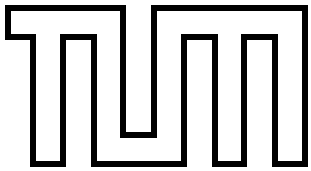


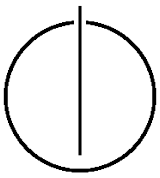
FAKULTÄT FÜR INFORMATIK

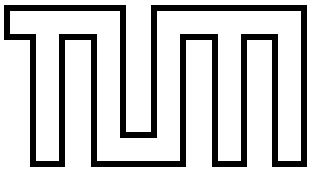
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Utilizing Crowd Intelligence for Online Detection of Emotional Distress

Siddhant Goel





FAKULTÄT FÜR INFORMATIK

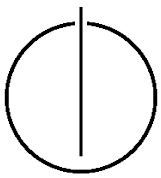
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Utilizing Crowd Intelligence for Online Detection of Emotional Distress

Insert thesis title in German here

Author:	Siddhant Goel
Supervisor:	Prof. Dr. Claudia Eckert
Advisor:	Han Xiao, M.Sc.
Date:	March 15, 2013



Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this master's thesis only supported by declared resources.

München, den March 15, 2013 Siddhant Goel

Acknowledgments

If someone contributed to the thesis... might be good to thank them here.

Abstract

An abstracts abstracts the thesis!

Contents

Outline of the Thesis

Part I: Introduction

CHAPTER 1: INTRODUCTION This chapter presents an overview of the thesis and it purpose. Furthermore, it will discuss the sense of life in a very general approach.

CHAPTER 2: RELATED WORK Related Work

CHAPTER 3: PROBLEM DEFINITION Problem Definition

Part II: Theoretical Background

CHAPTER 1: CLASSIFICATION METHODS

CHAPTER 2: TEXT REPRESENTATION

CHAPTER 3: ENSEMBLE LEARNING

Part III: Experiments

CHAPTER 1: EXPERIMENTS

CHAPTER 2: APPLICATION

CHAPTER 3: RESULTS

Part IV: Conclusion

CHAPTER 1: CONCLUSION

Part I.

Introduction

1. Introduction

We start the thesis with an introduction. Do not spend time on formating your thesis, but on its content.

1.1. Latex Introduction

There is no need for a latex introduction since there is plenty of literature out there.

2. Related Work

Related Work goes here

3. Problem Definition

Define the problem here

Part II.

Methodology

4. Classification Methods

4.1. Introduction

Machine Learning is a branch of computer science that deals with building and analyzing systems that are able to learn from data. Algorithms based upon such analysis involve constructing a model from a given dataset, and then using this model to perform the required tasks. Machine Learning techniques can broadly be divided into two categories - supervised learning, and unsupervised learning.

- Supervised Learning
  Methods falling in this category operate in two phases -
  - In the first step, we assume the availability of a training data, which is used to build the model so that it takes into account the structure of the given dataset
  - In the second step, we use this model to make predictions on the testing data (the real world data). This is the data that the model has not seen yet, and is required to make predictions on.

  Our primary focus in this thesis remains on supervised learning methods.

- Unsupervised Learning
  Methods falling under this category operate in a single phase - the model starts with zero knowledge about the structure of the given dataset. As we feed data into the model, it continuously learns the structure of the given dataset and calculates the predictions based on this knowledge. The main difference between this family of algorithms and supervised learning algorithms is the presence/absence of training data.

Classification is one of the fundamental problems in machine learning. Given a dataset D, we are required to separate the samples contained within the dataset into two (or more than two, depending on the input) classes. Formally, given a dataset in which each instance is of the form $[(d_1, d_2, ...d_n), l_j]$, where each d_i is the feature value of feature $k \in [1, n]$, and l_j is the label of the sample which can take a limited number of possible values, the aim is to calculate the value of l_j, given the feature information. This separation can usually be done using a supervised learning method, in which case we're given the training data (on which the model is built) and are required to predict the labels of the testing data, or using unsupervised methods, where the model is required to identify the categories of the samples without any external help.

The performance of a particular classifier depends on a number of factors, one of the major ones being the type of the data to be classified. Not all classifiers are good for all classes of problems. Some classifiers suit a particular problem more than some others; choosing a classifier for a problem still remains a decision which may or may not be completely

scientific, even though there have been a number of tests been done to correlate classifier performance with data type.

4.2. Support Vector Machines

Support Vector Machines (SVM) form a fairly popular class of machine learning algorithms used mainly for binary classification and regression analysis, making an assumption that the input (training) data is in the vector-space format. Given the training data, the goal of an SVM is to find a decision boundary (a hyperplane, or a set of hyperplanes in a high or infinite dimensional space) that separates the two classes of data while maximizing the distance of the boundary from any data point. The decision function that results is fully specified by a (usually small) subset of the data, and the points in this subset are referred to as *support vectors*.

