to calculate and a hour is a natural time span in today’s time measuring system. A history is needed that significantly larger than our time span to estimate the increase or decrease of specific term over time, and thus determine if it’s a trend or not.

To find a significant difference is not challenging, but to rank differences is not an obvious task and can sometimes depend on its context and use. A trend in our context is a term that has grown considerably the last hours. To rank them, we should include actual growth as a weight to the rank so a more frequent term is not punished by its smallness in percentage growth compared to the actual growth in size.

2.2 SMS

SMS is a communication protocol within the GSM system for exchanging short messages. Each message is limited to 1120 bits, which is 140 8-bit or 160 7-bit characters. Main uses include notification of voice mail, payments and chat (Peersman et al., 2000). SMS consists of types two services:

- Point-to-point for one-to-one communication between for example two phones or one phone and a software component;
- Cell broadcast for one-to-many used for mass messaging of same message to many receivers. Cell broadcast systems are used today in Japan and the Netherlands for public warning announcements (Cell Broadcast Forum, 2010).

The number of text messages sent annually has grown exponentially over the past years and is expected to continue to do so. Figure 1 shows quarterly message volume in Sweden with daily average message volume approaching 40 million messages in Q4 2009 (World Cellular Information Service, 2010).

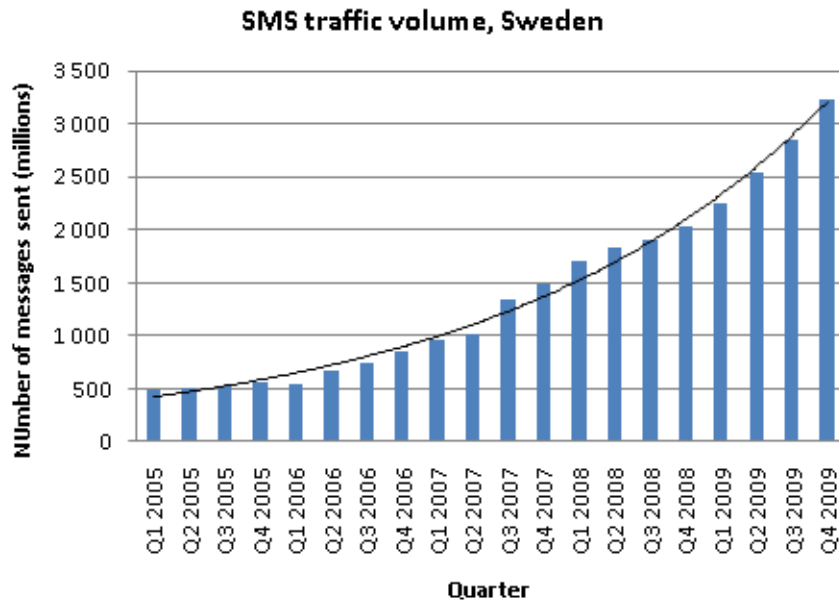


Figure 1: SMS traffic per quarter in Sweden over the past five years (World Cellular Information Service, 2010).

2.3 Twitter

Twitter is a webservice used for social networking and micro blogging. Twitter users can publish short messages to their profile page. Other users can read and subscribe to these messages. The messages are called a “tweet” and consists of at most 140 characters. Tweets can be published from either the Twitter website, SMS, or a third party application. Due to the 140 characters limitation, and the ability of sending tweets via SMS from a mobile device, Twitter has been described as the “SMS of the Internet” (D’Monte, 2009). Therefore Twitter is a good substitute and compliment to a SMS collection as these are hard to come by and are usually very small (Twitter, 2010).

Twitter has several Application programming interfaces (API), two REST APIs and a Streaming API. One of the REST APIs allows developers to access core Twitter data, update time lines, status data, and user information. The other REST API is for interaction with Twitter search and trends data. The Streaming API is a way to access a large amount of tweets in filtered or sampled form in real time (Twitter API, 2010).

In this project, we will use the Streaming API to gather a sample of Twitter’s messages for testing and validation for the trend extraction algorithm. We will also save Twitter trends to be able to compare the result.

2.4 SMS and Twitter language

As mentioned by Baron (2008), SMS is a one-to-one way of communication where messages are sent in an asynchronous way. The asynchronous style of SMS is reflected in text as some people are not always expecting a quick response or that people use SMS as a medium to deliver news or information about updates and changes to the surroundings. This is not the largest part of the genre in SMS (Ling et al., 2005) as the norm in SMS communication is to reply on received messages (Laursen, 2005).

Looking more into the genre of SMS, a study (Ling et al., 2005) shows that 23% of the messages are middle future coordination (things that will happen in the next hours or next day) and 8% are near future coordination (things that have already begun or would happen in the next minutes). But we also have 11% that are questions. This combined with the norm of replying makes SMS a good candidate to get useful information about what is happening and what that will be happening soon.

There are some problems to be dealt with SMS text. Hård af Segerstad (2005) lists some of the problems her rapport. The most important to us are:

- Shortenings, contractions;
- Acronyms and initialisms;
- Misspellings and typos;
- Unconventional spellings;
- Capitals or small letters only (whole messages);
- Substitutions of long words in native language with foreign shorter words.

This makes SMS text harder to compare to other texts, as SMS can have many different terms for the same place, event, and person. This can be solved by grouping synonyms together, but this is out of the scope for this report. We want to see what kind of terms are commonly repeated in a SMS text and if they can be useful in trend extraction.

Twitter is considered a microblog. The language used in Twitter will be close to the language in a real blog. Like SMS, Twitter is considered to be asynchronous communication (Baron, 2008), the user posts the tweet on her Twitter page like a blogger posts a new contribution. Since Twitter is one-to-many communication, many of the things we see in tweets will not appear as they do in a SMS. The user tries to use conventional spelling so everybody can understand, while SMS is adapted to the specific receiver.

2.5 MapReduce and Hadoop

MapReduce is a programming model for processing large data sets, introduced by Google. Programs written using this programming model can be automatically parallelized across multiple cores and machines in large clusters. An underlying application framework takes care of the program's execution and ab-

stracts away cluster management tasks from the user, such as data distribution and handling node failures Dean and Ghemawat (2004).

A MapReduce program takes a set of key-value pairs as input and produces a set of key-value pairs as output, see Figure 2. The program performs the computation using two functions, the *map* function and the *reduce* function, specified by the user. The map function takes a single key-value pair from the input and outputs one or more new key-value pairs. These new pairs, called intermediate pairs, are sorted and grouped by the key. The reduce function is then called for each intermediate key with a list of all corresponding values, and will output one or more key-value pairs as the programs output. To form more complex programs, it is possible to chain several MapReduce jobs, such that the output of one reduce phase is passed as input to the map phase of another job.

