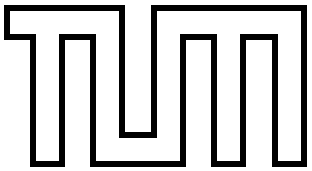


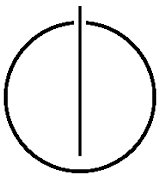
FAKULTÄT FÜR INFORMATIK

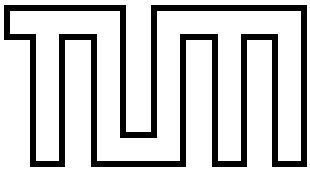
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Utilizing Crowd Intelligence for Online Detection of Emotional Distress

Siddhant Goel





FAKULTÄT FÜR INFORMATIK

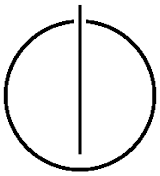
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Utilizing Crowd Intelligence for Online Detection of Emotional Distress

Die Verwendung von Crowd Intelligence zur unmittelbaren Erkennung von emotionalem Stress

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

According to the World Health Organization, nearly one million people die from suicide every year. This means one death almost every 40 seconds. The maximum number of people committing suicide are between 15 and 29 years old, which comprises the young section of the society. The recent success of social networking websites like Twitter, Facebook, Reddit, and Wordpress has shown that more and more young people these days tend to express their inner feelings on the web, irrespective of whether those feelings are positive or negative. In order to bridge the disconnect between these two pieces of information, we present a surveillance and monitoring system of suicide.

The system blends supervised machine learning techniques and semi-online learning to make predictions about a general level of distress (in %) amongst people posting on Twitter, and about particular instances of tweets which may lead one to believe that the author posting that content is depressed and may need further attention. The system also taps into crowd intelligence to learn about which pieces of text are depressed and which are not. In addition, we also present an evaluation of support vector machines and ensemble learning methods in the domains of text classification and sentiment analysis. The goal of the final system is to provide timely intervention and promote better health.

Contents

Acknowledgements **vii**

Abstract **ix**

Outline of the Thesis **xiii**

I. Introduction **1**

1. Introduction **3**
1.1. Problem Definition 3

2. Related Work **7**

II. Theory **9**

3. Machine Learning and Classification Methods **11**
3.1. Machine Learning 11
3.2. Classification Methods 11
3.3. Support Vector Machines 12

4. Text Representation **19**
4.1. Preprocessing 19
4.2. Representation and Vector Space Classification 20

5. Ensemble Learning **23**
5.1. Bagging 23
5.2. Boosting 24
5.3. Stacking 25

III. Experimental Results **27**

6. Experiments **29**
6.1. Dataset 29
6.2. Approach and Setup 30

7. Results **31**
7.1. Evaluation 31
7.1.1. Support Vector Machine 31
7.1.2. Ensemble Learning 32
7.2. System Details 36
7.2.1. Ratings 37
7.2.2. Monitoring 37

IV. Conclusion and Future Work **41**

8. Conclusion and Future Work **43**

Appendix **47**

A. Appendix **47**

Bibliography **49**

Outline of the Thesis

Part I: Introduction

CHAPTER 1: INTRODUCTION This chapter presents an overview of the thesis, and defines the problem which it aims to solve.

CHAPTER 2: RELATED WORK Related Work

Part II: Theoretical Background

CHAPTER 1: CLASSIFICATION METHODS

CHAPTER 2: TEXT REPRESENTATION

CHAPTER 3: ENSEMBLE LEARNING

Part III: Experiments

CHAPTER 1: EXPERIMENTS

CHAPTER 2: RESULTS

Part IV: Conclusion

CHAPTER 1: CONCLUSION

Part I.

Introduction

Chapter 1

Introduction

The Internet today is a vast source of information curated by real people. In recent times, the popularity of websites like Twitter, Reddit, and Wordpress has shown that every day, more and more people are getting comfortable with expressing their inner feelings on the web, irrespective of whether they feel good or bad. Negative feelings expressed this way provide an important indicator of what might be going wrong in their lives, or what may ultimately lead up to something more tragic or even terminal. In some cases, factors leading to suicide can be identified early on by looking at the physical and verbal behavior of the person under consideration. Public availability of a larger set of data that may contain content posted by emotionally distressed people, and non-availability of a system that can analyze such data to find such people is a clear disconnect. This thesis attempts to bridge this gap by applying machine learning techniques to build a system that is able to find instances of emotionally distressed content on the Internet. This can then serve as a first step towards providing further help to such people, possibly in the form of human intervention.

