UPPSALA
UNIVERSITET

UPTEC IT 12 014
Examensarbete 30 hp
Augusti 2012

Tracking the outbreak of diseases Using Twitter

A Machine Learning Approach

Erik Bohlin

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Tracking the outbreak of diseases Using Twitter: A Machine Learning Approach

Erik Bohlin

In this project I have investigated the correlation between talks of illness on Twitter and the amount of calls to the Swedish medical information services (Sjukvårdsupplysningen). The project has only considered tweets located in Sweden and written in Swedish. In order to fulfill the aim of the project I used a SVM-classifier trained on 20,000 tweets manually marked as indicating sickness or not indicative of sickness. The resulting classifier was then used on roughly half a million tweets collected during the spring of 2012. The results were correlated with data from the Swedish medical information services. I was able to show a Pearson correlation of 0.8707051, p = 0.00225 when compared with weekly values from the medical information services. I also use an ets-model fitted to the twitter data to try to predict future values. However I have not been able to evaluate the accuracy of these predictions.

Handledare: Gustaf von Dewall
Ämnesgranskare: Roland Bol
Examinator: Arnold Pears
ISSN: 1401-5749, UPTEC IT 12 014
Sponsor: Softronic

Tryckt av: Reprocentralen ITC

1 Populärvetenskaplig Beskrivning

Under de senaste åren har vi sett en explosionsartad utveckling inom sociala medier, mer och mer av våra liv tillbringas på internet. I det här projektet har jag försökt att utnyttja den informationen som vi frivilligt lägger upp på internet för att se om den kan användas för att beskriva när vi som grupp är troliga att vara sjuka. Jag har i det här projektet använt mig av Twitter[1], som är en mikroblogg. Twitter tillåter sina användare att publicera korta meddelanden, kallade "Tweets" på upp till 140 tecken, som sedan kan ses offentligt.

För att göra detta har jag under våren 2012 samlat in meddelanden från Twitters servrar. För att hålla projektet intressant för Sverige har jag haft en del krav på vilka meddelanden jag har samlat in. De kraven är att meddelandet kommer från inom Sverige, vilket här har definierats som att gps-positionen har funnits innanför en av tre cirklar som placerats ut över Sverige, där cirklarna kan beskrivas som södra, centrala och norra Sverige respektive. Jag har också krävt att språket som meddelandet skrivits påvar Svenska.

Totalt samlades 531,466 unika meddelanden in, spridda över 98 olika dagar. Jag gick sedan igenom de 20,000 första meddelandena och klassificerade dessa manuellt, antingen som relevanta eller ej relevanta. Relevant i det här fallet innebär att meddelandet i fråga indikerade att någon var sjuk. Exempel på meddelanden som skulle klassificerats som positiva, dvs. relevanta är då exempelvis "Jag är sjuk".

När jag sedan hade klassificerat dessa meddelanden manuellt använde jag mig av ett par olika maskininlärningstekniker för att anpassa en funktion, vanligtvis kallad för klassificerare, som givet ett nytt meddelande kan avgöra ifall detta meddelande skall klassificeras som positivt, det vill säga relevant, eller negativt, inte relevant. Då vår funktion kan göra den här klassificeringen automatiskt, utan att vi behöver göra någonting manuellt, så tillåter det oss att hantera väldigt stora mängder data. Om klassificeringen vore gjord manuellt skulle det vara omöjligt att hantera ens en liten del av den data som insamlats under projektets gång. Både insamlingen av Twittermeddelanden och klassificeringen av dessa är en löpande process, och pågår så länge det färdiga systemet körs.

När jag gjort bedömningen att systemet hade samlat in tillräckligt många Twittermeddelanden använde jag dessa för att bygga upp ett antal tidsserier för att se om den data mitt system samlat in var relevant i den riktiga världen. För att kunna avgöra om min data var relevant jämfördes de tidsserierna jag byggt upp med data som insamlats från sjukvårdsupplysningen.

[1] www.twitter.com

Sjukvårdsupplysningen styrs lokalt från landstingen, och det är också de som för statistik över inkomna samtal. Ett problem med detta är att statistiken inte existerar för alla landsting, och några landsting har inte svarat på förfrågan över huvud taget. I den färdiga rapporten använder jag statistik från nio stycken landsting. Den statistiken jag har fått tag på berör tiden mellan den 22 Februari och den 3 Maj 2012.

Statistiken som förs hos sjukvårdsupplysningen har delats upp i mindre delar, beroende på vilken anledning som individen valde att ringa till sjukvårdsupplysningen. För att se till att min data kan användas för jämförelser med den data som jag fått från sjukvårdsupplysningen har jag valt ut ett antal av dessa kontaktorsaker och jämfört statistiken för dessa med min data. För att välja vilka kontaktorsaker jag ville ha med i statistiken gick jag igenom de Twittermeddelanden som min klassificerare hade klassificerat som relevanta, och undersökte vilka klagomål dessa meddelanden visade på. Jag valde sedan de kontaktorsaker som passade in på de klagomål Twittermeddelandena hade.

Jag undersökte sedan vad som kallas för korrelationen mellan dessa tidsserier, det vill säga hur lika tidsserierna är. Det visade sig att om vi grupperar antalet samtal till sjukvårdsupplysningen per dag, och gör samma sak med antalet relevanta Twittermeddelanden så får vi två stycken tidsserier vi kan jämföra. När jag gjorde detta visade det sig att vi fick en negativ korrelation, vilket innebär att många positiva Twittermeddelanden under en dag skulle innebära få samtal till sjukvårdsupplysningen. Jag visar senare i den här projektrapporten att det troligen beror på att antalet Twittermeddelanden som samlats in varierade beroende på vilken tid under projektet det var. Under den tidigare delen av projektet samlades inte lika många Twittermeddelanden in som i slutet, då jag inte hade tillgång till en dedikerad dator för insamlandet.

För att anpassa tidsserien som representerade antalet positiva Twittermeddelanden valde jag att skapa en ny tidsserie, där antalet relevanta Twittermeddelanden under en dag blivit dividerat med det totala antalet Twittermeddelanden under samma dag. På det här sättet korrigerar vi för det faktum att vi har olika mycket meddelanden uppmätta under dagarna. Genom att undersöka korrelationen mellan den här tidsserien och data från sjukvårdsupplysningen fick jag nu en svag men positiv korrelation. De två andra tidsserierna som beskrivs här nedanför använder sig båda av dessa värden för de respektive korrigeringarna.

Som kommer att visas senare i rapporten är antalet positiva Twittermeddelanden olika beroende på vilken veckodag som det är. Den här skillnaden är också väldigt stor, framför allt beror den mycket på om det är en veckodag eller helgdag. För att korrigera för detta använde jag en glidande summa, där varje dags värde bestäms som summan av den dagens värde samt de sex tidigare dagarna. På så sätt blir varje veckodag representerat i varje datapunkt. Jag gjorde dessutom detta på den data jag fått från landstingen. När jag sedan undersökte korrelationen mellan dessa tidsserier fick jag en korrelation på

ca. 0.78, vilket är en väldigt hög korrelation. Korrelation går mellan -1 och 1. Där -1 innebär ett perfekt inverst förhållande och 1 innebär att tidsserierna är exakt lika.

Det starkaste förhållandet fick jag dock när jag summerade veckor för sig. Så att under tiden 22 Februari till 3 Maj fick nio hela veckor, där en vecka måste gå från Måndag till Söndag. När värdena för sjukvårdsupplysningen och Twittermeddelandena summerades på det här sättet så fick vi en korrelation på 0.87, vilket är en väldigt stark korrelation.

De uppmätta tidsserierna användes sedan för att anpassa en statistisk modell till de uppmätta värdena på Twitter. Denna hoppas jag ska kunna användas för att förutsäga hur många positiva Twittermeddelanden som kommer att uppmätas i framtiden. Jag dock inte kunnat evaluera hur bra detta fungerar.

Contents

List of Figures

List of Tables

2 Introduction

2.1 Reading Instructions

The introduction section introduces the problem and discuss what has previously been done in the field. Section 3 introduces the reader to the field of machine learning and the techniques that have been used throughout this project.

Section 4 and section 5 describe how the project has been carried out as well as the complete system. If you are interested in how the problem has been approached read these sections. The non-technical reader and those of you only interested in the results might want to skip these sections. Section 6 briefly introduces time series models as well as describes the Pearson correlation coefficient.

Section 7 and 9 are the most interesting parts of this report since they report my results and conclusions. I would recommend everyone who reads this report to read these sections.

2.2 Problem Formulation and Structure

My goal in this project is to investigate wether a lot of talk about illness on Twitter corresponds to a lot of people actually being sick in real life. Additionally, if this is found to be true, are we able to use the Twitter data to predict how many people will be sick on a given day? Or maybe in a given week?

There are a number of advantages that can be found if this is true. Among other things it could be used to estimate how much resources should be allocated to hospitals and other types of medical services. This could potentially translate into better medical service for the patients, since it's more likely that an adequate number of doctors and nurses are available. It could also save the medical services money, since it would be possible for the hospitals and medical services to allocate a suitable amount of personnel on a given day.

Another advantage with using Twitter for this is that in case it works it would allow us to monitor public health in close to real-time. This can allow us to detect outbreaks of diseases a lot faster than what would be possible to do otherwise. If we for example estimate that we would see 20 messages indicating that people are sick on a given day, but instead we find 60, we can probably assume that something has happened. This information can then be used to investigate what's going on. We might even be able to see that the tweets all seem to be coming from the same geographical area, and thus quickly be able to call for more doctors and nurses to the local hospitals if it is deemed necessary.

This project can be divided into a few smaller problems:

1. How do we automatically decide which Twitter messages indicate that someone is

sick?

2. Which real world data should we compare this to?
3. Is our collected Twitter data correlated with the real world data?
4. If so, how well does our approach compare to other approaches?
5. Can we use our Twitter data to predict future real world data?

The rest of this project report describes how I set out to solve these problems. It also chronicles other problems that I stumbled upon during this project and my efforts to solve these.

2.3 Previous Research

In the recent years there has been a lot of research into the area of social media. The research has focused heavily on the predictive power of social media. In 2010 Lampos et al.[26] used Twitter to track epidemics. Twitter has been shown to be able to give interesting results in a myriad of different topics, not just related to research relevant to health care. For example, in 2010 Sakaki et al. designed an earthquake detection system for use in Japan. This system was able to detect 96% of all the earthquakes of Japan Meteorological Agency (JMA) seismic intensity scale 3 or higher[32].

The approach of these research projects has been very different and it is worth contemplating which the best way of going about this is. One of the more interesting approaches I have come across is Collier et al. They looked at tweets indicative of self protective behavior[11]. They created five different categories, avoidance behavior, increased sanitation, seeking pharmaceutical intervention, wearing a mask and self reporting of influenza. They then classified 5,283 influenza tweets into one or more of these categories. They then trained a SVM classifier on this data and used the classifier on an already existing dataset. The best result thet found had a spearman's rho correlation of 0.68, with p-value of 0.008 when three categories were combined. This correlation was calculated against CDC[1] provided AH1N1[2] data.

There has been a limited amount of research in languages other than english, nevertheless some has been done. Pelat et al. studied the relations between the frequencies of queries for specific french search words and sentences to Google and surveillance data from the French Sentinel Network[30]. They found that there is a correlation between certain search queries and the reference data, for example the search query "grippe –aviaire –vaccin" had a Pearson correlation with the reported influenza data ($\rho = 0.82, p < 0.001$).

[1] http://www.cdc.gov/ The American Centers for Disease Control and Prevention.
[2] Influenza A virus subtype H1N1, the most common cause of influenza in 2009.

3 Introduction to Machine Learning

This section introduces the field of Machine Learning (ML) and shortly touches on the concepts that are relevant to the field in general and text categorization in particular. If you are somewhat familiar with the field of machine learning you can fairly confidently skip this chapter as it's very introductory and will not introduce any new concepts. This chapter is not necessary to understand the results of this project.