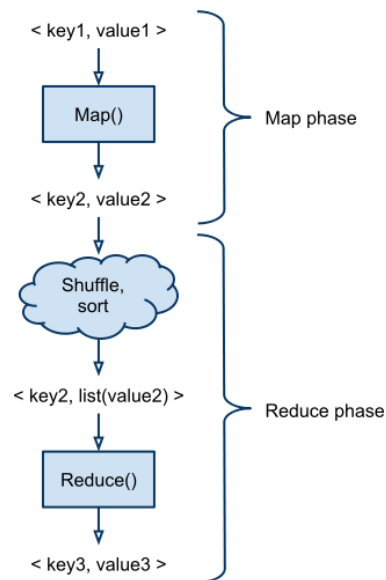


Figure 2: A MapReduce job where $\langle k1, v1 \rangle$ is input and $\langle k3, v3 \rangle$ is output.

Hadoop (2010) is an open source implementation of MapReduce along with a distributed and fault-tolerant file system called HDFS. Hadoop forms a complete client-server infrastructure for storing and processing large amounts of data, designed to scale to petabytes of data and thousands of machines per cluster.

Even though MapReduce is designed to be used in clusters of several machines, it is still effective for use in non-clustered environments, so called one-node clusters. The concurrent nature of the map and reduce phases can be exploited very well on machines with 4 or 8 CPU cores.

3 Purpose and method

3.1 Purpose

The goal with this Master's thesis is to collect a SMS corpus for future research purposes at Sony Ericsson and to develop and evaluate an algorithm for extracting trends in SMS text. The algorithm and implementation of it should also be scalable. Meaning that a computation run should finish in a reasonable amount of time, for data sets that are close to what can be expected in a real-world situation, in terms of amount of messages sent in Sweden.

In this section we will list the research questions, we want to answer in this Master's thesis. We will try to answer them in the conclusion section.

- Which problems can occur with SMS text compared to normal text?
- What technique can be used to extract trends from SMS text?
- Do we need to develop the technique? or is there any existing on SMS?
- Can the technique we find be scaled and used in real life?

3.2 Evaluation

To evaluate our results, we distribute the trends to the users of our application. That way we can run a continuous informal user study by periodically asking the users for feedback about the trends. However, due to the small number of participating users, the data we collected is not always sufficient to produce meaningful trends.

To collect a larger set of messages to work with, we use Twitter's streaming API. Conveniently, Twitter also provides an API for the trends they calculate for the data. We use a large set of messages from Twitter to test and develop algorithms for our application and the trends data provided by Twitter to evaluate our results. Twitter's trends consists of three types, current, daily, and weekly, each with 20 trending terms for any given moment. The exact workings behind Twitter's method of calculating trends is undocumented and therefore unknown to us, although we can speculate what the trend types means by looking at the names and studying the refresh rates. Because of the unknown properties of the trends provided by Twitter, the result of the evaluation against it has to be taken with a grain of salt. Still, we find it relevant enough to include.

For the evaluation against Twitter's trends we tried to find and tune an algorithm that generates trends as close to the trends provided by Twitter. To get results as unbiased as possible, we tested against data collected from several points in time and used the average over each of the points as the final grade. To score the results of each individual run, we used the concepts of precision and recall that are commonly used concepts in information retrieval and statistics. In our case all the retrieved documents are relevant documents so both concepts will show the same result.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|} \quad (1)$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|} \quad (2)$$

3.3 Test Setup

The functions that we use in our algorithm test and their parameters will be described in detail here. The test data we used consists of 15 sets of a total of 10 hours of data in each set. The sets are randomly chosen from our collected Twitter stream data from March 9-12 and March 15-25 of 2010. The sets are 10 hours of continuous data.

We normalize the text with a lower case filter, the privacy filters and a filter that makes token unigrams and bigrams (these filters will be described later in the report). We only count messages a token is in and not how many occurrences of the token in total. This is also known as the document frequency (DF).

We then remove all terms that do not fit our cutoff limit. The cutoff limits are as follows:

- $\sigma \leq y_n$
- *Total term count* ≥ 5
- $y_n \geq 1, y_{n-1} \geq 1, \dots, y_2 \geq 1$
- $y_1 \geq 2$

Where y_1 is the first data point's DF value and y_n is the last point's DF value. Tokens that do not meet all of the conditions are removed. *Total term count* is the sum of all data points. σ is standard deviation of the term for the point y_1 to y_{n-1} .

4 Models and term consideration

This part of the report describes the models to evaluate the terms, how terms are cut off, filtered, and how we deal with sensitive terms. We will be discussing the following models:

- *TF-IDF*
- *Delta*
- *Regression analysis*

4.1 Classes

To be able to compare changes in frequency over time, we partitioned the collection in several groups. These groups are called classes and each is a collection of messages divided by either a time period or fixed number of messages. This is needed so we can compare changes between the classes. A single group will not give us any information about changes over time. This will require more data storage as we need several different frequency counts each term, instead of a single value.

An example is given in Table 1–4. Table 1 shows six messages that will be divided into classes based on whole hours. This will give us two classes. Class 1 will have four messages sent from 19:00 to 19:59 and class 2 will have the remaining two as these are sent between 20:00 and 20:59. Tables 2 and 3 show the extracted terms broken down by class. The end resultat can be put together to form one big table. See Table 4.

<i>Message</i>	<i>Time</i>
Hello everyone	19:00
How is everyone doing?	19:10
Bye everyone!	19:35
What color is my car?	19:59
The car is red!	20:00
Has anyone seen my car? anyone?	20:11

Table 1: SMS texts that are going to be divided in to classes.

<i>Term</i>	<i>DF</i>
hello	1
everyone	3
how	1
is	2
doing	1
bye	1
what	1
color	1
my	1
car	1

Table 2: Class 1. Terms extracted from 19:00 to 19:59.

<i>Term</i>	<i>DF</i>
the	1
car	2
is	1
red	1
has	1
anyone	1
seen	1
my	1

Table 3: Class 2. Terms extracted from 20:00 to 20:59.

<i>Term</i>	<i>DF₁</i>	<i>DF₂</i>
hello	1	0
everyone	3	0
how	1	0
is	2	1
doing	1	0
bye	1	0
what	1	0
color	1	0
my	1	1
car	1	2
the	0	1
red	0	1
has	0	1
anyone	0	1
seen	0	1

Table 4: Showing complete table of data.

4.2 TF-IDF

TF-IDF is a method to weigh terms and phrases in a document. TF-IDF consist of two parts. TF that stands for “Term Frequency” and IDF “Inverted Document Frequency”.

$$(tf-idf)_{i,j} = tf_{i,j} \times idf_i \quad (3)$$

TF is used to determine how common a term is in a document. This is done by counting the number of occurrences of the term in the document and dividing this with the total number of terms in the document:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}. \quad (4)$$

$n_{i,j}$ is the total number of occurrences of the term i in the document j . The Sum in the divider is the total number of occurrences all terms in the document. This means that a high TF value represents a term that is common in the document and a low value represent a uncommon term.