All classifiers resort to a *distance function*, in some form or the other, that can provide a similarity value between two points. In the simplest form of an SVM, the distance function is simply the dot product between the two points, and such SVMs are referred to as *Linear Support Vector Machines*. Often, the case is that a simple linear SVM is not able to find a sufficiently accurate decision boundary that can separate the data points into two classes, simply because the input data is not linearly separable. In such cases, we use the *kernel trick*, transforming the input data into a much higher dimensional input space using a *kernel function* K, which makes separation in that space easier. The main property of such kernel functions that is exploited is the fact that even though calculating $K(x)$ may be expensive, calculating $K(x, y)$ (which is the dot product of vectors x and y in the higher dimensional space, and hence a similarity measure between the two points in that space) is much cheaper in cost.

The most popular kernel functions include -

- Linear (the simple SVM) - $K(x, y) = (x.y)$
- Polynomial - $K(x, y) = (\gamma * x.y + c)^d$
- Radial Basis - $K(x, y) = \exp(-\gamma * |x - y|^2)$
- Sigmoid - $K(x, y) = \tanh(\gamma * x.y + c)$

4.3. Multiple Kernel Learning

Multiple Kernel Learning makes an improvement to standalone Support Vector Machines by using multiple kernels instead of a single one. In contrast to normal SVMs which use a single kernel function to calculate the similarity score between two data points, MKL algorithms combine scores from multiple kernels to obtain one single score, which is then used as the final similarity score. As summarized by [1], multiple kernel learning algorithms can be put into 12 major categories, based on some of their key properties (learning method, functional form, target function, training method, base learner, and the computational complexity).

[1] also mentions the two main uses of MKL algorithms - the first one uses the fact that since different kernels may be used as different notions of similarity, we may maximize the accuracy obtained from each kernel, and then let the learning algorithm decide whether one kernel (which works better than the rest) or a combination of kernels is suitable for the task at hand. The second use is a more traditional use, where different kernels corresponding to different notions of similarity are combined, either in a linear or in a non-linear fashion to obtain a single kernel. As with all machine learning algorithms, multiple kernel learning does not suit all possible use cases, but in a lot of cases, combining multiple kernels may result in increased accuracy.

5. Text Representation

5.1. Introduction

Machine Learning algorithms are designed to operate on numbers. While classifying documents, a prerequisite is to convert the text in the documents to a format which such an algorithm can understand and work on. One of such format is the vector space model, which we earlier mentioned. Such processes include obtaining text from documents, building word vocabularies, text preprocessing (such as tokenization, stemming, stop words removal), and converting terms to actual usable feature values. Since our work takes into account only text in English, we do not lose our focus by digressing on to issues that arise due to other languages. We explore all such issues in this chapter.

5.2. Preprocessing

Preprocessing documents typically involves all the steps that are needed to be performed before proceeding on to extracting meaningful information from text. Text documents in their original forms normally contain a lot of extra information that usually does not affect how a classifier behaves. Preprocessing a document removes all the noise, and brings a document in a state where it can be used for scoring. The following are the various filters that are applied -

- Tokenization
  Tokenization is the process of splitting a sequence of words into distinct pieces of alphanumeric characters, called tokens. Extra characters which are not needed, such as punctuations, are removed. For instance, tokenizing the text *"The quick, brown, fox jumps over the lazy dog;"* results in the following list of tokens -
  the, quick, brown, fox, jumps, over, the, lazy, dog
  In most cases, tokens are split using whitespcaces, line breaks, or punctuation characters. Splitting on white spaces may also result in loss of information. For instance, if the string *San Francisco* appears in the text, then the tokens extracted will be *San* and *Francisco*, whereas the correct tokenization should treat both the terms in one token. Such problems are solved using n-grams, which are explained later in this section.

- Stop Words Removal
  Some of the words in the English language such as *and*, *the*, and *a*, besides some others appear in almost every text. These words add very little meaning to the text on the whole, and their frequency of occurrence is very high. Correspondingly, since the signal that these words add is very low, besides contributing incorrectly to document similarity (since these documents appear a lot in almost every document, this

may lead to high, but inaccurate similarity scores between two documents), the best option is to remove such words from the text.

- Stemming
  Stemming is the process of reducing words to their root form, usually by stripping some characters from the word endings. A strict requirement for stemming is that related words must stem to the same final word. For instance, if we have the words *car*, *cars*, *car's* and *car's*, they all must stem to the same word *car*. This helps in removing unnecessary information from the text which would otherwise inflate the vocabulary with low-signal information.