Machine Learning is generally considered a subfield of Artificial Intelligence. In Machine Learning we are concerned with finding patterns in large amounts of measured data. Machine learning techniques try to do this by building programs that improve with experience[29]. When we, for example have a large amount of data but very limited human expertise. It might be that human expertise is expensive, maybe we need a medical doctor to look at pictures of tumors to decide which ones are malignant. Lets say that only 1% of tumors are malignant, it would be very convenient to have a computer program remove the obviously benign ones and only have the doctor looking at the tumors the computer program is unsure of. It could also be that human expertise simply doesn't exist. Alpaydin brings up the task of speech recognition, almost everyone is able to do this without difficulty, however we are not able to explain how we do it. To solve this using machine learning we would collect a large amount of recordings of people uttering the same word. We would then apply one of our algorithms to these recordings in order to have them all be associated with the same word[2].
It should be noted that the selection of algorithms and concepts that are presented in this chapter is heavily biased by the choices I made during the course of the project, as well as the problems I had. Therefor this chapter should not be used by someone trying to get a general overview of machine learning or the algorithms associated with the field. In case you are interested in this, I would refer you to the excellent book "Introduction to Machine Learning" by Alpaydin[2].

3.1 Types of Learning

3.1.1 Supervised Learning

Supervised learning is the main part of this project, we have a number of documents (Tweets) and the corresponding label "relevant" (indicating sickness or not). We then want to find a model that maps the input document to the output label. More generally, in a supervised learning problem we have a set of documents, $d \in \mathbb{X}$, as well as a number of classes, or labels, $\mathbb{C} = c_1, c_2, ..., c_n$. As a starting point we then have a dataset which contains several labeled training instances, or documents $< d, c > \in \mathbb{X} \times \mathbb{C}$. We are then concerned with finding the classifier function γ that maps the input to the output:

$$\gamma : \mathbb{X} \rightarrow \mathbb{C} \tag{1}$$

We call this a supervised learning problem since we have to have a human supervisor labeling the different instances before our classifier can be created[28]. It is common

for the labeling process to be shared between several people. This is due to the fact that expert labeling can be very expensive, whereas non-expert labeling via for example Amazon Mechanical Turk[1] can be had at a low cost. It has been shown that the process of obtaining several labelers can improve the accuracy of the models learned from the data as well as the quality of the labeled data itself[34].

3.1.2 Unsupervised Learning

As the name suggest in unsupervised learning we don't have a human supervising the process. Whereas in supervised learning we have inputs that we are trying to map to already established labels, in an unsupervised problem we only have the input data. Then, instead of trying to find the mapping between input and output we are trying to find structures in the input. Clustering is an example of a unsupervised technique where we are trying to find similarities between the different inputs. This can for example be used to group customers into groups together with other customers with which they share similar characteristics[2].

3.1.3 Reinforcement Learning

In reinforcement learning as in unsupervised learning, we don't have a human supervisor. However, this does not mean that reinforcement learning and unsupervised learning are the same thing. In reinforcement learning we are instead using a trial-and-error based approach in which we are trying to maximize some kind of reward signal. This signal doesn't necessarily just focus on the last step, it can also be delayed. These two concepts are two of the most characteristic features of reinforcement learning[37]. Reinforcement learning have a lot of applications in different fields. Among other things it can be used to come up with novel strategies in games. TD-Gammon is an example of this, it is a program that learns to play back gammon using an artificial neural network combined with a learning algorithm called "Temporal Difference Learning". It then learned to play back gammon by playing against itself and learning from the results. Impressively TD-Gammon learned to play at a level just below the absolute world class players, and might even have come up with a few novel strategies[39].

3.2 Classification and Regression

Classification is the task of dividing examples into different classes, or categories, depending on the input attributes. This is what I've been trying to do in this project. We have a database of past tweets and a classifier deciding which category a specific tweet belongs to. That is, whether they indicated sickness or not, in other words if they were relevant. By then building a classifier from this data we hope that the classifier will be able to reliably place new data, in this case tweets, into the appropriate class.

Regression problems on the other hand consists of trying to predict numerical values.

[1] www.mturk.com

Examples of regression problems are for example price estimation. Let's say that we want to estimate the value of house. We would then use known data about the sales of other houses. The inputs to the system could then be the number of rooms, the size of the house, the location etc. The output would then of course be the prize that the house sold at. We then use this recorded data to create a mapping function between the input and the output value. We then use this function to estimate the value of a previously unknown house using our decided input attributes.

Since both classification and regression requires that we have measured data consisting of both input and outputs it's easy to realize that they are both supervised learning problems[2].

3.3 Classifiers

3.3.1 Multinomial Naive Bayes

The type of naive Bayes classifier that is commonly used for text categorization is called the multinomial naive Bayes classifier (MNB)[16]. Due to the fact that the categorization problem in this project is that of text categorization, it seems natural that is the task used to explain the MNB.

In the context of the MNB, we look at each document, in this case tweet, as a collection of words without caring about which order the words are in[16]. We then try to decide which class a document belongs to by computing $P(c|d)$, the probability that a document, d, belongs to category c. We then say that d belongs to the category c for which $P(c|d)$ is the highest. The probability of a document belonging to a certain class is calculated by[16]:

$$P(c|d) = \frac{P(c)\prod_{w\in d} P(w|c)^{n_{wd}}}{P(d)} \tag{2}$$

[16]

n_{wd} is the count of times the word w appears in the document, $P(w|c)$ is the probability of observing the word w given the category c. $P(d)$ is a constant that makes sure that the probabilities of the different classes sum to one. $P(c)$ is the a priori probability that a document belongs to c, it's estimated as the proportion of documents belonging to c[16]. $P(w|c)$ is calculated by:

$$P(w|c) = \frac{1 + \sum_{d\in D_c} n_{wd}}{k + \sum_{w'} \sum_{d\in D_c} n_{w'd}} \tag{3}$$

[16]

Where D_c is all the documents in class c. k is the number of unique words in the document collection.

3.3.2 Support Vector Machines (SVMs)

Support Vector Machines, or SVMs, are a type of supervised learning algorithm, the algorithm that is used as standard today was introduced in 1995[12]. It works by representing the different data points in space and then finding the maximum-margin hyperplane that separates the two different classes, in other words, the hyperplane that maximizes the distance from the hyperplane to the closest points of the different classes. For example, in a two-dimensional problem the maximum-margin hyperplane would be a line. In general it tries to find the hyperplane of dimension $n - 1$, where n is the dimension of the input data.

In it's original form SVM could only separate data that are linearly separable, that is, possible to be separated by a hyperplane. However, most interesting data is not necessarily linearly separable, this fortunately doesn't mean that SVMs are useless. What can be done is to apply what's called a kernel function. This kernel function then maps the non-linearly separable data into a feature space where the data can be linearly separable[5].

The kernels that are most commonly used are[18]:

Linear: $K(x_i, x_j) = x_i^T x_j$

Polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$

Radial Basis Function (RBF): $K(x_i, x_j) = exp(-\gamma ||x_i - x_j||^2), \gamma > 0$

sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

Where γ, d, r are kernel parameters[18].

3.3.3 Other Notable Classifiers

I have provided more detailed descriptions of SVMs and multinomial naive Bayes classifiers due to the fact that I use them in this project. There are however a lot of classifiers out there, so this section shortly introduces a few of the, in my opinion, more interesting ones.

Artificial Neural Network (ANN) this type of classifier is at least to some extent inspired by how biological learning systems use interconnected neurons[29]. It works by having a number of simple units, where each have a number of inputs and then returns one single output. These units can then in turn be used as input to another unit.

Decision Tree is probably one of the simplest algorithms for humans to intuitively understand. It works by classifying instances based on rules of the type "if x then y else z". Decision trees can be used both for classification (classification tree) and regression (regression tree). There are several algorithms for creating decision

trees from data, for example the C4.5 algorithm, which is the most well known of the decision tree algorithms[22].

Random Forest introduced in 2001[7], is an ensemble classifier, meaning that it is built up by several other classifiers. In this case it is as the name suggests built by combining several decision trees. The random in the name refers to how the individual decision trees are built. Each decision tree is then built using a subset of the original instances, as well as a subset of the variables used in the original in the classifier. The output of the final classifier is then the most common class among the individual decision trees. It has been used by, among others, Microsoft for pose recognition with the Kinect system[35].

3.4 Text Categorization

Text categorization is the process of labeling natural language text with describing categories from a set[33]. Currently this is mainly done using machine learning techniques to build a classifier that can automatically distinguish between different categories. In order to do this there are a number of preprocessing steps that can be taken in order to improve the classifier. This section introduces two such steps, it should also be noted that this naturally is not a comprehensive list and a number of other preprocessing steps are available.

3.4.1 Stop word Removal

Stop words are words that are very common in the dataset and are therefore not very likely to give away a lot of information regarding the class of the particular instance. Examples of stop words in English are "the", "is", "as" and so on. While a lot of stop words are task independent, there is also the possibility of having task dependent stop words[36]. Since these words don't provide any help with the classification it can be a good idea to remove these words, since it will work as a dimension reduction[36]. Silva and Ribeiro (2003) also showed that stop word removal can have very positive effects on the recall of the resulting classifier[36]. In this project I only used a general purpose stop list, further examination of the content of the tweets could result in a more specific list of stop words. For a complete list of stop words used, see table 1.

3.4.2 Stemming

Stemming is the act of transforming a word into it's corresponding word stem. This process can involve the removal of both suffixes and prefixes from words. Stemming is useful to ensure that words get appropriate weights when they essentially mean the same thing. For example the word "hundarnas" ("the dogs'" in genitive form) would be transformed to the word "hund" ("dog")[8]. Stemming has been shown to improve both precision and recall values in a number of languages, among those Swedish[8].

The use of stemming was something I considered for a long time, however, due to the

alla, allt, alltså, andra, att, bara, bli, blir, borde, bra, mitt, ser, dem, den, denna, det, detta, dig, din, dock, dom, där, edit, efter, eftersom, eller, ett, fast, fel, fick, finns, fram, från, får, fått, för, första, genom, ger, går, gör, göra, hade, han, har, hela, helt, honom, hur, här, iaf, igen, ingen, inget, inte, jag, kan, kanske, kommer, lika, lite, man, med, men, mer, mig, min, mot, mycket, många, måste, nog, när, någon, något, några, nån, nåt, och, också, rätt, samma, sedan, sen, sig, sin, själv, ska, skulle, som, sätt, tar, till, tror, tycker, typ, upp, utan, vad, var, vara, vet, vid, vilket, vill, väl, även, över

Table 1: The stop words used in the classifier.

good performance of the classifier I eventually decided against it, in order to ensure adequate amount of time for the rest of the project. However, investigating how stemming would have influenced the performance of the classifier on the dataset would make for an interesting future experiment.

3.5 Machine Learning Problems Relating to this Project

3.5.1 Overfitting

In a supervised learning problem we are trying to find the function that best maps input data to output data. As explained in section 3.1.1 we do this using a dataset of examples and their corresponding labels. The problem with this approach is that we measure the accuracy of our classifier on already known data, but we then try to use the classifier to make predictions on new data. Thus, the object is not to maximize the accuracy on the training data, but the unknown data. This means that if we try too hard to optimize the performance accuracy on the training data we might accidently fit our model to recognize noise in the training data. This problem is known as overfitting[13].

3.5.2 Dealing with Unbalanced Classes

This project has had to deal with the situation that tweets indicating sickness are comparably rare. More concrete numbers and statistics are presented further into the report.

To illustrate the problem with unbalanced classes, consider the case where we have trained a classifier to predict that a message is in category one with an error of 5%, but the message is in fact in category one 96% of the time. In this case, a classifier that simply predicted that any given message is part of category one would have an error of only 4%, which is better than our classifier but clearly not a very good classifier anyway.

Telling the world that you are sick on Twitter is not very common. In this project only around 0.3% of the observed tweets indicated that someone was sick. This leads to a few problems. Among other things, it's hard to correctly detect these tweets, so in order to simplify this there are a few techniques that can be applied.

Oversampling the minority class: This method means that we artificially increase the ratio of the minority class in the training set. This is usually done by randomly replicating minority class instances in the dataset. One problem with this technique is that it might increase overfitting[23].

Undersampling the majority class: Just like oversampling the minority class, undersampling the majority class aims to increase the ratio of positive and negative instances. This can, as in the case of this project, be done randomly, or it can be done by using some type of ranking to decide which examples to delete. One way to decide which examples to delete is by trying to choose the ones far away from the decision border[24].

SMOTE: SMOTE, or Synthetic Minority Over-sampling Technique[9] is a technique for oversampling the minority class. It works by synthetically creating new examples of the minority class by interpolating between the minority class example and its nearest neighbors[23]. It's a very interesting idea, but unfortunately I haven't been able to incorporate it into my final classifier.