IDF is used for evaluating the general importance of a term. This is done by counting the number of documents containing the term, known as “Document Frequency” (DF). To be able to see the general importance, we need to invert DF. This is done by counting the total number of documents in the collection and dividing with DF, then applying the logarithm of that quotient:

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}. \quad (5)$$

Where $|D|$ is the total number of documents. The divider is the total number of documents that contain the term i . This is only possible when the term is in the collection otherwise this will lead to a division-by-zero. Therefore a more common to use:

$$1 + |d : t_i \in d|. \quad (6)$$

A high TF-IDF value is acquired by having a high TF value, meaning that the term is frequent and a low IDF value meaning the term is not common in the collection. This will lead to that common terms will get a low TF-IDF value and be filtered out.

The most common place for using the TF-IDF formula is in search engines (like AltaVista), where it is used to calculate the score of a document for a specific query. This is done by calculating a TF-IDF score for each term in the query and all documents. The document with the highest sum of all TF-IDF scores is the most relevant one.

4.3 Delta

The delta model is based on the DF values as mentioned in the previous part. Due to the character limitation of SMS text, repeated terms do not have higher

significance, as they would in longer documents such as news articles and research papers. The purpose of the delta model is to see how much a given DF differs from its mean value. The model is very intuitive, simple, cheap and low complexity in its basic form, but can easily be made more advanced by adding weights. It is a simple model to detected significant changes. A low complexity model is advantageous as the extraction should work on large data quantities. Several classes will be needed in this model.

$$\mu_t = \frac{\sum_{k=1}^{n-1} DF_{t,k}}{n-1} \quad (7)$$

$$\Delta_t = DF_{t,n} - \mu_t \quad (8)$$

$$f_t = \frac{DF_{t,n}}{\mu_t} \quad (9)$$

Where $DF_{t,c}$ is the document frequency of term t in class c . n is the number of classes. $DF_{t,1}$ is the document frequency of the oldest class of term t . $DF_{t,n}$ is the document frequency of the latest class of term t . μ_t is the mean of all classes except the last. Δ_t is the difference between the last class and the mean. f_t is the fraction of the last class and the mean.

4.3.1 Weighting

Just using either the Δ_t or f_t is the simplest way to compare terms against each other. However this will quickly lead to problems. The problem with Δ_t is that terms that are used frequently will have a high score, as these terms will have a larger fluctuation over time. But looking at f_t for these terms, we will see a nonsignificant fluctuation. The opposite of this problem is found when we just weigh on the change in f_t . Only terms that have a small DF will have a high score, as their change in f_t will be big even with small, insignificant changes in Δ_t .

A solution to this problem is to combine these two weights so the significant changes are in both weights. There are several different ways these two can be combined. We tried all possible combinations and the best are the following:

$$Score_{t,1} = f_t, \quad (10)$$

$$Score_{t,2} = |\Delta_t| \log(f_t), \quad (11)$$

$$Score_{t,3} = \log(\Delta_t) f_t, \quad (12)$$

$$Score_{t,4} = \Delta_t f_t, \quad (13)$$

$$Score_{t,5} = DF_{t,n} f_t, \quad (14)$$

$$Score_{t,6} = DF_{t,n} \log(f_t). \quad (15)$$

Besides multiplying the Δ_t and f_t we also tried to use logarithm to minimize the impact of them. This was derived from the TF-IDF model.

4.4 Regression analysis

Fitting a number of data points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ to a function with one or more unknown variables is called regression analysis. It is widely used when trying to forecast or predict future values of a series of measurements, or finding a relationship between two dependent units. The function to fit against, also called regression function or model, can be any function. Common models are linear (16), polynomial (17), and sinusoidal (18). A common example for regression analysis is fitting temperature measurements over a year. It turns out that in many places it fits surprisingly well with a sinusoidal model (Sauer, 2005).

$$F(x) = \alpha + \beta x \quad (16)$$

$$F(x) = \alpha + \beta_1 x + \beta_2 x^2 + \dots + \beta_n x^n \quad (17)$$

$$F(x) = C + A \sin(2\pi\omega x + \varphi) \quad (18)$$

There are many methods for estimating the unknowns in the model. One of the simpler methods is called least squares. The principle is to choose the unknown values such that they minimize the sum of all squared errors, where the error ε , is the difference between the data point y -value and function value for the corresponding x -value, $\varepsilon_n = y_n - f(x_n)$.

Linear regression with least squares through origin (19)–(22) (Nielsen, 2009).

$$x_k = k, \alpha = 0 \quad (19)$$

$$y_1 = \beta x_1 \quad (20)$$

⋮

$$y_n = \beta x_n \quad (21)$$

$$\beta = \frac{\sum_{k=1}^n (x_k y_k)}{\sum_{k=1}^n (x_k^2)} \quad (22)$$

4.4.1 Our model

We used the regression analysis with a linear model for predicting the use of a term in the next hours. A term with a high predicted increase of usage is considered a trend. Our data points are number the frequency of the term during the 3 to 5 latest classes. We want to have a model that applies regression analysis on growth of the classes with respect to the oldest one. We do this by normalizing all the classes and adjusting the values such that the oldest is in origin. This way the regression line will go through the first point and we can reuse the previously mentioned (22) by adjusting our model to match it.

First we have the data points unchanged, see Figure 3. The first step is to normalize all the classes by dividing them by the oldest one, y_1 . See (23)–(26) and the new points will be shown in Figure 4.

$$y'_1 = \frac{y_1}{y_1} = 1 \quad (23)$$

$$y'_2 = \frac{y_2}{y_1} \quad (24)$$

⋮

$$y'_n = \frac{y_n}{y_1} \quad (25)$$

$$x_k = k \quad (26)$$

The second step is to translate all the normalized points down by one unit and left by one unit. This is done by subtracting all the normalized values with one and then subtracting the k -value by one, so we use the previous x -value. See (27)-(29) and the new points will be shown in Figure 5.

$$y'_1 - 1 = \beta x_{1-1} = \beta x_0 = 0 \quad (27)$$

$$y'_2 - 1 = \beta x_{2-1} = \beta x_1 \quad (28)$$

⋮

$$y'_n - 1 = \beta x_{n-1} \quad (29)$$

The final step is to calculate β on all the normalized points except the oldest, y'_1 . The formula for this is taken from (22) and using the new model (27)-(29) we get (30). The calculated regression line is displayed in Figure 6.

$$\beta = \frac{\sum_{k=2}^n ((\frac{y_k}{y_1} - 1)x_{k-1})}{\sum_{k=2}^n x_{k-1}} \quad (30)$$

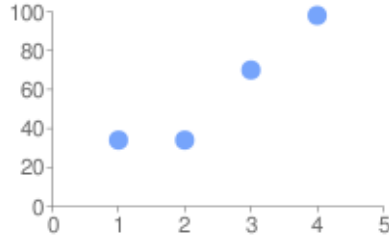


Figure 3: Classes before normalization. 4 data points.

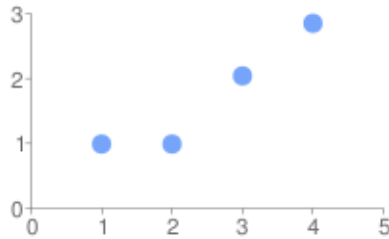


Figure 4: Classes after normalization. 4 data points.