5.3. Representation and Vector Space Classification

After a document has been preprocessed, we need to represent it in terms of the content that it has, such that the relative importance of each word has been taken into account. Since a document is just a collection of terms, we represent a document d_i as a vector $d_i = (f_{i,1}, f_{i,2}, ..., f_{i,m})$, where each dimension $f_{i,j}$ corresponds to a term, and the value depends on its occurrence, either only in the document, or in the document as well as in the entire corpus. This approach is called the bag of words model.

For instance, if there are two simple text documents -

We are headquartered in Munich, Germany
We also have an office in Berlin, Germany

Based on these two documents, the term dictionary is built like the following -

"We": 1, "are": 2, "headquartered": 3, "in": 4, "Munich": 5, "Germany": 6, "also": 7, "have": 8, "an": 9, "office": 10, "Berlin": 11

and the documents are represented as the following vectors -

1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1

where each value represents the number of times the corresponding feature appeared in the document. The order of words does not matter; the features simply indicate whether or not the word appears in the document.

In this approach, the values contain term frequencies. A slight improvement is to divide the term weights with the total number of terms in the document, which assigns the feature values while also taking into account the relative importance of the term in the document. But an even better improvement is to use the *tf-idf* value, which (for a term in the document) is defined as -

$$tfidf_{t,d} = tf_{t,d} * idf_t$$

where $tf_{t,d}$ is the term frequency of the term t in document d, and idf_t is the inverse document frequency of the term t across the entire collection of documents, usually defined as $log(N/df_t)$, where N is the total number of documents, and df_t is the number

of times this term appears in the entire collection of documents. The main idea here is to reduce the term frequency weight of a term by a factor that increases proportional to its frequency in the corpus. This score is the highest when the term occurs many times within a small number of documents, and the lowest when the term occurs in almost all documents, hence providing a good representation of a document in the vector space format. This representation is now used to calculate the similarity between two documents, the standard way for which is to compute the vector dot product between the two vector representations -

$$sim(x, y) = \frac{(\vec{x}.\vec{y})}{|\vec{x}||\vec{y}}$$

Representing documents as vectors and calculating similarities based on their vector products leads to a view of a corpus as a *term-document* matrix, the rows for which represent the documents, and the columns for which represent the terms present in the corpus.

6. Ensemble Learning

Ensemble Learning is a class of supervised machine learning algorithms that use multiple models instead of a single one to obtain better predictions. A common problem with classifiers is that not all classifiers are suited for all kinds of problems. Ensembles combine multiple classifiers to construct a (hopefully) better classifier. The performance of an ensemble is not guaranteed to be better than the individual classifiers, but depending on the problem, ensembles usually outperform their constituent classifiers by a fair margin. Using ensemble learning implies performing more computation, but ensemble learning provides better results when the classifiers are not similar to each other. In this chapter, we explore the main ensemble learning techniques that we employ in our work.

6.1. Bagging

Bagging, also referred to as Bootstrap Aggregation is one of the most basic forms of ensemble methods. Given a training dataset D of n samples, each sample having f different features, the aim of bagging is to combine a certain number of classifiers, each trained using a (mostly different) subset of the main data, such that every classifier learns about a different structure of the input data, and when the outputs from all such classifiers is combined, the final output is expected to be better than the individual predictions. Bagging is known to improve the performance when the classifier is slightly better than a random classifier. On the other hand, in cases where the constituent models are already strong, the performance is known to often degrade.

To train different classifier on a different subset of the data, the training data is either split on the number of samples, or on the number of features. In the first case, we pick $n'(< n)$ samples (with replacement) and each time train a new classifier using the new dataset. Since the data points are picked with replacement, some of the data points may be common to some classifiers. In the second case, we train every new classifier using $m'(< m)$ features (with replacement). Whenever the ensemble is required to make a prediction on a new sample, a majority vote from all the constituent classifiers is taken, which becomes the final output of the ensemble. Bagging is known to reduce variance and avoid overfitting.

6.2. Boosting

Boosting aims to build an ensemble by iteratively training weak classifiers on a distribution, and then finally putting them all together to build a strong classifier. In the training phase, the classifiers are trained one by one. Each training sample is assigned a numeric weight, which is the same for all samples in the beginning. After the first classifier has been trained, the weights for the points which were misclassified are increased, and the

resulting input data (along with the modified weight values) is fed to the second classifier. This process continues until the last classifier has been trained. Each classifier has an incentive to focus a little more on classifying correctly those points which the previous classifier classified incorrectly.

At all times, each classifier's performance is maintained in a vector (usually named α). Whenever a new point has to be classified, the individual predictions are obtained from the constituent classifiers, and a weighted sum (from the α values) is taken across all the classifiers to obtain the final prediction. One of the very popular Boosting techniques is AdaBoost. The main aim of Boosting is to build a strong classifier from a set of weak classifiers. To that effect, even classifiers only slightly better than random are considered useful, because in the final predictions, they will still contribute positively to the aggregate prediction by behaving like their inverses because of having negative coefficients in the final linear combination of classifiers.