3.6 Evaluating Performance

3.6.1 K-Fold Cross Validation

K-Fold Cross Validation is one of the most common ways of evaluating the performance of a classifier. It works by splitting the dataset into k equal, or close to equal subsets and then training on $k-1$ of these subsets, and testing on the remaining one. This process is then repeated until we have used every subset as the testing subset[20]. The k in k-fold refers to how many subsets the dataset is divided into. It is very common to use ten folds, however in this project I have used five folds, since the computing power I've had access to have been very limited.

3.6.2 Precision and Recall

In this project we have largely been working on a problem where there is a skewing of the classes. This means that there is a lot more messages that are classified as not indicative of an illness compared to the amount of messages that are not. This means that we can't rely on only the percentage of errors that are either classified correctly or wrongly.

Because of these problems we use two additional ways of measuring the performance of any given classifier, called **precision** and **recall**. Precision and recall are defined respectively as $P = \frac{C}{M}$ and $R = \frac{C}{N}$, where C is the number of correct slots (true positives)[27], M is the sum of true positives and false positives and N is the sum of true positives and false negatives.

Kubat and Matwin[24] use the geometric mean: $g = \sqrt{a^+ * a^-}$. Where a^+ and a^- are

the accuracies observed on the positive and negative examples respectively. However, they also note that the metric F-measure, introduced by Kononenko and Bratko[21], has a lot of positive traits and therefor that is the measurement I will be using when evaluating classifier performance in this project.

3.6.3 F-Measure

The F-measure is a metric that combines the precision and recall in one. It is defined as the harmonic mean of precision and recall[28].

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{4}$$

Where P is the precision of the classifier and R the recall. By adjusting the value of β the measurement can emphasize either the precision or recall of the classifier. If β is < 1 the measurement will emphasize precision and conversely values > 1 will value recall more[28]. When $\beta = 1$ we get the most commonly used version of the F-measure, normally called F_1 which is short for $F_{\beta=1}$:

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5}$$

Whenever this report makes a mention of the F-measure it refers to the F_1 metric.

4 Methodology

4.1 External Dependencies

In this project I have made use of a number of external software components. The reasons for this varies, but mainly it has had to do with time saving. A lot of the problems related to the implementation of this project have already been solved, and instead of reinventing the wheel I have decided to use the already implemented solutions. This have allowed me to spend more time on the core problems of this project. This section introduces the external software I have used and explains the reasons for the use of these.

4.1.1 Twitter4J

Twitter4J[1] is the library that I use in order to simplify the interaction with the Twitter servers. It contains all of the functionality that I need to have in order to collect tweets from the Twitter servers. Like Weka and R it is open source, it allows us to use it for any purpose, even commercially. Since Twitter4J already had all the functionality I needed when it came to handling tweets, it was not necessary for me to build my own component for interaction with the Twitter servers.

[1] http://twitter4j.org

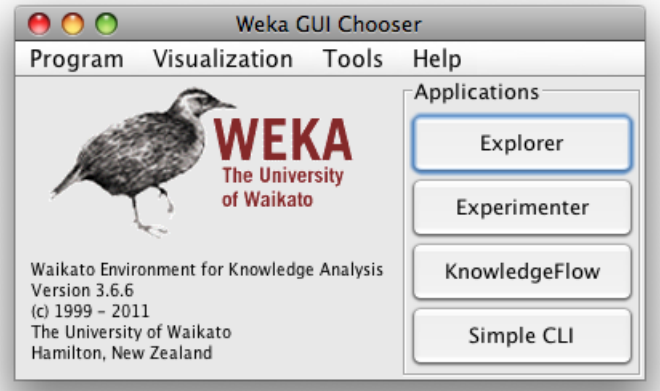

Figure 1: A screen shot showing the GUI chooser for the Weka data mining software.

4.1.2 Weka

Weka[17] is a open source toolbox containing a collection of machine learning algorithms developed by the Machine Learning Group at the University of Waikato in New Zealand. It contains tools for performing preprocessing tasks, classification, regression, clustering as well as other functions. It is divided into four main modules, explorer, experimenter, knowledgeFlow and Simple CLI.

The explorer contains a GUI that let us load data-files, apply preprocessing filters as well as applying all the different classifiers and regression algorithms that Weka contain. The experimenter allows us to run several experiments in a way that is easier than doing it individually. The knowledgleFlow let us define a flow for our data by visually design the order in which we want our information to flow. For example we might put a filter before a classifier. The SimpleCLI is a way for us to run commands to Weka without using a GUI. In this project I have only used the explorer and the SimpleCLI.

4.1.3 R

R[38] is a free software under the GNU-license that allows us to perform a variety of different computations and visualizations on data. In a way it's quite similar to MATLAB. R can be thought of as a different implementation of the S programming language. R contains a lot of different mathematical functions for dealing with time series, such as the possibility to automatically compute the autocorrelation function of a time series and visualize it in a compelling manner. All plots of time series data in this project have been created using R.

4.1.4 PostgreSQL

PostgreSQL is an open source database system. PostgreSQL allows you to download a Jar-file which contains everything that is needed in order to work with a database from your Java-code. I have used PostgreSQL since it is fairly easy to work with, and since I have some previous experience with using it I got it up and running fairly quickly. The database choice is not very important in this project since we never work with an extreme amount of data. In case this system would be implemented in a real world setting I would recommend evaluating which database to use more thoroughly.

4.2 Data Collection

4.2.1 Description

In order to access the Twitter feeds I use a third party Java library called Twitter4J. This library supports most of the things that I need in order to access tweets. In order to keep the Twitter-module separate from the rest of the code all of the functionality have been put in a wrapper class, TwitterWrapper.java that is only used to call the library with the necessary function calls and then handle the responses from the Twitter servers and format the data in the desired way.

In reality the data provided by the underlying Twitter api uses the JSON-format to store the data. However, due to the fact that I use the Twitter4J library instead of directly calling the api I don't have to deal directly with the JSON-format, this is done by the Twitter4J library.

Date Handling in tweets: Dates in Twitter are in a very verbose format, because of this it is important that we have a method to parse these date formats since it's much simpler than trying to work with strings. By creating a new SimpleDateFormat object to parse we get the following layout of the parser: EEE MMM dd HH:mm:ss zzz yyyy, where the each letter corresponds to a specific date attribute as specified in the SimpleDateFormat class. In order the meaning is weekday, month, numbered date, hour, minute, seconds, time-zone and year. So an acceptable date could be "Sat Feb 18 20:10:40 CET 2012". The convertTwitterDate method takes one string in the given format and returns a Date-object with the specific properties.

Regional concerns: Due to the fact that I only have access to reported verification data from Sweden, I need to make sure that we only focus on Tweets from inside of Sweden. In practice this is hard to do with the Twitter api. The way I have solved this is by using Twitters built in support for extracting Tweets within a specific geolocation. Twitters search function can take an optional argument called "geocode", which takes four arguments, the first two define a geographical position, with it's latitude and longitude, and one number which corresponds to the area of a circle, the fourth argument is either km or mi depending on whether the area of the circle is defined in kilometers or miles. Twitter then uses the geographical location as the center of a circle with the

Latitude	Longitude	Radius
58.979	14.634	250km
62.158	15.051	275km
66.437	19.797	275km

Table 2: The positions and radii of the three chosen geographical centers.

defined radius in kilometers or miles, when this is done Twitter only gives us tweets that have been geotagged within this circle.

Since Sweden is not a circle I have divided up Sweden into three smaller regions, each region is defined by a center and a radius, the locations and radii of these positions can be seen in table 2. Unfortunately it has not been possible for me to restrict this area entirely to Sweden, therefor tweets from parts of Norway may also been included into the data set. Additionally, parts of the circles overlap, making it possible that one tweet will be included in two search queries, in order to avoid this each tweets individual id is checked to make sure that it has not already been entered into the database.

This project focuses on the spread of diseases in Sweden, and in order to make sure that we don't try to use a classifier with two or more different languages we need to make sure that we only use tweets that are in Swedish. This can also be done via the Twitter search query, but not without using a search keyword, and since we often are interested in fetching all tweets during a specific time period using a specific keyword would induce bias into the system, which we would like to avoid, thus we would also like to avoid using the language as an argument in the search query. Fortunately this is not very hard to do, every tweet is tagged with a ISO language code, which for Sweden is "sv". Therefor all we need to do is to extract this code, which can be done with the Twitter4J method $Tweet.getIsoLanguageCode()$ which returns a string with the language code corresponding to the specific tweet. We can then just remove all tweets not tagged with the Swedish language code.

4.2.2 Rate Limiting

The Twitter API does not allow unlimited requests to their server. Instead they limit the amount of requests to 150 per hour for unauthenticated calls and 350 per hour for authenticated calls. The way this is implemented is by limiting the amount of requests from a specific ip-address for unauthenticated calls and to the token specific for the application in the case of authenticated requests.

4.2.3 Description of the tagging process

The training data was acquired by using the Twitter search API to download as many tweets in Swedish as possible by the method described in section 4.2. A short summary is that data was collected by searching for tweets with a tagged geo-location that is

Original Tweet	English Translation
såklart är man sjuk när de är fredag och allt	Of course one is sick when it's friday and everything
Mördarförkylningen! http://t.co/5yo0eVOE	Killer cold! http://t.co/5yo0eVOE
Förkylningen håller på att försvinna! TACK	The cold is about to go away! Thanks
Mår illa och går hem. JAG VILL INTE	Feeling ill and going home. I DON'T WANT TO
kollar på tv och är sjuk :/ snälla vill ha ett bättre liv just nu!!!!!	watching tv and being sick :/ please want a better life right now!!!

Table 3: Examples of tweets belonging to the positive (minority) class.

Original Tweet	English Translation
@username HAHAHA tänker du på Valborg?	@username HAHAHA are you thinking of Valborg [Walpurgis night]?
@username Tack! Ha en fin dag	@username Thanks! Have a nice day
6 månader kärlek	6 months love
Snöstorm och sol....	Snowstorm and sun....
@username Gå ut i solen!	@username Go out in the sun!

Table 4: Examples of tweets belonging to the negative (majority) class.

contained within the three circles mentioned in table 2. All tweets were then scanned for their language tag in order to make sure they were recognized by Twitter as being written in Swedish, after which the data was stored in the database. For the purpose of gathering training data Twitter was polled for new tweets on an average of once a day.

Once the tweets were stored in the database the tagging process commenced. In general tweets that were indicative of someone being sick were tagged as relevant, meaning the relevant column in the tweet-table for the indicated tweet was set to true, correspondingly tweets that were not indicative of someone being sick was marked as irrelevant by setting the relevant flag to false. It is important to note here that all tweets that were indicative of someone being sick were classified as relevant. This means that tweets that indicated that a third party was sick was classified as true. This could happen for example if a mother is tweeting about being home with a sick child or someone telling their friend to feel better.

For the purpose of this study only temporary illnesses were considered, no injuries were included. This means that tweets regarding influenza and similar diseases were classified as relevant whereas broken bones and cancer tweets were not considered. Of course a lot of tweets are somewhat dependent on the tweet history, for example a tweet with the text "I feel so sick right now" would be classified as positive even if the users tweet history would indicate that the user suffers from cancer. This is something that I do

not consider in this project, taking the tweet history into consideration would require a much more sophisticated system.

4.3 Project Approach

4.3.1 Initial approach

In an initial approach two hundred samples where used, 100 positive samples and 100 negative. Since the multinomial naive bayes classifier traditionally have been one of the standard classifier to use for text classification this seems like a good place to start. The initial experiment was conducted using Weka (see section 4.1.2). First an arff-file was created consisting of two fields, text, in this case the text in the tweet, as well as the category that the specific instance (tweet) belong to, this is a binary choice of true or false, true in case the instance indicates that someone is sick, otherwise false

Wekas built in preprocessor was then used, in the first attempt only the filter StringToWordVector with loweCaseTokens set to true was applied. The StringToWordVector takes a string, and converts into a vector consisting of the individual words from that string. This is necessary because the multinomial naive bayes classifier doesn't work on strings directly. lowerCaseTokens just converts all upper case letters to lower case ones. For this first attempt no other preprocessing was done.

The multinomial naive bayes classifier was then built using Wekas built in methods. For testing the classifier I used 10-fold cross validation. The resulting classifier uses 1150 different attributes to make the classification. For now we are only using different words to make the classification so our attributes are just words. The classifier classified 75.5% of the instances correctly, with a weighted average precision of 77.5% and recall of 75.5%.