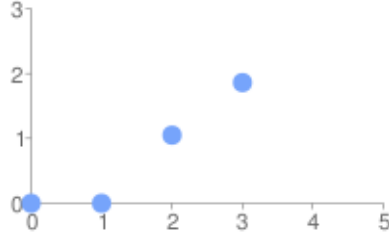


Figure 5: Classes after translation. 4 data points.

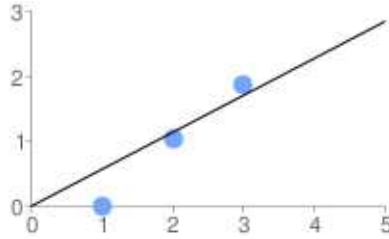


Figure 6: Linear regression on the remaining classes that are not in origin.

4.4.2 Weighting

We need to test different weights along with the regression analysis, for the same reasons as with the delta model. We use the mean value as a baseline and detect significant changes from it. We tried all possible combinations and the best are the following:

$$Score_{t,7} = \beta, \quad (31)$$

$$Score_{t,8} = \beta \cdot (y_n - y_1), \quad (32)$$

$$Score_{t,9} = \beta \cdot (y_n - \mu_t), \quad (33)$$

$$Score_{t,10} = \beta \cdot \left(y_n - \frac{y_{n-1} + \dots + y_1}{n-1} \right), \quad (34)$$

$$Score_{t,11} = \beta \cdot \left(y_n - \frac{y_n + \dots + y_1}{n} \right). \quad (35)$$

We tried to use the difference between the latest value y_n and different mean values to weigh against β . We tried the mean value of all regression points, all but the latest regression point, all data points or just the point we normalized with.

4.5 Cutoffs, filters, and bigrams

We have implemented cutoffs both for privacy reasons (see section 5) computational reasons, and for removing reoccurring terms.

For the computational reason, we try to cutoff terms that occurs less times than a certain value. Even if this value is small, it has a big impact in the number

of terms we need to calculate. This have been used in the delta model and regression analysis. We set the cutoff value to be half of the total number of classes used. So if ten classes were used in total, our program will remove terms with a DF of five or below.

Removing reoccurring terms is done by calculating the standard deviation. We remove every term, whose last DF value is smaller than standard deviation of the previous values. This mathematical concept is derived from signal processing. By doing this we will only calculate terms that are peaking over its normal interval and will remove many grammatical terms, as these are used to a great extent and their occurrence can shift dramatically over time periods.

4.5.1 Filters

Some special filters have been implemented to filter the output from tokenizing part of the process. We use *NoNumberFilter*, *NoCreditCardNumberFilter* and *NoSocialSecurityNumberFilter* to filter terms that may be sensitive personal information. Additionally, we use the following filters to transform tokens.

- *LowerCaseFilter* to set all terms to lower case. This filter would transform a stream of tokens like “Abc”, “DEF”, “ghi” to “abc”, “def”, “ghi”.
- *ShingleFilter* that is based on the concept of shingles that was introduced by Broder (1997). This filter produces n-grams of tokens, and would transform a stream of tokens like “abc”, “def”, “ghi” to “abc”, “abc def”, “def”, “def ghi”, “ghi”. We use it to find unigrams and bigrams.
- *UniqueFilter* to remove duplicate tokens from the input stream. A input stream that looks like “a”, “b”, “c”, “a” will output “a”, “b”, “c”. Thus, removing the duplicate “a”.

We use *LowerCaseFilter* to be able to treat terms that essentially are equal, but only differ on the letter case. People tend to write text differently (Hård af Segerstad, 2005). Some with only lower case letter, some with only upper case letter and some mixed. Most phones include software that automatically converts first letters of sentences to upper case. This is the kind of transformations we undo with this filter to simplify later calculations. However, this approach comes with drawbacks. We obviously lose information about the original case of a term that might be useful in later stages, like the final presentation. There are some tactics to artificially introduce the case information again. One way would be to convert the first letter of each term to uppercase, and keeping the rest lowercase. Another approach is to find the most common spelling of the term and replace it with that. This is however very costly and was left out due to being outside the scope of the report.

In text analysis, it can sometimes be interesting to examine n-grams, especially bi- and trigrams. In this project, we have after discussion with our advisers agreed to be handle bigrams and unigrams only as that will include many names and places that would have otherwise be filtered out. We use *ShingleFilter* to construct shingles (token n-grams) from a stream of unigrams. It outputs combinations of tokens as well as the original unigrams.

4.5.2 Bigram

As mentioned before, we are using bigram as terms. This will give us a lot of unwanted bigrams, as some bigrams are going to contain terms that we will not find significant. The bigram will however have significant changes, so we need a way to either weigh a bigram by how its specific unigram are scored. So if both unigrams have a high score then the bigram should also have it, but if one or both unigrams has a low score the bigrams score should also be lowered:

$$Score = \frac{Score(A)Score(B)}{Score(AB)}. \quad (36)$$

An other approach to this is to remove all bigrams that are not wanted, on the same criteria as before, but we will not change the score. And as we only are interested in the top 10-20 terms this can be easily done in post processing. One way of doing this is to take out the highest scored terms, and fill this list until you have at least 20 unigrams (this is if all bigrams are removed we should still have 20 terms). And then we take out a list with the 20-100 best scored unigrams only, so we have a mixed list and a unigram list. We then go through the mixed list and look at its bigrams. If the bigram's two unigrams are in the unigram list then the bigram is not removed otherwise it is. This way will have a constant calculation time of removing unwanted bigram, despite what ever increase in terms we have so it's the best complexity for scaling.

5 Privacy issues

SMS users, apart from Twitter users, expect the contents of the message be private between the sender and receiver. That opens up problems when extracting and publishing terms and phrases from these messages. It is very important to make sure that the published trends contain no personal information that can harm the involved individuals. We present a couple of solutions to how to protect personal and sometimes secret information.

Our first idea was to implement a way to let the user review messages that are included in the extraction of trends. We do this by buffering all messages in a queue before they are uploaded from the phone. The messages stays in the queue for at least 15 minutes. During that time the user can review and reject messages. If the user doesn't take action, all messages will be uploaded. If users know that they can control what part of their personal information that is published and what is kept private, they are more likely to use a service and publish information (Govani and Pashley, 2005).

Another technique is to filter out terms that match certain patterns that are known to be sensitive. This happens on server-side in a later stage, during the tokenization of the messages. These filters can be built as one or more regular expressions. We have integrated the following list of filters.

- *NoNumberFilter* to exclude numbers greater than a certain amount of digits. Primarily this is to filter out phone numbers.
- *NoCreditCardNumberFilter* to exclude credit card numbers. Credit card numbers follow a very strict pattern.
- *NoSocialSecurityNumberFilter* to exclude social security numbers.