1.1. Problem Definition

A large portion of the nearly one million people who die every year from suicide includes young people. In recent times, these people have started to express their inner feelings on the web, on websites like Twitter, Wordpress, Reddit (Figure 1.1 and Figure 1.2), and many others. Suicide is a medical condition that has the potential to be detected early on, by observing the physical and verbal behavior of the person under concern. The work presented in this thesis exploits these two facts, and attempts to build a system that can monitor the public feed of Internet websites such as Twitter (on which people post about what they feel) and detect the content that may have been posted by a person under emotional distress.

Pointing out the exact phrases which lead one to believe that a person may be under some degree of emotional distress is a problem best handled by psychologists and linguists. Even though prediction with machine learning algorithms is less sophisticated than human classification, it is known (and also presented in this thesis) that such tech-

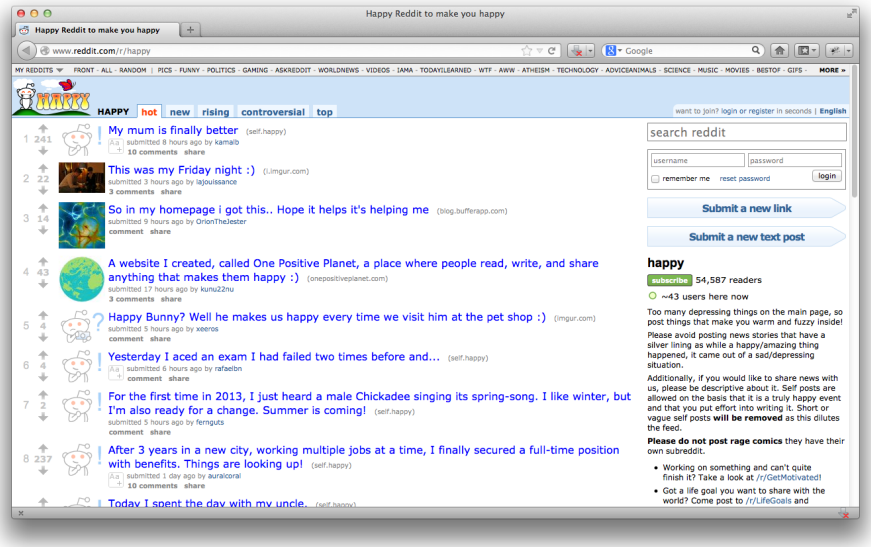


Figure 1.1.: "/r/happy", the section of the website Reddit [23] where people post content when they are happy and want to share it with everyone else.

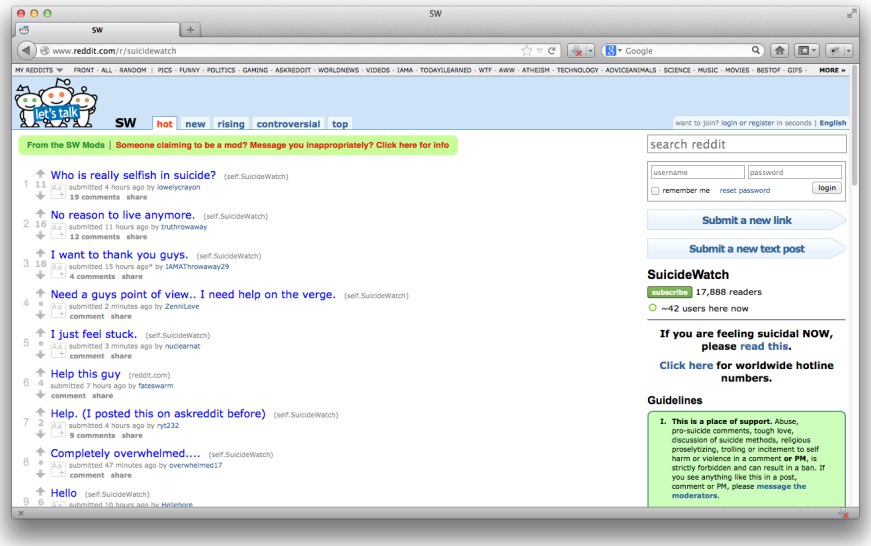


Figure 1.2.: "/r/suicidewatch", the section of Reddit [23] where people post when they are depressed, or are on the verge of taking their own lives.

niques can identify and categorize the sentiment of a piece of text to a reasonable accuracy. A person may be under emotional distress if he/she posts content which is indicative of sentiments and/or actions like depression, suicide, loneliness, and helplessness. During the work performed in this thesis, the following two categories of phrases were found indicative of whether or not a person needs help.