Words that only recur once are usually not very good to use since they skew the results. By definition a word occurring only once has to belong to just one category, however there are no single words, or at least very few, that can't mean the opposite if there is a negating word somewhere in front of it. For example the sentence "I have influenza" might generally indicate that someone is sick, however by inserting "don't" we get "I don't have influenza" which probably means that someone isn't sick. Because of this we take the next step in the preprocessing and remove all words that don't occur at least twice in the dataset. In this dataset we only have 200 instances meaning that a lot of the words only occur once, this filtering actually removes most of the attributes, leaving 236 words that occur twice or more.

Not very surprisingly the resulting classifier performs better than the original classifier, classifying 78% of the instances correctly with a weighted precision of 80.8% and recall of 78%. We can give the performance a significant boost by applying yet another few preprocessing steps. By introducing a list of stop words to remove before classification, the list in question is the stop word list that is used in the PunBB forum software.

After this all words that were shorter than 3 characters were also removed, since they were not considered important. The resulting classifier shows large improvement, correctly classifying 85% of all instanced. Both the precision and recall show improvement, with precision up to an average of 87.1% and recall up to 85%. Interestingly we also have a very good recall for the class consisting of positive instances, in this class recall is as high as 97%. In the complete dataset positive instances are very rare, making them more valuable. Since they are so rare it's more important to have a good recall in this class, which means that we can accept a bit more miss classifications in the negative category if that means we classify all, or close to all of the positive instances correctly.

4.3.2 Continued experiments

While the multinomial naive bayes classifier have shown decent results during the initial testing it turns out that it does not perform very well when we use a larger dataset. In my experiments I have applied some oversampling of the minority class. Meaning that all instances in the minority class have been sampled twice. It has been shown that non-random sampling which favors the minority class can greatly improve the performance of the classifier when we work with skewed class distributions[15].

In addition to the oversampling of the minority class I also choose to under sample the majority class. This was done by removing parts of the dataset where the instance were classified as not relevant. The results using this technique with a multinomial naive bayes classifier, tested using 5-fold cross validation resulted in a classifier with an accuracy of 88.02% with the average precision 0.984 and recall of 0.88. Unfortunately this classifier is not very good at classifying the minority class which is the most important part in this particular task. In fact it only returns a precision of 0.104 and recall of 0.887 for the minority class. This means that 89.6% of the tweets that were classified as relevant, i.e. indicating that someone is sick were false positives.

Due to the bad results generated by the multinomial naive bayes classifier I decided to use another classifier very common in text classification tasks, namely the Support Vector Machine (SVM). Using SVM for text categorization have been shown to produce very good results. Dumais et al. show that a linear SVM outperformed the naive bayes classifier, with the SVM averaging 87% accuracy over all categories versus the naive bayes classifier which had an accuracy of 75.2%[14].

As the results of [14] indicate our SVM classifier outperforms the multinomial naive bayes classifier by a large margin. Just as with the multinomial naive bayes classifier we used 5-fold cross validation to evaluate our results. The resulting classifier was surprisingly good, having an accuracy of 99.58% with an average precision of 99.6%, the average recall was also 99.6%. It turns out that the SVM also significantly outperforms the multinomial naive bayes classifier when it comes to classifying the minority class. Where the multinomial naive bayes classifier had a precision of 0.104 and a recall value of 0.887 the SVM classifier have a precision of 0.919 and a recall of 0.806. This means

that the SVM classifier misses more instances of the minority class but instead barely have any false positives at all.

4.4 Preprocessing

4.4.1 Removing usernames

A lot of tweets are directed at someone, containing the tag @username. A tweet containing a username is a way to reply to this user or start a new conversation with them. If your username is included in a tweet you'll be notified by Twitter.

Usernames are unlikely to contain any useful information in themselves. Since I am only interested in finding out whether a tweet indicates that someone is sick usernames are likely useless. However, due to the fact that we have a very limited amount of positive training examples it is likely that a classifier trained on a corpus with very limited data would be vulnerable to over-fitting and take into account who a tweet is directed in it's classification.

Consider for example that a tweet: "@user I have the flu" was encountered in the manually labeled training data. This tweet would naturally be classified as a positive tweets since it implies that the poster currently suffers from the flu. Later when a classifier is trained to determine" whether a tweet is positive or negative it won't differentiate between the words that actually indicate sickness, in this case "flu" and words that don't ("@user", "I", "have", "the""). Many of these words are likely to be removed from the text that's actually used for classification anyway since they are likely to be included in the list of stop words. However, "@user" won't be removed since it is impossible to include all possible usernames in the list of stop words. Thus we have to remove them manually. Luckily for us the rules for what username can be selected for a Twitter user is quite restricted. We can therefore apply a simple regular expression ("@[A-Za-z0-9_]1,20") to the text before inputting it into the classifier to remove all usernames.

It's important to note that this will also remove parts of an email address. For example, this regular expression will also match the string "erik@something.com" and thusly remove part of the string ("@something"). I don't consider this a negative since e-mail addresses are also unlikely to add any actual value to a tweet.

4.4.2 Removing Retweets

A retweet in Twitter is structurally the equivalent of email forwarding[6]. Unfortunately, there is no uniform universally agreed upon standard for denoting a retweet, however the structure "RT @user" followed by the original message is one common way of denoting a retweet[6].

Retweets, as it pertains to this project can be quite damaging for my results. This

is because if a tweet is classified as positive I assume that it indicates that someone is sick, however if one of these tweets then get retweeted it's unlikely that the person doing the retweeting the message is sick as well. Instead it is much likelier that he or she is simply telling their followers that the original tweeter is sick as to spread that information. The problem is that a classifier really have no way of knowing that a retweet most likely don't imply that the poster is sick, and therefor the classifier is likely to classify the retweet as positive as well, meaning that we know have two positive instances where there's probably only one sick person. Obviously this is very likely to influence and decrease the accuracy of my model.

4.4.3 Detecting Retweets

As mentioned in section 4.4.2 there is no standardized way of indicating that a tweet is a retweet. However, there are a few ways of finding them anyway. For example[6] claim that, as stated in 4.4.2 the typical way of doing this is with the syntax "RT @user message". They did also note that there is a number of other ways of denoting a retweet, for example via the syntax "RT: @", "(via @)".

In their paper Ruiz et al. used the Jaccard distance between the bag of words of the two different tweets to find retweets[31]. In this project I have decided to take a very simplistic approach to finding retweets. I have simply designed a regular expression to find the syntax "RT @user message" and remove the tweets that match this regular expression. The reason for my simplistic approach is that when I during the project analyzed my dataset and looked at which retweets were getting classified as positive I noticed that they all shared the common syntax "RT @user message". It should be noted that they didn't all start the message with "RT @user", but it was always present somewhere in the message. The retweets where found by looking for messages in the dataset that contained the word "RT" and were classified as positive, therefor it's possible that some retweets using not using "RT" to indicate that it's a retweet might be missed.

Looking at the dataset [2012-04-23] there were 32 retweets that had been classified as positive, out of a total of 930 positive instances. It is also important to note that while the case was that all the positively classified retweets fulfilled this description, it by no means mean that all retweets do this. However, during the course of the project the system have collected over half a million twitter messages. This means that it would be close to impossible to go through all of these to find all the ways retweets have been designed.

4.4.4 Results of Preprocessing

Table 4.4.4 describes the results that the different steps of preprocessing had on the resulting classifier. All of the tests where carried out using an SVM classifier and tested using 5-fold cross validation. The results that are described in table 4.4.4 all refer to the results on the minority class, except for the accuracy which refers to the overall accuracy

Preprocessing	Accuracy	Precision	Recall	F-Measure
Nothing extra	0.9958	0.919	0.806	0.859
Removal of usernames	0.9952	0.892	0.787	0.836
Removal of retweets	**0.9972**	**0.94**	**0.896**	**0.917**
Removal of usernames and retweets	0.9962	0.919	0.858	0.887

Table 5: The performance of the classifier depending on what type of preprocessing is done.

of the classifier. The reason for not listing the results of the majority class is that they are so close that it is very unlikely that they are influencing the resulting classifier in a meaningful way. The results listed in table 4.4.4 are all on the manually labeled test data, as such a better result does not necessarily correlate with a better result on the unlabeled data. This is because it is possible that our classifier overfits, that is adjusts itself too much to the training data, and this could lead to weakened performance when we remove data from the training set. This should still lead to better performance on unknown data. As it turns out removing usernames from the tweets before classifying them does indeed lead to worse performance on the training data as shown in table 4.4.4.

A likely reason to why removal of usernames leads to weakened performance when testing is that the classifier ends up overfitting to the training data, causing better results on this data, but is likely to cause decreased performance on new data.

It is interesting to note that the removal of retweets lead to increased performance on the training data.

4.4.5 StringToWordVector

StringToWordVector is a filter that is built in to Weka. It's responsible for converting a category of documents into individual categories for each part of text from the documents. In our case, we start out by having two different categories in our arff-file. The categories are "text" and "category". The category "text" refers to the tweet and "category" refers to if it belongs to the minority class by having the value "true", or the majority, in which case it will have the value "false". StringToWordVector will take the "text"-category, which is a attribute and convert it into a vector that represents word occurrence frequencies[40].

- **IDFTransform** this is the inverse document frequency transformation[3].
- **TFTransform** this is the term frequency transformation. In Weka this means that frequencies are transformed into $\log(1 + f_{i,j})$[3].

- **attributeIndices** this parameter sets which attributes to work on[3].
- **attributeNamePrefix** if we want to have a prefix for names of the attributes, it's set here.
- **doNotOperateOnPerClassBasis** set this to true if you don't want some of the other attributes to be set on per class basis.
- **lowerCaseTokens** decides if all individual parts of text should be converted to lower case or not.
- **minTermFreq** the amount of times a part of text needs to be used in the corpus for it to not be removed. This is very useful for removing words that are not used a lot, for example by setting this attribute to two, we can remove hapex legomena, which are words that only appear once[40].
- **normalizeDocLength** decided if the word frequencies should be normalized[3].
- **outputWordCounts** whether or not word counts or word presence should be outputted.
- **periodicPruning** decides if we should periodically prune the dictionary or not, and if so at which rate we should do so[3].
- **stemmer** if we want to use a stemmer and if so which one. For more information on what a stemmer is, see section 3.4.2
- **Stopwords** A file containing a list of stop words to be removed from the corpus. This list should contain each word on a new line.
- **tokenizer** Weka allows us to use a number of tokenizers. A tokenizer decides how to divide text into word. This attribute allows us to choose which one to use.
- **useStopList** a boolean that decides whether stop words are removed or not. As standard it is set to true if we have loaded a list of stop words.
- **wordsToKeep** this attribute sets the number of words that we would like to keep after the transformation to word vectors. It is not an exact metric, so if, for example, we set it to 10000 it might save slightly more or slightly less than 10000 unique words.

5 Final Implementation

5.1 Database layout

This section aims to give the reader a description of the database and the its tables. The database is built around three different tables, of which only one is completely necessary.

Attribute	Value
IDFTransform	True
TFTransform	True
attributeIndices	first-last
attributeNamePrefix	
doNotOperateOnPerClassBasis	False
lowerCaseTokens	True
minTermFreq	2
normalizeDocLength	Normalize all data
outputWordCounts	False
periodicPruning	-1.0
stemmer	NullStemmer
Stopwords	See table 1 for complete list
tokenizer	AlphabeticTokenizer
useStopList	True
wordsToKeep	10000

Table 6: The settings used for the Weka filter StringToWordVector.

This table is the Tweet table which contains all of the observed tweets as well as a few characteristics of the tweet. The tweetsbydate table can be generated from the tweet table. The ttdatum table is essentially used as a help table for the generation and update of the tweetsbydate table. From the beginning ttdatum was created to make sure that no dates where missed when generating the tweetsbydate table.

5.2 Tweet

text	date	geo_code	relevant	id (primary key)	region

Table 7: The fields of the tweet table.

The tweet table is as the introduction suggested the heart of this project. It contains all the tweets that have been collected during the course of this project. Aside from the tweets themselves it also contains the corresponding date and time that the tweet was posted at, the geographical coordinates if they are available, the id of the tweet as well as the geographical region that we queried when we observed the tweet.

The text field is quite self-explanatory, it's simply the text of the tweet in question, the message that the poster is trying to get across. The date is the date and time that the message was posted, for example it could be "Mon Apr 23 02:11:52 CEST 2012". The geo_location field is not always available, I'm not entirely sure why, but it might

have something to do with privacy options or something similar within Twitter. If the geographical location is available, it is saved in the form GeoLocation{latitude=59.3398, longitude=14.5129}. The ID is the tweets individual ID, the ID is used as the primary key in the table, since it already uniquely identifies a tweet I decided that it was better to use this existing ID instead of creating a new internal ID as primary key. The region is the geographical region from where the tweet was collected. As explained the software makes three requests to the Twitter server, one for each Swedish region, the options here are "South", "Central" and "North".