Furthermore we have a cutoff value to remove terms that occur fewer times than a certain value. This to exclude low frequency terms that are sometimes sent over SMS such as one-time passwords or codes. But also so the user do not see the SMS she received or sent, as can be the case with low traffic. The exact cut-off value should adapt to the total number of messages sent. It could typically be a function of the total message volume.

6 Scalability

If we collect all SMS messages sent in Sweden during a week we may end up with millions, and possibly up to 10 million messages. It is therefore very important to consider the scalability of the system. We want our system to not just work for small data sets, but also for huge sets of data that are closer to something in the real world. To realize our requirements on scalability, we have come up with a few points that if followed, our system is more likely to scale well.

1. **Cache intermediate calculations.** Since our system is intended to use data generated during a sliding time window, we will run calculations on the same data multiple times during different runs. If we instead can cache the results from the first calculation and fetch it in the later runs we can eliminate redundant operations.
2. **On-disk storage.** Due to the massive amounts of data, we can not possibly hold it all in memory at the same time. It may very well work for small data sets, but since we aim to scale this system we will have to assume our data is larger than the available memory. We work around this problem by only keeping parts of the total data set in memory and write the result out to disk when we are done working with it.
3. **Avoid file seeking.** Seeking in a file on disk takes time. That's why we try to avoid it as much as possible. We try to save the data on disk in a format that can be easily read sequentially, thus avoid any seeks. Additionally, if we store the data on disk sorted it will optimize common operations such as merging and joining files on the key that we sorted on. Merge and join on sorted data are linear operations which is very efficient when working with large data sets.
4. **Parallel execution.** Without parallel execution a system is limited to one thread. If our system can be parallelized over both local cores and different machines, we can speed up our calculations by simply adding more machines or cores.

It turns out that Hadoop does all these points well.

6.1 Benchmarks

We are interested in benchmarking and ultimately minimizing the wall clock time a single calculation run takes, and the time complexity for the system as a whole. To do this we reuse the data we gathered from Twitter. By measuring the run time of our system when we do repeated runs with varying size of input and number of cluster nodes. We can then interpolate and extrapolate to estimate how the system will perform for other parameters. All tests are performed on clusters of machines with identical hardware specifications, and are equipped with modern software packages.

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (37)$$

$$\mu = \frac{1}{N} \sum_{k=1}^N x_i \quad (38)$$

We will repeat the tests multiple times and present the mean run times μ (37), the standard deviation σ (38). N is the number of test runs performed and x_i is the run time for test number i .

$$S_p = \frac{T_1}{T_p}. \quad (39)$$

We will also calculate the speedup S_p (39) as we make the cluster larger. T_1 is the execution time of single node and T_p is the execution time of p -nodes.

6.2 Bill of material

- AMD Athlon 64 X2 Dual Core Processor 3800+ (2.0 GHz);
- 2 GB RAM;
- 100 Mbps switched networking;
- Single harddrive;
- Ubuntu 9.04;
 - Linux kernel 2.6.28;
- Hadoop 0.20.2;
 - Replication factor of 2 ;
- Sun JRE 1.6.0_16.

7 Result

7.1 Measurement and evaluation

The test data that will be presented will be the result from testing 15 data sets with 10 hour of Twitter sample data. The number of messages in a data set is range from 335,000 to 588,000 number of messages. The total number of trends for the test set is 300.

We evaluated the data based on the daily precision. The current precision will only be used to differentiate when the same daily precision occurs on other scores. The result can be summarized in the following points:

- We found that the TF-IDF did not find any trend that corresponded to the Twitter trend data.
- The Delta model had the best precision of 6.77% with the $Score_4$ (15). See Table 5.
- Regression analysis with 3 data points had the best overall precision of 8% with $Score_8$ (32). See Table 6.

	<i>Daily</i>	<i>Current</i>
$Score_1$	3.00%	1.33%
$Score_2$	3.67%	2.67%
$Score_3$	0.00%	0.00%
$Score_4$	6.67%	5.33%
$Score_5$	3.33%	2.33%
$Score_6$	6.00%	5.33%

Table 5: Precision result from testing delta.

	<i>Daily</i>	<i>Current</i>
$Score_7$	4.33%	1.67%
$Score_8$	8.00%	4.67%
$Score_9$	7.33%	4.33%
$Score_{10}$	7.33%	4.33%
$Score_{11}$	7.33%	4.33%

Table 6: Precision result from testing regression analysis with 3 data points.

	<i>Daily</i>	<i>Current</i>
$Score_7$	3.00%	1.00%
$Score_8$	5.00%	2.67%
$Score_9$	4.33%	2.33%
$Score_{10}$	5.67%	3.00%
$Score_{11}$	5.67%	3.00%

Table 7: Precision result from testing regression analysis with 4 data points.

	<i>Daily</i>	<i>Current</i>
<i>Score</i> ₇	4.67%	2.33%
<i>Score</i> ₈	4.00%	1.33%
<i>Score</i> ₉	4.33%	1.67%
<i>Score</i> ₁₀	3.67%	1.33%
<i>Score</i> ₁₁	3.67%	1.33%

Table 8: Precision result from testing regression analysis with 5 data points.

7.2 Benchmark results

The results from our benchmark runs are presented in Table 9 to 12.

<i>Nodes</i>	1	2	3	4
μ	19m 30s	11m 59s	9m 51s	8m 20s
σ	14s	15s	29s	32s
<i>Speedup</i>	1.00	1.63	1.98	2.34

Table 9: 1 day of input. 1.3 million messages.

<i>Nodes</i>	1	2	3	4
μ	37m 37s	21m 20s	17m 27s	14m 3s
σ	44s	37s	27s	34s
<i>Speedup</i>	1.00	1.76	2.16	2.68

Table 10: 2 days of input. 2.6 million messages.

<i>Nodes</i>	1	2	3	4
μ	1h 16m 1s	42m 7s	30m 27s	26m 52s
σ	7m 12s	2m 45s	41s	1m 3s
<i>Speedup</i>	1.00	1.80	2.50	2.83

Table 11: 4 days of input. 5.1 million messages.

<i>Nodes</i>	1	2	3	4
μ	2h 29m 58s	1h 24m 0s	1h 5m 12s	54m 5s
σ	34s	6m 11s	1m 59s	1m 41s
<i>Speedup</i>	1.00	1.79	2.34	2.77

Table 12: 8 days of input. 10.2 million messages.