- **Direct** Phrases such as *"thoughts of suicide make me happy"*, *"I have a rope around my neck"* or *"my suicide note"* indicate in a very straightforward manner that the author is not only depressed, but is also on the path of taking his/her own life. Such phrases are very direct indicators of the emotional condition of the person.
- **Indirect** Phrases such as *"I don't know anything anymore"* or *"Need someone to talk to"* or *"Please help"* indicate that the person who is posting such content does not necessarily want to kill themselves yet, but they are not in a very good emotional condition either. Such words indicate that this person is under depression, or is feeling lonely. Even though they may not be suicidal yet, such a dramatic and tragic event might be the next step in their lives.

Both the category of phrases indicate that the emotional health of the person under consideration is not normal, and may need attention.

The work presented in this thesis tackles this problem by following a two-step approach. In the first step, various text classification algorithms are explored and evaluated that could be a good fit for the problem at hand (identification of text which may or may not have been posted by a depressed person). In the second step, these algorithms are used to build a system that proactively looks for content on the Internet (on places like Twitter), and identifies people who have been posting emotionally distressed content. The system also allows for making improvements to its abilities by tapping into crowd intelligence, which distinguishes the problem being addressed from normal text classification problems. A detailed explanation of this particular aspect can be found in Chapter 6. This system then serves as a first step towards extending a helpful hand to people who need it.

Chapter 2

Related Work

The work presented in this thesis falls under the combined domain of document classification and sentiment analysis, both of which have been very well researched in the past, and still continue to be an active area of research. A summary of some of the main approaches to document categorization based on their content was presented in [27]. This work also discussed representing documents in such problems (approaches similar to building n-grams), building classifiers that can perform this categorization, as well as how to evaluate such classifiers. Some work that used n-grams to build a document classifier was presented in [4], wherein an accuracy as high as 99.8% was achieved (in one particular test involving classifying documents of length more than 300 and selecting the top 400 n-grams). In the same work, documents of length less than 300 bytes and selecting the top 100 n-grams yielded an accuracy of 92.9%.

Much of the research done in the field of sentiment analysis has been summarized in [20], which also covers techniques that can be used to build systems that can enable information retrieval in opinionated data. The work done in [21] concludes that sentiment analysis is a much more challenging problem than simple text classification, and classifiers that perform well in text classification usually perform worse when it comes to sentiment analysis. For classification of movie review data from IMDB, they were able to obtain a cross validation accuracy of 82.7% using a linear-kernel SVM. Most of the research in this field focuses on full scale text documents, but not so much on microblogging platforms. Although one such work is [19], presenting a linguistic analysis of data collected from Twitter for the purposes of sentiment mining.

Support vector machines have been found to be highly successful in text classification problems mainly because of their ability to handle a large number of sparse features. This has been addressed theoretically in [14], including thorough explanations of document representation as well as efficient algorithms for training SVMs. An early analysis of the use of support vector machines in text categorization problems can be found in [13] and [6]. For the problem that [6] addressed (classifying emails as spam or not), SVMs provided the best performance when using binary features, and also took significantly less training time as compared to another model presenting comparable performance. The problem of

categorizing the Reuters [5] dataset was also addressed in [18], wherein documents were represented using the tf-idf representation. From a practical perspective, [11] presents a software implentation of support vector machines as well as a guide to effectively using them. Their implementation is (indirectly) used in the system presented in Section 7.2.

A lot of the research in ensemble learning focuses on creating ensembles out of decision trees (summarized in [26]). The reasons behind the fact that a combination (ensemble) of models can outperform a single classifier are covered in [31]. Bagging, first proposed in [3], is well suited for addressing variance problems. Bagged ensembles of support vector machines for analyzing gene expression data are built and used in the work done by [32], where it is observed that such ensembles achieve better or equal, if not worse, classification accuracy with respect to a single SVM. As noted in [1], the performance of boosting is not uniformly better for *all* datasets. AdaBoost, short for Adaptive Boosting was first introduced in [9], and was found to be performing better than using individual classifiers by [2] when tested on the Reuters [5] corpus. Stacking, first introduced in [33] is another ensemble learning method that has been studied extensively. [7] concludes that a stacking ensemble performs equally as good as the best individual classifier in the ensemble.