5.2.1 TweetsByDate

ttdatum	region	num_tweets	numpostweets	observedhours	numpostweetsadjusted

Table 8: The fields of the tweetsbydate table.

The tweetsbydate table stores the details of the tweet table, grouped by date and region. It is not an essential table, it could in theory be generated on the fly by joining different fields in the tweet table. In fact, that is how the tweetsbydate table was originally created. Maybe the most intuitive way of viewing the tweetsbydate table is to see it as a cache for the tweet table. Except for the numpostweetsadjusted field, which ended up not being used in the final implementation, it doesn't contain any original information. However, the amount of joins that are needed to create the table is quite substantial, so while it might work to do them in real time right now it is not feasible once the implementation have been running nonstop for a significant amount of time.

The table consists of the ttdatum field, which is a shortened version of the date in question. The region is the same as in the tweet table, "South", "North" or "Central". Each combination of date and region will have its own entry in the database. This means that in total each unique date can have a total of three entries. The reason for this is that I want to be able to look at the possible regional differences. The field num_tweets is simply the amount tweets observed in the region in question during the given day. Just as intuitive is the numpostweets, this field contains the amount of tweets that were classified as positive during the relevant day, in the affected region.

The observedhours field might be slightly confusing. What it describes is the number of unique hours from which we have observed tweets during a given day and region. Its meaning might be easier understood using an example, consider the very unlikely scenario where we have only observed three tweets for a given date and region. Let's say that one tweet was observed at 17:21, one at 21:22 and the last one at 21:43. We would then have observed tweets during two distinct hours, the hours 17 and 21. This means that in this case the observedhours field would have the value 2.

The numpostweetsadjusted field is the only field in this table that is not generated from the tweet table. In this project it was used for experiments when it came to adjusting for missing values in the database. Currently the values here contain the values adjusted with loess regression, see section 7.3.2. In the end I decided against using loess regression to adjust for missing values since it didn't produce good results, so in the current implementation it's not used. I however decided to leave this in the database anyway, since it could be a useful field in the future.

5.2.2 Tdatum

date

Table 9: The fields of the Tdatum table.

The Tdatum table only contain one field, date. This date is in the same format as in the tweetsbydate table. This table only contains a list of dates, it was created to assist with the generation of the tweetsbydate table. In the final implementation it's only used to simplify certain sql joins. It could be removed as long as a few sql queries were changed to not depend on this table.

5.3 Program implementation

As can be seen in figure 2, the entire implementation is very centered around the class TwitterMain. This class is responsible for the continued running of the program, and in turn calls the other classes who performs very specific functions. In figure 2 direct connections between classes are indicated with a thick line and dependencies to third party packages are indicated with dotted lines.

5.3.1 TwitterMain

This is the main part of the program, essentially it is an infinite loop. It starts by obtaining a list of tweets from the class TwitterWrapper, it then calls DBWrapper and converts this list of tweets to a list of ExtractedTweets. The list of converted tweets is then passed off to Preprocessor, which perform all the preprocessing of the text such as removing retweets. We then send the list of ExtractedTweets to the Weka class where each instance is classified. The list of preprocessed and classified tweets are then passed of to DBWrapper again for insertion into the database. Lastly we have another call to the DBWrapper class and have it update the TweetsByDate and Tdatum tables.

This process is then repeated three times, one for each of the parts of Sweden. Once this process is done we call REvaluator and it does the predictions for the day.

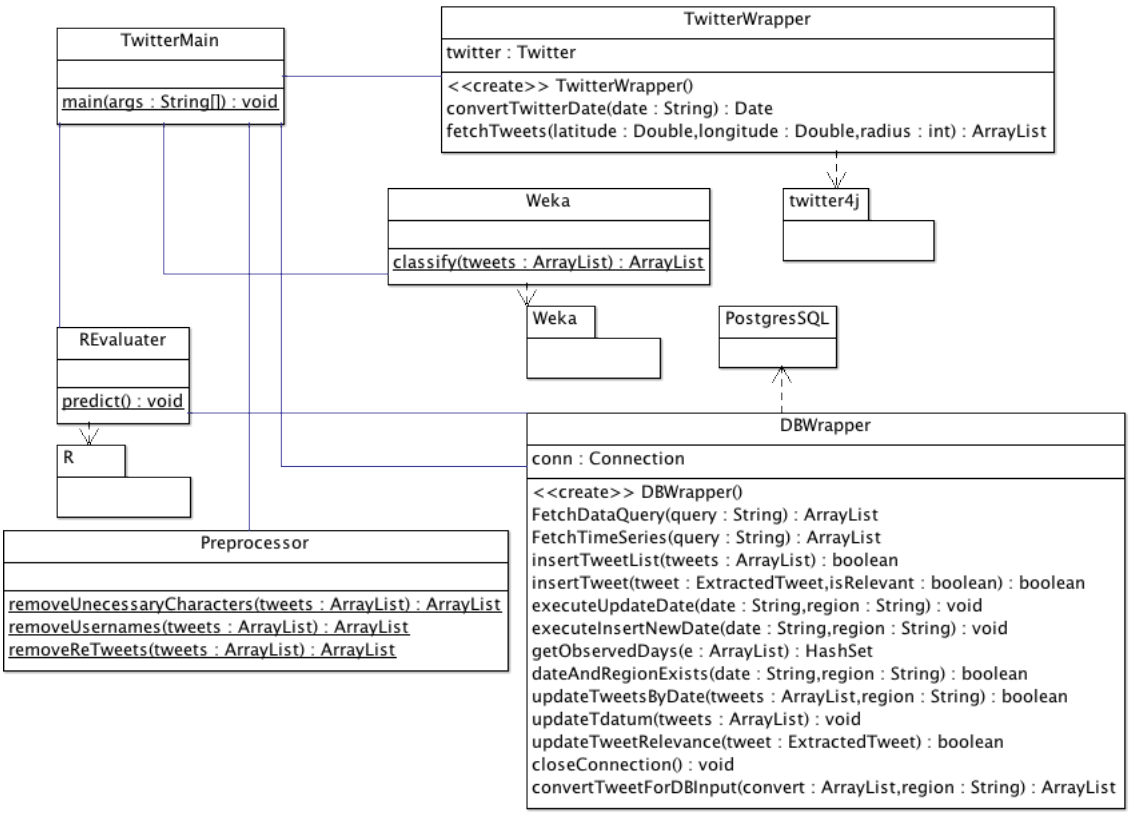

Figure 2: Rough outline of the relationships between the classes as well as the dependencies.

5.3.2 TwitterWrapper

The TwitterWrapper class is the only class which directly talks to Twitter4J, which in turn talks to the Twitter servers. The main part of this class is the fetchTweets method, which takes a latitude, longitude and a radius. It then creates a query, by setting the geolocation, radius and a constant to indicate that we are using kilometers for the radius. The method then continues to query the Twitter servers until we receive less tweets than 100, which is the maximum amount of tweets that can be retrieved at one time. The way this works is that if we for example detect 590 new tweets, it will come in in six pages. Five of these will include 100 tweets and one will include 90. When we see that the last page only has 90 tweets we know that it is the last page. Each individual tweet is then checked to make sure that the language set is Swedish, in order to make sure that we don't confuse our classifier. In case that the tweet is in some other language it is dropped. The list of tweets in Swedish is then returned.

5.3.3 ExtractedTweet

The ExtractedTweet class is not present in figure 2, the reason for this is that it is only a reduced version of the Tweet-class present in the Twitter4J package. What it does is that in contains variable describing a unique tweet. It also contains getter and setter methods for these variables. The reason for this class is only to simplify database handling, since the Tweet table in the database does not contain all the fields available in the Twitter4J package.

5.3.4 DBWrapper

DBWrapper is the class that is, as its name suggests, responsible for all interaction with the database. To do do this it have a number of methods that can execute queries on the database as well as helper functions to do this.

5.3.5 Preprocessor

This class is the one that contains methods to do preprocessing on tweets before they are inserted into the database. Currently it contains three methods, removeReTweets, removeUsernames and removeUnecessaryCharacters. These methods do what the name suggests, removeReTweets removes tweets which are likely to be retweets, that is a copy of another tweet. removeUsernames removes usernames such as "@username" from tweets. The name of the method removeUnecessaryCharacters might not be as telling as the other two methods. What it does is that it removes apostrophes, line breaks and back slashes from the tweet text.

This method was mainly necessary earlier in the project, during the experimenting stage when I started working with Weka I used wekas arff-files for input. Apostrofes, line breaks and backslashes ended up breaking the arff-files causing the input process to

abort. The removal of all this is not entirely necessary anymore, but it is kept to keep the consistency between older instances and newer ones.

5.3.6 Weka

The Weka class is made in order to interact with the Weka-package. It only contains one method, predict. The predict method takes a list of extracted tweet and returns the same list but with the different instances classified as either positive or negative.

It loads a filtered classifier, which has been created using the Weka software earlier. A filtered classifier in Weka is a sort of meta-classifier that combines a classifier and a filter. The filter in this case is the weka stringToWordVector filter and the classifier is the SVM that I created earlier.

The reason that we use a filtered classifier instead of using the SVM classifier straight up is that the instances need to be converted into vectors of individual words. When this is done each word is mapped to a number, or a dictionary, and this dictionary is unique for the input file, but when this is done for one file, and then later for another file, which is the case here, we end up with files with different dictionaries. This means that we can't compare them and thus the classifier won't work.

A filtered classifier avoids this problem, so that when in this program we classify new instances it uses the same stringToWordVector filter that was used on the training data set.

The predict method then uses this filtered classifier to classify each instance in the ExtractedTweet list, updates the list and returns the resulting list.

5.3.7 REvaluator

REvaluator is in my opinion the most interesting class in the implementation, since it combines two different programming languages, Java and R in one class. The class uses the external package JRI in order to allow the connection between java and R in order to let us call R functions from Java.

When the class is started, a new Rengine object is created. Then it loads the time series we want to use from the database, using the DBWrapper class. As stated in figure 3, this is the amount of positive tweets per day since the start of the database, a few of the earliest dates were removed since they had so few measured tweets. As you will find in the results section of this report this is not the best measurement to use for the data that we have collected. The reason for this is stated more clearly later, but in short the amount of tweets collected per day have increased during the project, due to the fact that I have been able to have the computer that was used for collection of tweets on for a longer amount of time. The implementation however, is designed to be kept running all day and night, and thus I have decided to not change the way the time series is created.

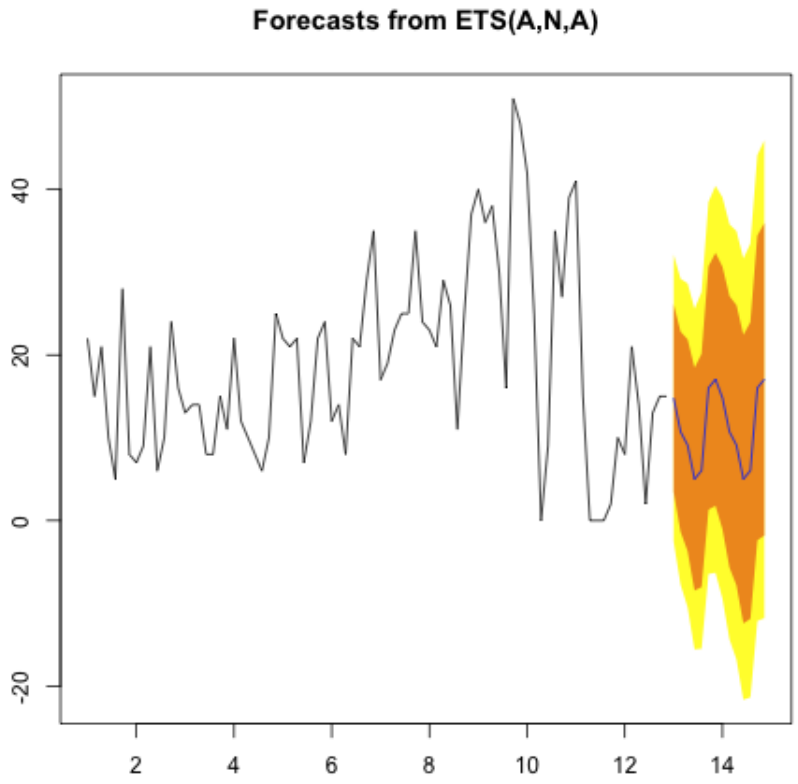

Figure 3: Output from the REvaluator class, an ETS-model fitted to the positive instances over time and the predictions made from this model. The x-axis is the time period, given in weeks, and the y-axis is the amount of positively classified tweets.