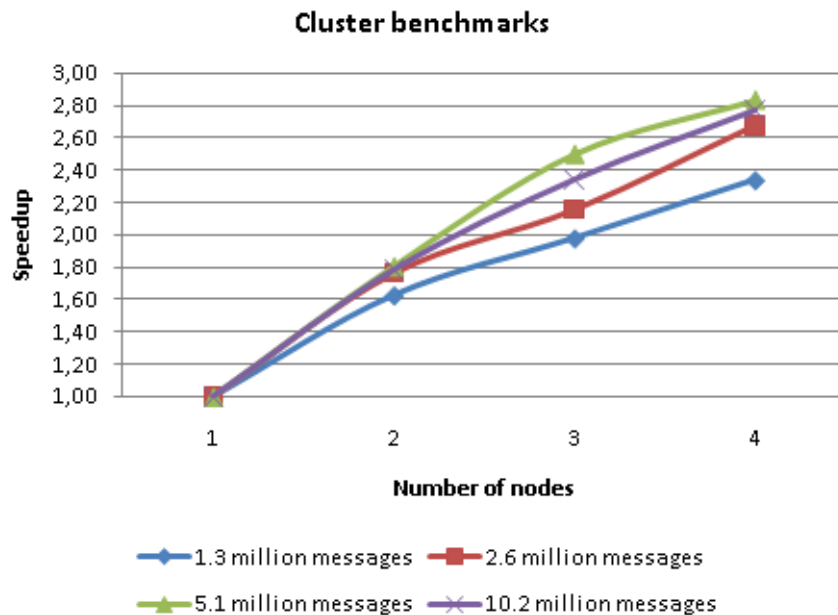


Figure 7: Speedup as a function of cluster size, for 1, 2, 4 and 8 days of SMS traffic in Sweden.

<i>Feature</i>	<i>SMS corpus</i>	<i>Twitter corpus</i>
transmissions (in terms)	11.0 terms	12.1 terms
transmissions (in chars)	47 chars	58 chars
one-word transmissions	9.9% of message	2.0% of messages

Table 13: Data from our SMS and Twitter corpus.

<i>Term</i>	<i>Term Frequency</i>
mms	2.34
gratulerar	2.0
rachael	1.78
gratis	1.75
hurra	1.67
tele2	1.625
grey	1.6
pkt	1.6
to	1.58
parkering	1.57
kr	1.55
heja	1.5
sas	1.5
tråd	1.5
tripit	1.5
gul	1.5
ci	1.5
garantin	1.5
servicebesök	1.5
natti	1.5

Table 14: The 20 most repeated terms per messages they are in, from our SMS corpus (unigrams only).

7.3 SMS and Twitter corpus

We have during this Master’s thesis built up a SMS corpus for use at Sony Ericsson. The corpus has been gathered by some Master’s thesis workers and some regular workers have used the prototype during three months time. All user’s had free SMS in their phone plan, so they just texted as usual.

The SMS corpus consists of 3,152 unique messages on the date of 2010-05-19 and is growing each day. The uniqueness is based on sender, receiver, time and text. The Twitter corpus had the size of 23,030,887 messages.

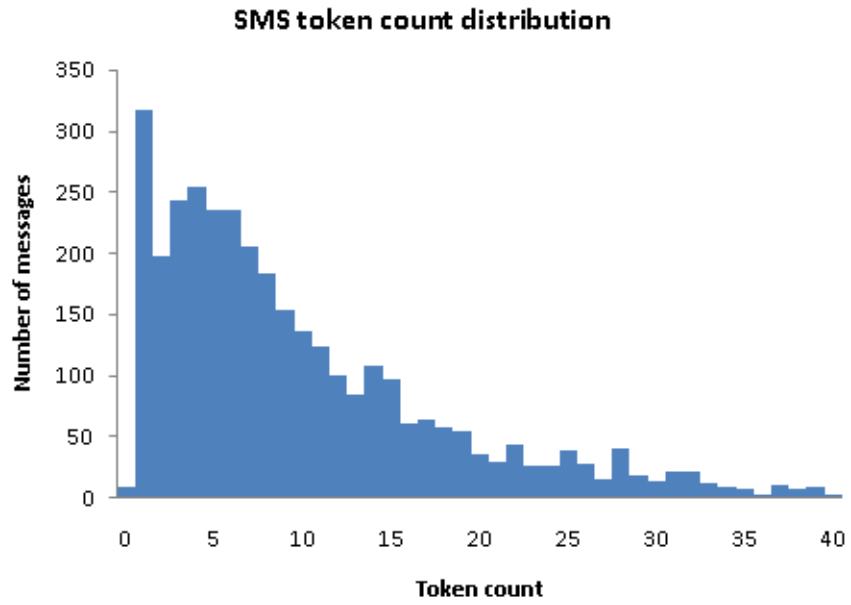


Figure 8: Histogram over SMS corpus.

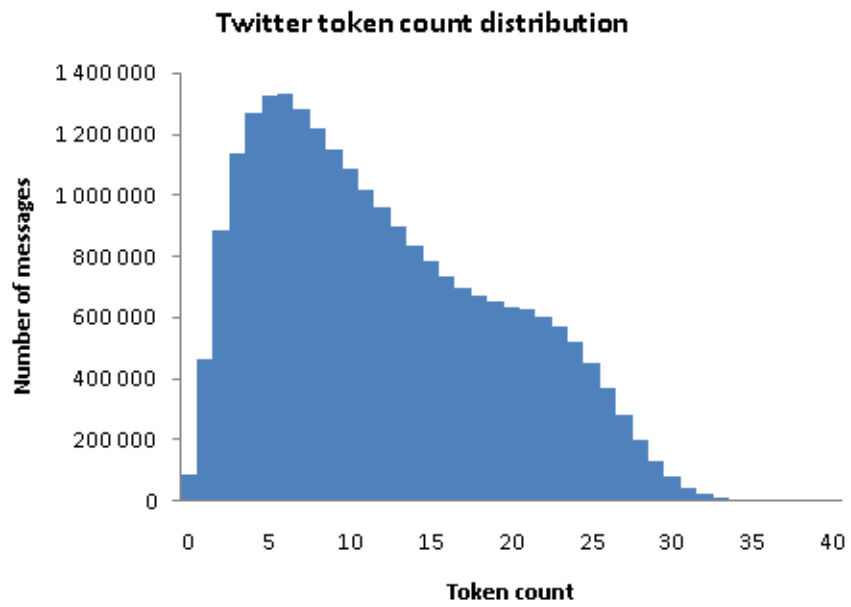


Figure 9: Histogram over Twitter corpus.

8 Discussion

We have found out that there are many similarities between Twitter text and SMS text, but we have also found obvious differences. While the maximum length of messages are about the same, the average character counts and average word count differs on some interesting points. 10% of the SMS messages in our corpus contained only one word, compared to only 2% in the Twitter messages we collected. We believe this is a result of more personal conversation with higher rate of direct and short replies, such as “yes”, “no” and “ok”. It is worth noting that our corpus has very few users and all of them use smart phones with advanced input devices like qwerty keyboards, that might not represent the general public.

The relatively large number of very short messages can pose a problem when trying to further analyze them. For example, language detection can be a problem due to the small amounts of information the messages contain. Also, methods that take the position of the first occurrence of a word becomes less useful.

Regarding the precision from our tests, we got a quite low score. This is because that Twitter do not calculate the trends in the same way we do, and we speculate that searches on Twitter has a big impact in their trend algorithm. We have seen some cases in our data, where a word is decreasing over the last 10 hours. Twitter still ranks that word as a trend. When looking on the Twitter trends we also found many words that have had a clear increase in usage in the last 10 hours, so no real conclusion can be done. The search data is not published.