Even though not a lot of work has been done in applying ensemble learning to perform classification on data extracted from microblogging websites, text classification on Twitter has seen a lot of activity in the recent times. Works including [28], [10], and [12] thoroughly investigate the problem of identifying the sentiment of a tweet based on its content. While [28] focuses on putting a tweet in one out of five categories (news, opinions, deals, events, and private messages), [10] investigates the issues faced in assigning a binary sentiment (positive or negative) to a tweet in regards to a particular query term, achieving an accuracy of above 80% when including emoticons data. In spite of the fact that a tremendous amount of work has been done on identifying sentiments on Twitter, not a lot of visible work was found that addressed specifically the problem of finding out tweets displaying depressive emotions.

Part II.

Theory

Chapter 3

Machine Learning and Classification Methods

3.1. Machine Learning

Machine learning is a branch of computer science that deals with building and analyzing systems that are able to learn from data. Algorithms based on such analysis involve constructing a model from a given dataset, and then using this model to perform 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, the availability of training data is assumed, which is used to build a model so that it takes into account the structure of the given dataset. In the second step, this model is used 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.
- **Unsupervised Learning** Methods falling under this category operate in a single phase. It starts with a model with zero knowledge about the structure of the given dataset. As data is fed 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 labels.

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

3.2. Classification Methods

Text classification is a subset of machine learning algorithms used to assign specific categories to pieces of data. It is one of the msot fundamental problems in machine learning. More precisely, given some sample information (such as what kind of data points are to

be assigned to which category), these algorithms allow for assigning categories to unseen data points. Under the scope of this thesis, the data is always assumed to be English-text. Since machine learning algorithms only deal with real numbers, a necessary first step is the conversion of natural language text into real numbers. The work presented in this thesis makes extensive use of supervised text classification algorithms.

Given a dataset $\mathbf{X}$, it is required to separate the samples contained within the dataset into two (or more than two, depending on the input) classes. Formally, given a dataset that contains N instances,

$$\{(x_n, y_n)\}_{n=1}^{N},$$

where each instance (x_n, y_n) is of the form,

$$[(x_{n,1}, x_{n,2}, ...x_{n,D}), y_n],$$

each $x_{n,d}$ being the value of the feature $d \in [1, D]$, and y_n being the label of the sample which can take a limited number of possible values, the aim is to calculate the value of y_n given this feature information. In the supervised learning problem, this dataset is divided into two parts - training data (which contains the y_n values for each instance), and test data (no information on y_n is available), and the problem is to identify the values of y_n in the test data given only the information present in the training data. On the other hand, in unsupervised text classification, the model is required to identify the y_n values given only the values for x_n, there being no distinction between training and test data.

The performance of a particular classifier varies with 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 a lot of work has been done to correlate classifier performance with data type.

3.3. Support Vector Machines

Support Vector Machines (SVMs) form a fairly popular class of machine learning algorithms used mainly for binary classification and regression analysis. Given the training data, the goal of an SVM is to find a decision boundary (a hyperplane 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 resulting decision function is fully specified by a (usually small) subset of the data, and the points in this subset are referred to as support vectors.

Most classifiers resort to a distance function, in some form or the other, that can provide a similarity measurement between two points. In the simplest form of an SVM, the distance function is simply the dot product between two points, and such SVMs are referred to as *Linear Support Vector Machines*. In the case that a simple linear SVM is not able to find a sufficiently accurate decision boundary that can separate the data points into two classes