Once the time series have been fetched from the database it is then loaded into R. Here we also load the forecast library, which contains the functions we need to fit an ETS-model to the data.

We then create a time series object in R with a period of 7, which in this case should be understood as a weekly period. After this is done we create an ets-object, and use a forecast object to predict the values of the time series for every day, two periods, or weeks, in advance. The visualized ETS-model with the predictions is then saved to disc, which can be seen in figure 3. We then find todays date and see if we have collected more or less tweets than the upper limit would suggest we would find.

The visualized output from the forecast-object is interesting. Since the ETS-object is created automatically from the time series, the output when running the implementation might be different, since a different model might have been different. In the example shown in figure 3, the model that has been chosen is an ETS(A,N,A) model. The letters in the parenthesis are, in turn, the error, trend and seasonality. So in this example we have a model with the attributes additive error, no trend and an additive seasonality. The different attributes that a model can have is A for additive, N for none and M for multiplicative.

In figure 3 the predicted values are the values in the shaded areas. The dark values in the middle is the mean of the prediction, so this is the one the program uses as predicted value.

6 Time series

In order for us too be able to evaluate our results, as well as compare it to real world data it is necessary to view the collected data as a time series. When I in this section talk about a time series I refer to a series of numerical values that have the same difference in time between them. In the results section of this report (see section 7) I have only used two types of time series, the ones with daily values as well as others which have had weekly values.

This section aims to introduce a few concepts which are important to have a decent grasp over in order to understand what the results actually mean.

6.1 Pearson Correlation coefficient

The pearson correlation coefficient is a way of determining how strong the relationship between two variables are. Formally it is defined in 6. Given that we have two variables

x and y as well as their means $\overline{x}$ and $\overline{y}$ then the Pearson correlation coefficient is:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{6}$$

[4] The Pearson correlation coefficient, r, is always a number between -1 and +1, where +1 denotes a perfect positive linear relationship between the two variables and a value of -1 indicates a perfect negative linear relationship between the variables[4]. A value close to 0 means that there is no linear relationship between the two variables, however there might still be a nonlinear relationship between them[4].

6.2 Modeling the time series

In order for us to better understand the significance of the measured data during a day we need to be able to have a rough idea of how many positive tweets we expect to have on a given day. If we know how many positive tweets we expect on a given day we can see if we have gotten more or less than expected and we can use that data for a number of different applications. One useful application could for example be to alarm hospital staff if the number of positive tweets exceed some sort of threshold. This could indicate that there is an outbreak of a disease, especially if this is happening on a more local scale than I have investigated in this project.

Two ways of modeling time series that are common are, either using an Auto Regressive Integrated Moving Average (ARIMA) model or using an Exponential Smoothing model. Hyndman and Khandakar [19] suggests that exponential smoothing might be a better choice compared to ARIMA when we have a seasonal component. The R package forecast contains methods to create both types of models, to create an ARIMA model we can use the function arima(). If we don't know which parameters we want to have we can use auto.arima() to have this done automatically for us. An exponential smoothing method can be created using the ets() command. In this case we have a Twitter data, which show weekly seasonality[31] and thusly we use exponential smoothing to model our time series.

7 Results

7.1 Summary

In total 531,466 unique tweets where collected from 98 distinct days, days on which no tweets were collected are not part of this statistic. This means that on average 5423 tweets where collected per day. The maximum amount of Tweets collected during one day was 17,156 tweets collected the first of may 2012. The lowest amount of tweets was 1 tweet and that was collected the 14th of February 2012. Please note here that the reason that so few tweets were collected on this day is not because no one tweeted that day, instead it is because the actual collection of tweets started a few days later, and

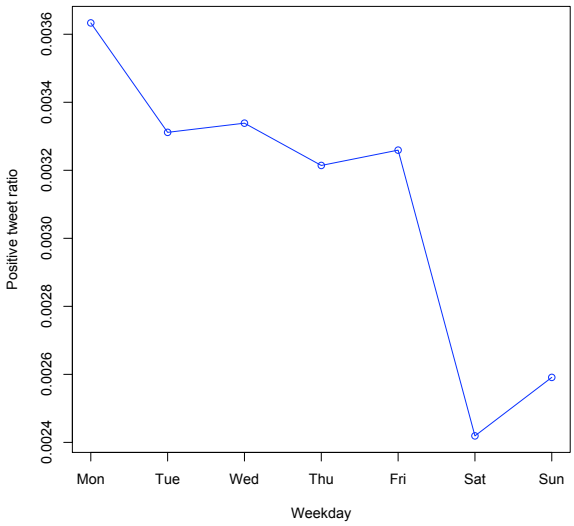

Figure 4: The ratio of positive tweets per weekday.

Region	Amount of Tweets	Percentage of Total Tweets	Positive Tweets	Percentage of Positive Tweets
South	301,461	57%	900	54%
Central	98,998	19%	349	21%
North	131,007	25%	414	25%

Table 10: Summary of the geographical distribution of Tweets during the course of the project.

some tweets from previous days remained accessible.

By applying our resulting classifier on all the instances that were collected during the collection process we plot this over each week day. Figure 4 might be more enlightening. This plot shows the ratio of positive tweets to negative tweets plotted over each week day. As we can see in the figure, the ratio of positive tweets to negative is much higher during Mondays compared to Saturdays. This could indicate that people are more sick early in the week compared to later in the week, or at least talk more about being sick during this time. However, there could also be a number of different explanations, for example the amount of tweets during the weekend might be much higher, thus generating more noise and pushing the ratio down.

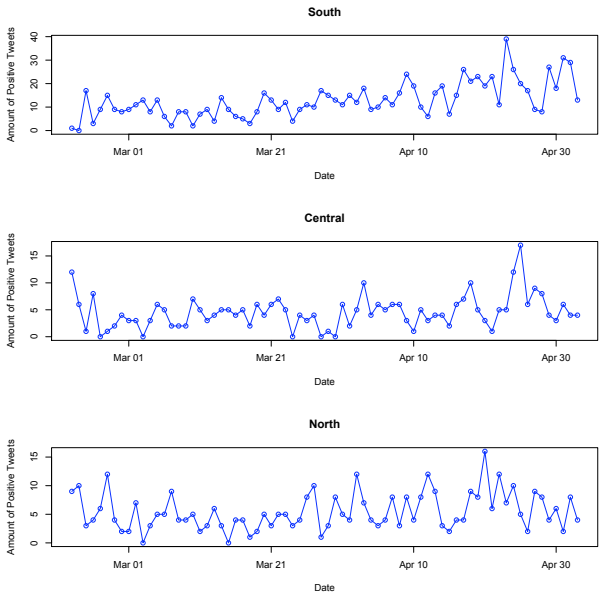

Figure 5: The amount of positive tweets, plotted per region and day.

	South	Central	North
South	X	0.09242403	0.08374806
Central	0.09242403	X	0.1748594
North	0.08374806	0.1748594	X

Table 11: The Pearson correlation coefficients between the different regions.

7.2 Regional

Overall most of the tweets have been observed in the area that is defined as the south of Sweden. The percentage of positive tweets is distributed roughly as expected, with each region having pretty much the same percentage of positive tweets as its amount of tweets.

The part of Sweden which is considered the south have substantially more tweets than any of the other regions. It has more than twice the amount of tweets that the north of Sweden have, and three times as many as the central part of Sweden. The main reason of this is that the population in the south circle is much higher than in the other circles.

Another contributing reason for this could be that the circles that define the three different regions are slightly overlapping, in order to make sure that the entire map is covered. The south region of Sweden is the one that gets queried first, and this would

lead to the overlapping areas between the south and central parts to be grouped to the south region. The same way, the overlapping areas of the central and north parts would be grouped with the central region. Of course, the difference in the active Twitter population would be the most likely reason for the difference in tweets.

I decided to look at the correlation between the different regions, to see if a lot of positive tweets in one region would indicate a lot of positive tweets in one of the other regions at the same day. Table 7.2 summarizes the results from the correlation tests. The Pearson correlation coefficient at a lag of 0 is shown, the correlation coefficient is not dependent on which order the time series are presented in and thus we get the same value for south and central as central and south.

As table 7.2 show, all of the correlation coefficients are positive, so it seems that there is no inverse relationship between them. However, none of the correlations are strong, so there is doubtful whether or not it exists any correlation between them, at least at the same day.

7.3 Connection with the Real World

The system which tries to predict the amount of Twitter messages that indicate sickness in Sweden during a given day using a statistical model is quite interesting in and off itself. However, if it's to have any impressive real world applications it's very important that the amount of positive tweets that have been measured actually correlate with how many people are sick. It is not impossible that the amount of tweets indicating sickness and the amount of people that actually are sick are unrelated.

Due to the fact that I haven't had access to statistics that show how many people actually are sick during a given day I have decided to compare my data to the data that was collected by Sjukvårdsupplysningen (the medical information service) which keep statistics on how many calls they have gotten during a certain day. The medical information service is different depending on which county (landsting) in Sweden you are in, and as such the data is different. Unfortunately the data does not exist for some counties and I have not been able to get access to data from others. Sweden consists of 20 different counties and in the end I was able to get access to the data from nine of these. The counties I was able to get data from was: Jämtland, Dalarna, Uppsala, Östergötland, Kronoberg, Västernorrland, Västmanland, Skåne and Örebro. I'd like to thank Pär Bjelkmar at Inera for helping me with the contact with the different counties as well as advice on how to do the statistical comparisons.

At the different medical information services statistics are divided into a number of subgroups depending on why someone decided to call them. In order to keep the data as comparable as possible I decided only to request the data that are similar to what I can detect with my text classifier. I ended up selecting 14 different types of reasons for calling the medical information service, and then summed all of the data over all the

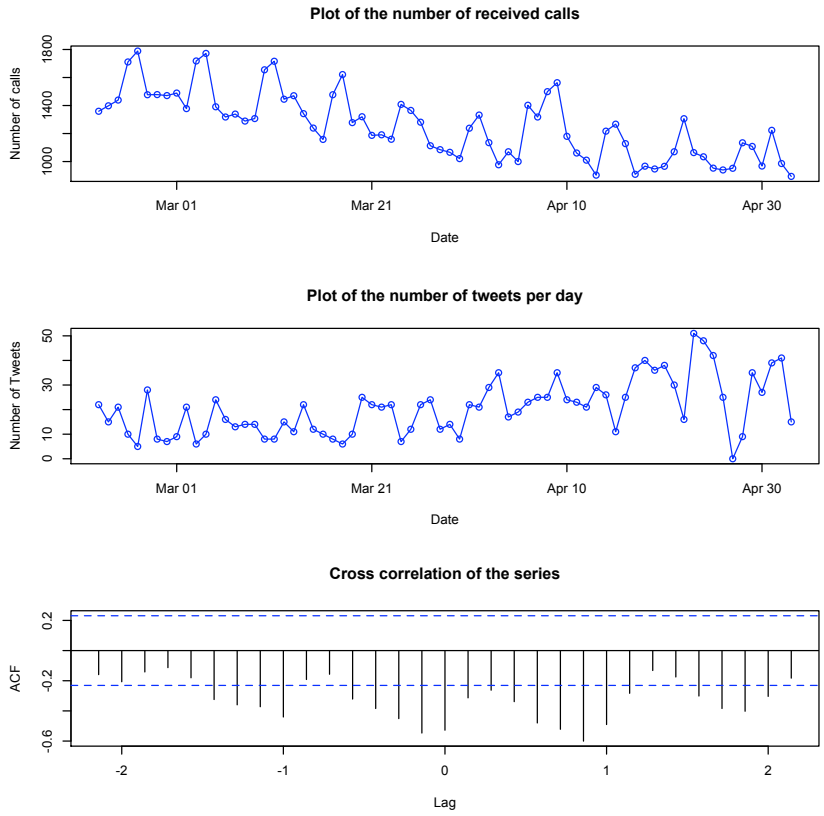

Figure 6: The amount of calls to the counties medical information services and the amount of positive tweets. The plot also contains the cross correlation function of the two series.

reasons for contact and counties grouped per day. The reason for doing this and not comparing the different sub groups is that my classifier only classifies tweets into two groups, and to compare sub groups I would have to rebuild my classifier, which was not feasible given the time limitations.