We have during the test phase tried our models on some special days, to see if the events of these days would show up, like the did on Twitter. We tested on March 10 (Chuck Norris’ birth day) and March 17 (Saint Patrick’s day). On March 10 we found “Chuck”, “Norris”, and “Chuck Norris” in the top 10 trend words with both Delta and Regression analysis model. Similar result was on 17 March, but due to “Saint Patrick’s day” consists of three tokens and we only deal with uni- and bigrams, we encountered some problems. Almost all top 10 words were different terms like “St”, “Patrick’s,” “St Patrick’s”, “Patrick’s day”, “Patty”, and “St Patty”. This shows that the algorithms can find trends like Twitter, but unlike Twitter all different combinations will be shown.

We have found the feedback we got from user’s who tested our application very useful. User’s care very much for their privacy, so convincing them to share their SMS messages can be a difficult task. Therefore we took the feedback very serious and implemented and released features based on it. We know that it is important to have the users’ trust when handling data that can be potentially sensitive. If we move to fast forward user’s can get scared or chocked, stop using the application, and leave the service. In most cases you only get one try, so by damaging the user trust can be fatal. An advice for anyone trying to work with user’s the sensitive information is to plan it with privacy in mind from the very beginning, get the user involved, and listen to their feedback carefully.

9 Conclusions

In this chapter we try to answer the research questions and present other conclusions we have found during this project. The questions were:

- Which problems can occur with SMS text compared to normal text?
- What technique can be used to extract trends from SMS text?
- Do we need to develop the technique? Or is there any existing on SMS?
- Can the technique we find be scaled and used in real life?

On the question “Which problems can occur with SMS text compared to normal text?” we have found some problems in studies (see section 2.4) done on this field, that point out that there are problems in the language used that make SMS hard to work with. Data from our SMS corpus have shown us that 10% of every message is made up from one word. This is observed in both SMS and IM (Baron, 2008). There are two major things that differentiate Twitter text from SMS text. First, only 2% of Twitter are single worded messages. Secondly, the distribution of number of tokens per message is different. See the histograms Figure 8 and Figure 9.

Another conclusion we can see is that the number of terms in our SMS corpus is an approximated exponential distribution (see Figure 8). The majority of the SMS is going to have few terms in them. We also looked at the 20 most commonly repeated terms, to see if they have any significance. Unfortunately we can’t say much as the words didn’t show any useful pattern.

The second question “What technique can be used to extract trends from SMS text?” was a difficult question to answer as all the major services that publish trends keep their algorithms secret. So the answer to this question we need to look on the third question “Do we need to develop the technique? Or is there any existing on SMS?”. As we didn’t find any technique we needed to develop our own. So the techniques we tried out and worked, are the technique that can be used to find trends in SMS.

We were only able to develop two working techniques. We called them the delta model and regression analysis model. The delta model was quite cheap to calculate and to implement. But didn’t show as high precision in the tests as the regression analysis. So the conclusion we have made about the techniques are that regression analysis with the weight $Score_9$ is the best and is what we recommend to Sony Ericsson and was the one we implemented in our prototype.

On the last question “can the technique we find be scaled and used in real life?” we conclude that our implementation handles large data sets very well. Our system handles the one week worst-case of 10 million messages just under a hour. If extra performance is desired it can be archived by adding more machines to the cluster, shown by the extrapolated speedup.

10 Future work

During the work on this Master's thesis we have found some areas that need to be more researched or more information is needed to be gathered.

- A large SMS corpus. As there are none, besides a french with 30 000 messages. But all the SMS language had been normalized.
- A synonym database with different slang/names for corporation, places and events.

For our project we need to find a better data storage for messages, so we do not need to implement our own system. One possible solution might be to integrate the storage with Hadoop using Hbase.

To enhance functionality and improve user experience location filtering can be added to the trends calculation, to display location dependent trends. Some changes to both the prototype need to be implemented for this.

A Prototype

This part describes the prototype system we developed for Sony Ericsson during the Master's thesis, like the system architecture and how the different components communicate. Our implementation is property of Sony Ericsson. Due to disclosure agreement we can not go in to details or provide code.

A.1 Introduction

To collect messages from phones we have developed a prototype phone application that every participating user installs and runs. It runs on phones running the Android operating system developed by Google.

Additionally we have a server that receives and saves messages collected by the phone application. The client communicates with it using a HTTP REST API. The server then inserts the messages into a MySQL database. The server is written as Java Servlet that runs in a Tomcat HTTP Servlet container.

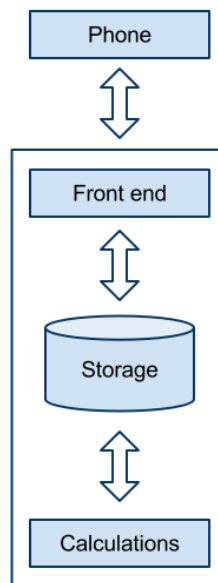


Figure 10: Schematic overview over our prototype system.

A.2 Phone

The application consists of three parts.

- A service running in the background listens for changes in the phone's built in SMS message storage. When something changes, like when a new messages is sent or received, the service will notice and queue it (see section 5) for upload to our server.

- A activity that downloads and displays trends as a list. The trends can be clicked on to bring up a number of options, for example search the trend with Google Search or on Wikipedia.
- A widget that displays trends on the phone home screen like a icon. It is updated automatically every 20 minutes.



Figure 11: Widget on a home screen.

A.3 Use-cases

To get all the three parts to work together we have designed and implemented a list of use cases. Each use case is described below.

A.3.1 Application

Case 1 - Start the application.

- User starts the application from the application menu.
- Application starts and downloads the trends list.
- Trends list is displayed.
- Downloaded trends list is sent to the widget.
- Widget updates itself with the sent trends list.

Case 2 - Failed download or no Internet connection.

- User starts the application from the application menu.

- Application starts and downloads the trends list.
- Application gets notified that trends couldn't be downloaded.
- A message is displayed "Failed to download trends" in the trends list.

A.3.2 Widget

Case 3 - Widget starts up but failed download or no Internet connection.

- Widget is initialized.
- Widget starts and downloads the trends list.
- Widget gets notified that trends couldn't be downloaded.
- Widget displays text "Loading trends..."

Case 4 - Click widget.

- User clicks on the widget.
- Application is started.
- Application gets the trends from the widget.
- Trends list is displayed.

Case 5 - Widget update fails, but trends have been initialized.

- Widget is initialized.
- Widget starts and downloads the trends list.
- Widget gets notified that trends couldn't be downloaded.
- Widget shows the previous displayed trends.

A.3.3 Service

Case 6 - New SMS are sent or received on device.

- The device notifies the service that the SMS database has changed.
- Service checks if any new messages has been added since last upload.
- Service sends the new SMS to privacy queue.
- After 15 minutes the queue tries to upload the SMS.
- If not successful the service does nothing.