6.3. Stacking

Stacking combines classifiers not by analyzing their performances on the training data, but by treating the individual predictions (on the training data) as a second level of training data, and then using this piece of data to build a final model that is ultimately used for making the final prediction. Different from the previous ensemble learning methods, stacking operates by feeding the initial training data to the set of base classifiers chosen. After these base classifiers have been well calibrated, the predictions on the training data are obtained from the base classifiers, forming a new dataset. This dataset is now fed into a second level classifier, which is responsible for making the final prediction. Often, the base classifiers (at the first level) are trained using different features of the input data to increase classifier diversity.

For making a prediction on a new point, predictions are made using all the base classifiers, forming an equivalent data point in which the number of features is the same as the number of base classifiers. This point is then run through the second level classifier to obtain the final prediction. In contrast to other ensemble learning methods, no voting or aggregation takes place between classifiers. This class of methods has proven to be very successful, one recent example being that of the winning team in the Netflix prize competition.

Part III.

Experimental Results

7. Experiments

7.1. Dataset

Our work focuses on detecting emotional distress from publicly available tweets/blogs. To predict the label for a particular tweet, we need a similar dataset that contains similar sized text, each having a label corresponding to whether or not the text indicates depression. This dataset comprises the training dataset.

Non availability of such data in the beginning of our work led us to explore a slightly different problem in the same domain that operates on a similar dataset. We initially performed our experiments on a dataset made available by a machine learning competition website [2], where the aim is to predict whether a certain piece of text (a comment from a conversation on the internet) can be insulting to a user or not. The dataset provided contained the list of comments, each with a binary label. The dataset was split into two parts, the first file containing 3947 comments, while the second one containing 2235 comments.

After conducting our experiments on this dataset, we start building the dataset which can finally help us towards building a system that can identify emotional distress from the given text. We download (on-demand) stories from the website Reddit [3], which is an online community for people to interact with one another, hosting two main subreddits of our interest - the subreddit where people post if in case they are planning to end their lives ([5]), and the subreddit where people post if in case they want to share their happy moments with others ([4]). We integrate the process of building the dataset on our main web interface, and every time there's a request to increase the size of the dataset, we download 500 stories from each of the subreddits mentioned and store them in our database. Since our system also aims to incorporate crowd intelligence into the system (letting people assign labels to the training data), we initially label 2000 stories manually to build a strong foundation for the system, and then label the stories on demand.

To actually incorporate identifying emotional distress into our system, we fetch data from Twitter. We use Twitter's public streaming API [6] to fetch 100 tweets every 3 hours, hence fetching 800 tweets every single day. This gives us an overall view of the general sentiment of the public, on which we can perform our analysis.

7.2. Approach and Setup

We experiment and evaluate with various machine learning techniques, including standalone support vector machines, multiple kernel learning algorithms, and finally ensemble learning methods. Our work can broadly be seen as being done in two phases - in the first phase, we evaluate (using cross validation) the machine learning techniques mentioned, and in the second phase, we build a system that monitors emotional distress on the internet.

In the first phase, we use the comments [2] dataset, construct n-grams (of length 2) and then obtain the vector space representation for each comment, along with the label for each. This data is now split into training and testing data according to holdout cross validation. The model is then trained on the training data, and the predictions are obtained for the testing data. Since we already have the actual predictions of the testing data, we calculate the accuracy of the classifier thus trained. We perform a 80-20 split on the data, that is, we use 80% of the data to train the models, and 20% of the data to calculate predictions on.

This procedure is repeated for standalone support vector machines (using linear, polynomial, gaussian, and RBF kernels), multiple kernel learning methods, and the ensemble learning methods which include bagging, boosting, and stacking. The calculated accuracy for each method is then reported.

In the second phase, we implement our system that is able to monitor public content (we concentrate on fetching content from Twitter) and identify the tweets which may signal emotional distress.

7.3. System Details

8. Results

Results

Part IV.

Conclusion

9. Conclusion

Conclude

Appendix

A. Appendix

Appendix

Bibliography

[1] Mehmet Gönen and Ethem Alpaydin. Localized algorithms for multiple kernel learning. *Pattern Recognition*, 46(3):795–807, 2013.

[2] Kaggle. Detecting Insults in Social Commentary. http://www.kaggle.com/c/detecting-insults-in-social-commentary.

[3] Reddit. reddit: the front page of the internet. http://www.reddit.com.

[4] Reddit. /r/happy. http://www.reddit.com/r/happy.

[5] Reddit. /r/suicidewatch. http://www.reddit.com/r/suicidewatch.

[6] Twitter. Twitter Streaming API. https://dev.twitter.com/docs/streaming-apis/streams/public.