The data from the counties that I've had access to only goes from the 22nd of February 2012 until the third of May 2012. This means that in order to compare this time series to the ones that I have built from the Tweet database I have had cut off a few values from the start and end of the time series.

7.3.1 Raw Comparison

I started off by comparing the unaltered time series. As you can see in figure 6 the amount of calls to the medical information services went down as time went on. This is

Approach	Pearson correlation coefficient	P-value
No adjustment	-0.5279998	1.875e-06
Loess adjusted	-0.5801088	9.268e-08
Using ratio	0.1934317	0.1035
Using moving window	0.7848228	6.217e-15
Weekly sum	0.8707051	0.00225

Table 12: Summary of the different approaches as well as their Pearson correlation coefficients.

to be expected, since we are looking at time series that take place during the Swedish spring, which generally goes from very cold weather to warmer weather.

As figure 6 show, the amount of positive tweets on the other hand does not follow the same pattern as the amount of calls do. Instead we have that the amount of positive tweets seems to increase as time goes by. If we look at the Pearson correlation coefficient between the two series, we get a correlation of -0.5279998, with a p-value of $1.875e-06$, which is very low. We have a negative correlation here, which would indicate that a large amount of positive tweets during a given day would not indicate a lot of calls to the medical information services, instead a large amount of positive tweets would indicate that the medical information services would receive fewer calls than usual.

This result is quite unexpected and also counterintuitive. However, since we have a large discrepancy between the amount of time that was dedicated to collecting tweets during different days we have to adjust for that, which we try to do in the coming sections.

7.3.2 Missing Values

Owing to the fact that I've not had the possibility of running a computer 24 hours a day we have a few problems with the data. The main problem is that some of the data is missing or incomplete, for example, during a few days I have not been able to collect tweets at all, and some other days I have only been able to collect partial data. In order to get the best possible result we need to account for this. Like Bozdech et al.[1] I have chosen to start by using a locally weighted regression algorithm implemented in R via the function loess(), this local regression was first introduced by Cleveland[10] in 1979.

The standard settings have been used. The missing or incomplete values have then been replaced by the imputed values, unless the imputed value for a specific day is lower than the number of tweets that have already been collected. The motivation for this is that as we have already detected x number of tweets during the specific day, the actual number of tweets is at least x, and likely higher, but never lower. The amount of positive tweets plotted per day and the local regression plot that was fitted to that data can be

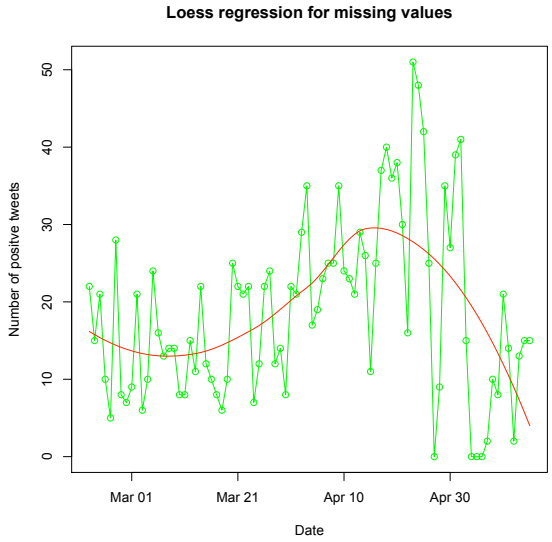

Figure 7: The amount of positive tweets per day and the local regression plot fitted to that data.

seen in figure 7.

Data for a day was marked as incomplete if it lacked any tweets from less than 16 unique hours. Since a day has 24 unique hours this roughly translates to us requiring tweets to be present for at least 2/3 of the day. If tweets can be found for more than 16 hours we consider the data to be complete and use the observed amount of tweets in out analysis. Otherwise we use the process with loess described in the previous paragraph to impute the amount of positive tweets for a given day.

Unfortunately this approach didn't improve the results, so it was quite quickly scratched. In fact, this approach gave an even stronger negative correlation, here we got a Pearson correlation coefficient of -0.5801088, with a p-value of $9.268e-08$. On the advise of Pär Bjelkmar I instead decided to use the ratio of positive tweets to the total number of tweets during a given day, or more formall in equation 7.

$$ratio = \frac{Positive\,Tweets}{Total\ Amount\ of\ Tweets + 1} \tag{7}$$

The reason for the added one in the denominator is that some days we haven't measured any tweets at all, giving us division by zero which of course is terrible for our resulting time series. The resulting time series can be seen in figure 8. As shown the resulting time series looks more believable, which is to be expected since we now adjust for the different amount of observed tweets.

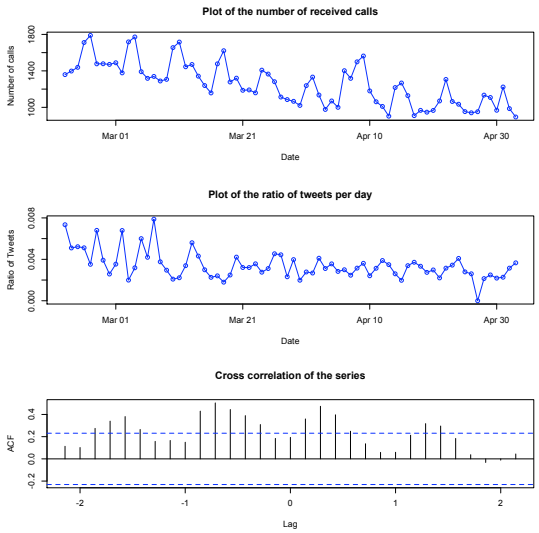

Figure 8: The amount of calls to the counties medical information services and the ratio of positive tweets. The plot also contains the cross correlation function of the two series.

The resulting correlation between the two series is also more like expected, instead of having a quite significant negative correlation we now have a small positive correlation of 0.1934317, with a p-value of $p = 0.1035$. It is still very possible that we get distorted data since we don't do anything to adjust for the weekdays, so that's what we will do in the next section.

7.3.3 Adjusting for weekly seasonality

As we showed in section 7.1 the ratio of positive tweets depends heavily on which day of the week it is. In order to adjust for this I decided, again on advise from Pär Bjelkmar, to apply a moving window filter on the data. This filter adjusts each datapoint so that it is the sum of itself and the previous six values, so that in each datapoint all the weekdays are represented. This approach should make the correlation between the time series independent of the underlying distribution over weekdays. This moving window filter is of course also applied to the data that we got from the county councils. The resulting time series as well as the cross correlation function between them can be seen in figure 9.

We can then look at the Pearson correlation coefficient between the two time series. When we do this we see that the correlation coefficient is 0.7848228 with a p-value of $6.217e - 15$. This is quite a high correlation, since the maximum of the Pearson correlation coefficient is 1.

Something that might be even more interesting is how the correlation looks if we only look at actual weeks. That is, we look at the correlation between the time series, but instead of doing as in the previous plot, look at the moving window, we look at actual weeks. To do this we unfortunately have to lose a few days worth of data in the beginning and the end of the time series, to adjust for the fact that the data we have is not all in entire weeks.

The procedure for doing this involves at first cutting off data at the start and end of the time series to make sure we only work with even weeks. Once this was done, I summed the ratio of positive tweets from the individual days of the week, and did the same thing with the total amount of calls to the medical information services. The resulting plots can be seen in figure 10. As shown, we end up having nine entirely complete weeks.

When we look at the Pearson correlation with this approach we end up with the highest correlation among all of the approaches. We get a positive correlation of 0.8707051, with a p-value of 0.00225. This is even higher than the one that we ended up with in the windowed filter approach. A correlation of 0.8707051 is indicative of a strong relation between the two time series, and figure 10 suggest that it is statistically significant, since the cross correlation function is above the blue line at lag 0.

7.4 Performance of the System

It is quite hard to evaluate the performance of the system. This is because the system currently uses the raw amount of tweets to make predictions on how many tweets are going to be measured in the near future. Even though the system haven't been up and running for long enough to make the predictions on unadjusted data reliable, I have decided to leave it this way anyway. Since the finished system is supposed to run 24 hours a day.

Not until the system have been up and running non stop for a while will it be possible to decide if this is the right approach to use, or if it would be better to use one of the adjusted approaches that was described in section 7.3.

8 Discussion

In the problem formulation and structure chapter of this project report I stated five different questions that I tried to answer during the course of this project. In this section I will discuss them and try to say if the problem was solved, if so to which degree, or if not why that was the case.

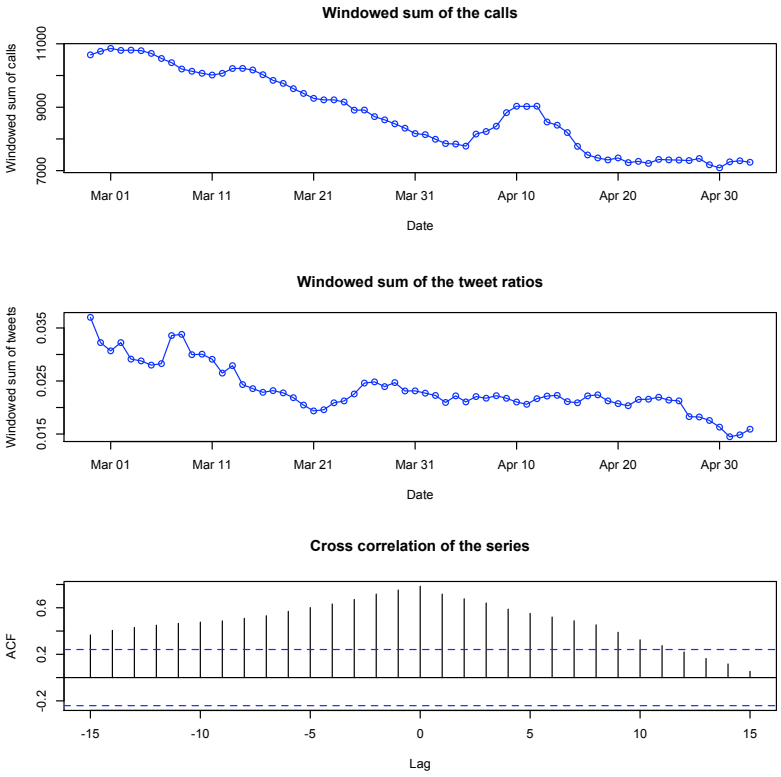

Figure 9: The amount of calls to the counties medical information services and the ratio of positive tweets, both run through a moving window filter which makes every datapoint the sum of itself and the previous six values. The plot also contains the cross correlation function of the two series.

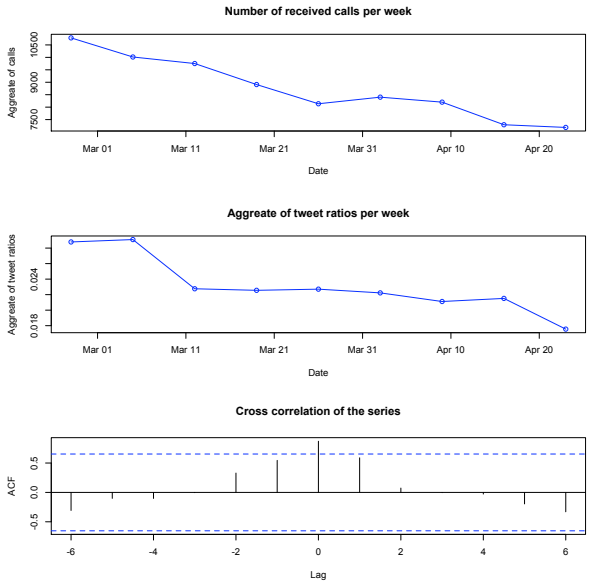

Figure 10: The sum of the total amount of calls to the medical information services as well as the sum of the tweet ratios per whole week. The plot also contains the cross correlation function of the two series.

8.1 Automatically Deciding which Tweets indicate Sickness

I decided to approach this with a machine learning approach. Instead of deciding myself on which key words were relevant I let an algorithm do this for me. This was done by manually marking a training set of tweets as either relevant or not relevant and letting the SVM train itself on this data. The resulting classifier performed very well on the training set. I would say that it also performed very well on new data. This is however something that is very hard to decide, since the data set is so large. This means that the only really feasible way do decide this in a one man project is to look at the tweets that was classified as positive and see if there seems to be a lot of misclassified tweets. This did not seem to be the case.