Case 7 - Large amounts of SMS are being uploaded.

- The queue tries to upload more than 50 SMS.
- SMS are divided in to groups of 50 SMS each.
- Groups are sent one by one, if one of them fail it is uploaded next time.
- Removes all successful uploaded SMS from the queue.

A.4 Android SDK

Our application has been developed for the Android 1.6 platform. This is also the version Sony Ericsson bases their recently released and upcoming phones on. But also that the majority of the other users have not changed to the latest version of 2.1 (see Figure 12). The program is forward compatible as we follow the best practices of Android development. It should be possible to port it to version 1.5 if needed. This will increase compatibility with older devices, that still have large market share.

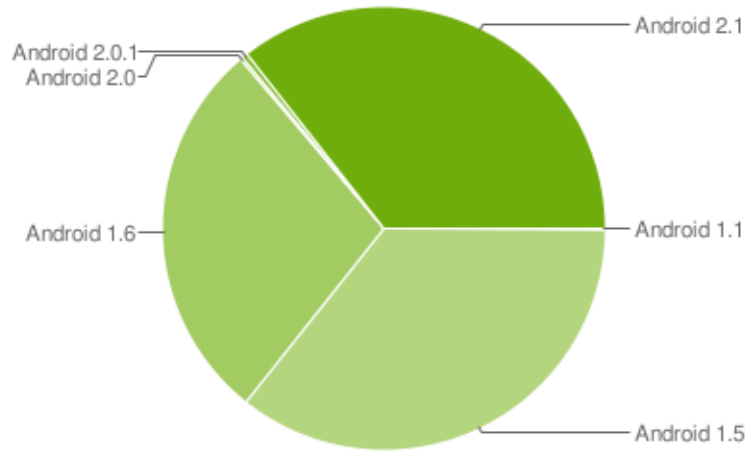


Figure 12: Showing version segmentation (Android, 2010).

<i>Android platform</i>	<i>Percent of devices</i>
Android 1.1	0.1%
Android 1.5	37.2%
Android 1.6	29.4%
Android 2.0	0.3%
Android 2.0.1	0.6%
Android 2.1	32.4%

Table 15: Showing Version segmentation (Android, 2010).

A.5 Front end

The front end has two major functions. The first one is to collect SMS send from the phone so they can be used to calculate trends. The second one is to wait for trends request and send the trends to the phone. These two functions are using the JSON(JavaScript Object Notation) to exchange data between each other. JSON is a lightweight data-interchange format (JSON, 2010).

The front end has several versions of the trend information being sent to the device. This was implemented so old versions of the phone application should

work despite changes in the format. The phone application includes its current API version along with the request to retrieve trends. We currently supports two API versions.

List of Tables

1	SMS texts that are going to be divided in to classes.	12
2	Class 1. Terms extracted from 19:00 to 19:59.	13
3	Class 2. Terms extracted from 20:00 to 20:59.	13
4	Showing complete table of data.	13
5	Precision result from testing delta.	24
6	Precision result from testing regression analysis with 3 data points.	24
7	Precision result from testing regression analysis with 4 data points.	24
8	Precision result from testing regression analysis with 5 data points.	25
9	1 day of input. 1.3 million messages.	25
10	2 days of input. 2.6 million messages.	25
11	4 days of input. 5.1 million messages.	26
12	8 days of input. 10.2 million messages.	26
13	Data from our SMS and Twitter corpus.	27
14	The 20 most repeated terms per messages they are in, from our SMS corpus (unigrams only).	27
15	Showing Version segmentation (Android, 2010).	35

List of Figures

1	SMS traffic per quarter in Sweden over the past five years (World Cellular Information Service, 2010).	7
2	A MapReduce job where $\langle k1, v1 \rangle$ is input and $\langle k3, v3 \rangle$ is output.	9
3	Classes before normalization. 4 data points.	17
4	Classes after normalization. 4 data points.	17
5	Classes after translation. 4 data points.	18
6	Linear regression on the remaining classes that are not in origin.	18
7	Speedup as a function of cluster size, for 1, 2, 4 and 8 days of SMS traffic in Sweden.	26
8	Histogram over SMS corpus.	28
9	Histogram over Twitter corpus.	28
10	Schematic overview over our prototype system.	32
11	Widget on a home screen.	33
12	Showing version segmentation (Android, 2010).	35

References

- Android (2010). Android - Platform Versions. <http://developer.android.com/resources/dashboard/platform-versions.html>.
- Baron, N. S. (2008). *Always On: Language in an Online and Mobile World*. Oxford University Press, USA.
- Broder, A. Z. (1997). On the Resemblance and Containment of Documents. In *In Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29. IEEE Computer Society.
- Cell Broadcast Forum (2010). Cell Broadcast Forum. <http://www.cellbroadcastforum.org/whatisCB/index.html>.
- Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *OSDI '04*, pages 137–150.
- D'Monte, L. (2009). Swine flu's tweet tweet causes online flutter. *Business standard*.
- Govani, T. and Pashley, H. (2005). Student Awareness of the Privacy Implications When Using Facebook. Unpublished paper presented at “The Privacy Poster Fair” at the Carnegie Mellon University School of Library and Information Science. Retrieved June 3, 2010 from <http://lorrie.cranor.org/courses/fa05/tubzhlp.pdf>.
- Hadoop (2010). Apache Hadoop. <http://hadoop.apache.org/>.
- Hård af Segerstad, Y. (2005). Language in SMS - a Socio-Linguistic View. In *The Inside Text. Social, Cultural and Design Perspectives on SMS*, pages 33–51. Kluwer Academic Publishers.
- JSON (2010). Introducing JSON. <http://www.json.org/>.
- Laursen, D. (2005). Please reply! The replying norm in adolescent SMS communication. In *The Inside Text. Social, Cultural and Design Perspectives on SMS*, pages 53–73. Kluwer Academic Publishers.
- Ling, R., Julsrud, T., and Yttri, B. (2005). Nascent Communication Genres within SMS and MMS. In *The Inside Text. Social, Cultural and Design Perspectives on SMS*, pages 75–100. Kluwer Academic Publishers.
- Nielsen, A. A. (2009). Least Squares Adjustment: Linear and Nonlinear Weighted Regression Analysis. Technical University of Denmark, Image Analysis and Computer Graphics, Geoinformatics.
- Peersman, C., Spear, H., Cvetkovic, S., and Smythe, C. (2000). A tutorial overview of the short messageservice within GSM. *Computing and Control Engineering Journal*, pages 79–89.
- Sauer, T. (2005). *Numerical Analysis with CD-ROM*. Addison Wesley.
- Twitter (2010). Twitter - About. <http://twitter.com/about>.
- Twitter API (2010). Twitter - API Overview. <http://apiwiki.twitter.com/API-Overview>.

WordNet (2010). Princeton WordNet Search. <http://wordnetweb.princeton.edu/perl/webwn?s=trend>.

World Cellular Information Service (2010). World Cellular Data Metrics, March 2010.