Even though the system seems to perform well I still did have some problems with misclassified tweets. Most of these are really hard to avoid and have to do with the ambiguity of the Swedish language. As with the English language, the word ”sick” (”sjuk” in Swedish) can be used to both indicate sickness, as well as to indicate that something is very good or bad. Consider for example the sentence ”That is so sick”. It is quite likely that this sentence would be classified as positive, simply because of the presence of the word sick, however it's clear to the reader that this sentence in fact does not indicate that anyone is sick.

Another problem that I encountered during this project was in contrast to most of the others highly specific to the Swedish language. The problem was that one Twitter user used the username "feber", which translates to the word "fever" in english. This meant that tweets which was not relevant to illness was classified as positive. After a quick investigation it turned out that there is a Swedish language website[1] which used Twitter to market their articles and keep in contact with their users. This problem was largely solved by removing usernames in the preprocessing stage, in order to make sure that the username did not make it to the weka classifier.

8.2 Real World Data Comparison

Even do the text classifier created in this project seem to do well, it's really hard to tell if it's actually useful without comparing the results to real world data. The main problem I had here was to get access to data detailing illness. At first I looked into getting data reporting the aggregate number of people being sick during a given day. However, I was unable to locate data that contained this. Instead I decided to use data from the Swedish medical information services.

When using this data, it's important to realize what we are correlating. We are not actually comparing our Twitter data to the amount of people that are sick. Instead we are correlating the data with the amount of calls to the medical information services. While it's very likely that those are correlated I don't have any actual evidence for this. This would be a very interesting test to do for anyone with access to the actual medical data.

8.3 Real World Data Comparison Performance

The Twitter data that was collected during the project seems to correlate very well with the data from the medical information services. This seems to be especially true when we look at the data that have been grouped weekly. When we look at the weekly correlation between the measured twitter data and the data from the medical information services we get a very high correlation (correlation of 0.87). This result is highly encouraging and suggests that Twitter can be used to predict the amount of calls to the medical information services with high accuracy.

8.4 How does this compare to previous research?

This is very hard to say, since as far as I can make out no previous research have been done that compare Twitter-data with data from the Swedish medical information services. This makes my results really hard to compare to other results, since we are essentially comparing different things. Most of the other research that use social media to detect diseases etc. are also very specific in what they are looking for. One of the things that are often looked for is influenza outbreak, and I have not been as specific in

[1] www.feber.se

my approach. Most research also deals with massive data set when compared to mine, for example Sakaki et al. used a data set of 96 million tweets[32], which is about 192 times as many tweets as I did. A lot of other research projects were also done over a lot more days, for example Lampos et al. used 50 million tweets over 303 days[25], whereas I have compared 72 days.

Even though we have these discrepancies we should be able to say if we have comparably good results or not. Lampos et al. tried to infer influenza rates from Twitter in England and Wales by using a bootstrapped version of LASSO. They found an average correlation of 91.11% with actual flu rates[25]. This result is as expected better than ours. On the other hand, Collier et al. found a Spearman's rho correlation of 0.68 when looking at influenza rates through avoidance behavior[11]. The result I have achieved in this project translates to a weekly Spearman's correlation of 0.85, which is higher.

It looks like our models performance is pretty good, performing at about the same level as an average influenza detector. It is however very important to note that the data I am comparing the performances with are not trying to investigate the same thing, or even using the same language, so the comparisons are not fair and they do not translate. They are only presented here to show that the correlation found in this project is comparable to what have been found in similar projects. By no means should this be taken as a comparison that can be used.

8.5 Can we use this for predictions

Whether or not these results can be used in predicting the number of people that will be sick is debatable. The high weekly correlation with the measured data from the medical information services are very encouraging in this respect. Unfortunately I have not had enough time in this project to conclusively show either way if prediction is possible.

The system currently uses a fitted ets-model to make predictions of the amount of positive tweets that we can expect on a given day. The performance of this model have not been evaluated in this project and thusly the predictive behavior remains unknown. My hope with the ets-model is that it will be able to predict the amount of tweets with a high accuracy, and since the correlation between the measured twitter data and the data from the medical information services is so high the predictions will also be accurate for that data.

The system currently fits the ets-model to daily time series data. The correlation with daily data is not very high at the moment. My hope is that this will change once the system has been able to run continuously for an extended period of time, so that daily predictions can be made. Before any decision on which data to use for the predictions are made it will be needed to be evaluated, and it's not unlikely that we have to use weekly data for the predictions.

In the end it's hard to say how good this system is at predicting the number of sickness related tweets, due to the fact that it currently uses the unadjusted amount of positive tweets to make this prediction. This means that the current measured data is not a good base to build the model on. To get an effective model using the unadjusted data we would have to run the system uninterrupted for a period of time and then evaluate the performance. The system can very easily be made to use one of the other adjusted models that was presented in section 7.3 simply by adjusting one sql-query in the REvaluator.java file.

As stated previously, I expect the unadjusted amount of tweets to do well as a predictor in a real world setting, and therefor have left it there. This assumption might prove to not hold true, especially since the moving window filter over a seven day period seemed to do very well when compared to the real world measured data. However, this is something that would need to be investigated further. Anyway, adjusting the ets-model in order to use a time series created by using a seven day moving window filter is quite simple, it only requires one extra line of R-code to be inserted:

```
SevenDayWindow = apply (embed (ts,7), 1, sum)
```

Since the system uses the automatic function ets() in the R programming language to build the model used for prediction, it ends up being very flexible. This is because the ets-model that is used will be generated again every time that the prediction function is called. In turn this means that the most current data is used to generate the model, and we don't rely on a model that was created using older data. Since the most current data is used to generate the ets-model, and this is done automatically by the ets() function, the model that is created will have the parameters that give the best model.

It would be desirable for our ets-model to use daily data to do the predictions, instead of using a weekly data. The reason for this is that the predictions will be done with the same frequency as the input data, so daily data will generate daily predictions, whereas weekly input data will generate weekly predictions. Since we have a high correlation at lag 0 for both the filtered data and the weekly data with the medical information services, a good prediction of the amount of positive tweets should also be a good prediction for the amount of calls to the medical information services.

9 Conclusions

In this project I have designed a system capable of the estimation of how many people will be talking on Twitter about being sick during a given day. Do to this several techniques from the fields of machine learning and statistics have been applied. The text classifier shows good performance on the testing data. The F-measure is 0.887 after the preprocessing steps, which includes removing usernames and retweets. Removal of the

usernames decreases the performance of the classifier on the testing data, however this is expected to not translate into new data since usernames are very unlikely to carry relevant data.

In order for the system to show any correlation with the data from the medical information services we have to adjust the values so that we look at the ratio between positive tweets and the total amount of tweets on any given day. If we do not do this the raw amount of positive tweets is too heavily influenced by how long the system was ran for that day, which makes the collected information unreliable.

Once the final data is adjusted for missing values it shows a high correlation with real world data collected by the Swedish county councils medical information services. This is particularly true when we compare data that have been weekly summarized. When comparing weekly summarized data collected from this projected with data from the Swedish county council medical information services we find a Pearson correlation of 0.8707051, with a p-value of $p = 0.00225$.

I have also shown that the system can show a good correlation of daily values, if the values are adjusted to be the sum of the daily value as well as the previous six daily values. This is done to cancel out any effects the day of the week have on the measured values. If this is done, we get a Pearson correlation of coefficient of 0.7848228 with a p-value of $p = 6.217e - 15$.

The performance of the predictions made by the final system have not been investigated. The model that is used to make the predictions is an ETS-model, created in R-code. Hopefully the performance of this predictor is good, but as stated it has not been fully investigated.

9.1 Future Work

While the final system in this project show significant promise, with a high correlation with real world data, there is still things left to investigate and improve. This section aims to describe future work that could be done.

9.1.1 Running the system for an extended amount of time

During this project I was only able to run the final system for a few months. It would be interesting to see how the results would look if it instead were allowed to operate for a longer time, preferably 24 hours a day. This should also make the correlation with real world data more interesting when comparing the raw data, instead of the adjusted. It would also be interesting to see what would happen if the system was running during a health crises.

9.1.2 Evaluating the performance of the predictor

As stated earlier, the system includes a function to predict how many positive tweets will be collected during the next time period. I haven't evaluated the performance of this, and it would be interesting to know how well it does.

9.1.3 Improving the classifier

The text classifier performs very well, nevertheless it should be possible to improve it. It might be that the choice of classifier is not optimal and that some other classifier has a higher performance. It might be that the current classifier is "good enough" and that even if it can be improved it might not improve the final results notably.

9.2 Final words

As concluding remarks I'd like to summarize the results of the project. I have shown that the amount of tweets about sickness seems to be a good indicator of how many calls the medical information services will receive. This is especially true if we adjust the data to remove the weekly seasonal effects. The performance of the text classifier is very good on the training data, and since the results correlate very well with the data from the medical information services, it seems likely that the text classifier generalize to new data very well. The only thing I wish was different is that I would have liked to be able to run the software on a dedicated computer in order to accurately be able to evaluate the predictive performance of the ets-model.

References

[1] The transcriptome of the intraerythrocytic developmental cycle of plasmodium falciparum. *PLoS Biol*, 1(1):e5, 08 2003.

[2] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[3] The Weka api on StringToWordVector last accessed on 2012-07-10. http://weka.sourceforge.net/doc.dev/weka/filters/unsupervised/attribute/stringtowordvector.html.

[4] Viv Bewick, Liz Cheek, and Jonathan Ball. Statistics review 7: Correlation and regression. *Critical Care*, 7:1–9, 2003. 10.1186/cc2401.

[5] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152. ACM, 1992.

[6] D. Boyd, S. Golder, and G. Lotan. Tweet, tweet, retweet: Conversational aspects of retweeting on twitter. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, 2010.

[7] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[8] Johan Carlberger, Hercules Dalianis, Martin Hassel, and Ola Knutsson. Improving precision in information retrieval for swedish using stemming, 2001.

[9] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[10] William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836, December 1979.

[11] Nigel Collier, Nguyen Son, and Ngoc Nguyen. Omg u got flu? analysis of shared health messages for bio-surveillance. *Journal of Biomedical Semantics*, 2:1–10, 2011.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018.

[13] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327, September 1995.

[14] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, 1998.

[15] George Forman and Ira Cohen. Learning from little: Comparison of classifiers given little training. In *Knowledge Discovery in Databases: PKDD 2004*, volume 3202 of *Lecture Notes in Computer Science*, pages 161–172. Springer Berlin / Heidelberg, 2004.

[16] Eibe Frank and Remco R. Bouckaert. Naive bayes for text classification with unbalanced classes. In *Proceedings of the 10th European conference on Principle and Practice of Knowledge Discovery in Databases*, PKDD'06, pages 503–510, 2006.

[17] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[18] Chang C. C. Lin C. J. Hsu, C. W. A practical guide to support vector classification. 2003.

[19] Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 27(i03), 2008.

[20] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[21] Igor Kononenko and Ivan Bratko. Information-based evaluation criterion for classifier's performance. *Machine Learning*, 6:67–80, 1991. 10.1007/BF00153760.

[22] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, 2007.

[23] Sotiris Kotsiantis, Dimitris Kanellopoulos, and Panayiotis Pintelas. Handling imbalanced datasets: A review. 2008.

[24] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. 1997.

[25] Vasileios Lampos and Nello Cristianini. Nowcasting events from the social web with statistical learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3, 2011.

[26] Vasileios Lampos, Tijl De Bie, and Nello Cristianini. Flu detector - tracking epidemics on twitter. 6323:599–602, 2010.

[27] John Makhoul, Francis Kubala, Richard Schwartz, and Ralph Weischedel. Performance measures for information extraction. In *In Proceedings of DARPA Broadcast News Workshop*, pages 249–252, 1999.

[28] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. Introduction to information retrieval. In *Online Edition*, 2009.

[29] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[30] C Pelat, C Turbelin, A Bar-Hen, A Flahault, and A Valleron. More diseases tracked by using google trends. *Emerging infectious diseases*, 15(8), 2009.

[31] Eduardo J. Ruiz, Vagelis Hristidis, Carlos Castillo, Aristides Gionis, and Alejandro Jaimes. Correlating financial time series with micro-blogging activity. In *Proceedings of the fifth ACM international conference on Web search and data mining*, 2012.

[32] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 851–860, 2010.

[33] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002.

[34] Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 614–622, 2008.

[35] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. June 2011.

[36] C. Silva and B. Ribeiro. The importance of stop word removal on recall values in text categorization. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pages 1661 – 1666 vol.3, july 2003.

[37] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[38] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[39] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3), March 1995.

[40] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, San Francisco, CA, 2nd edition, 2005.