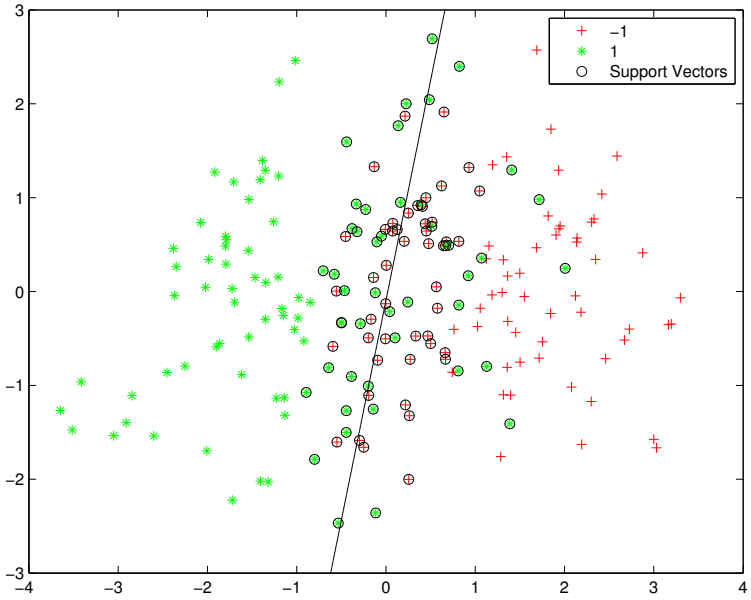


Figure 3.1.: Classification using a linear kernel SVM

(simply because the input data is not linearly separable), the so-called *kernel trick* is used, which involves transforming the data from a low dimensional space to a much higher dimensional feature space (in which the input data may be separable) using an appropriate function

$$\phi(x) : \mathbb{R}^L \rightarrow \mathbb{R}^H (L \ll H)$$

and then using the kernel function $\mathbf{k}(\phi(\mathbf{x}), \phi(\mathbf{y}))$ to actually perform the separation. The trick involved is that even though the transformation $\phi(x)$ may be expensive, computing the final similarity value $\mathbf{k}(\phi(\mathbf{x}), \phi(\mathbf{y}))$ is not.

Some of the most popular kernel functions include

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

Figure 3.1 shows a scenario where a support vector machine is used to classify two sets of points that are generated from partially overlapping Gaussian distributions. This dataset can be classified by a linear kernel SVM since this data is linearly separable.

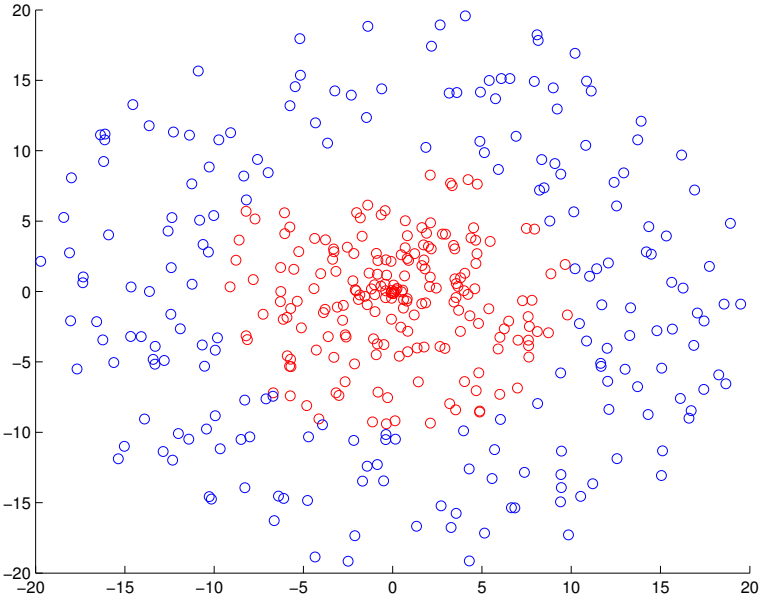


Figure 3.2.: A dataset which is not linearly separable

However, for a case when the data is not linearly separable and looks like as shown in Figure 3.2, the linear kernel performs poorly. This is shown in Figure 3.3. When the dataset has such characteristics, a much better option is to use a kernel that is able to work with such data. For instance, an RBF kernel draws a much better boundary in this case, as shown in Figure 3.4.

Intuitively, the work done by the $\phi(x)$ function can be visualized from Figure 3.2, and Figure 3.5. Figure 3.2 shows the original dataset that the linear kernel failed to classify with reasonable accuracy. A simple transformation can be applied in this case to make the data separable, transforming the two dimensional input to a three dimensional input by adding the following transformation.

$$\phi(x) = \sqrt{x_1^2 + x_2^2}$$

This transformation transforms the input data in the shape of a three dimensional cone which is separated into two parts by a simple plane in three dimensions. The points in the inner circle form the tip and the surrounding areas of the cone, while the rest of the cone is constituted by the remaining points. The resulting data looks like Figure 3.5.

The code for this transformation (including the generation of the original data) is shown in Listing 3.1.

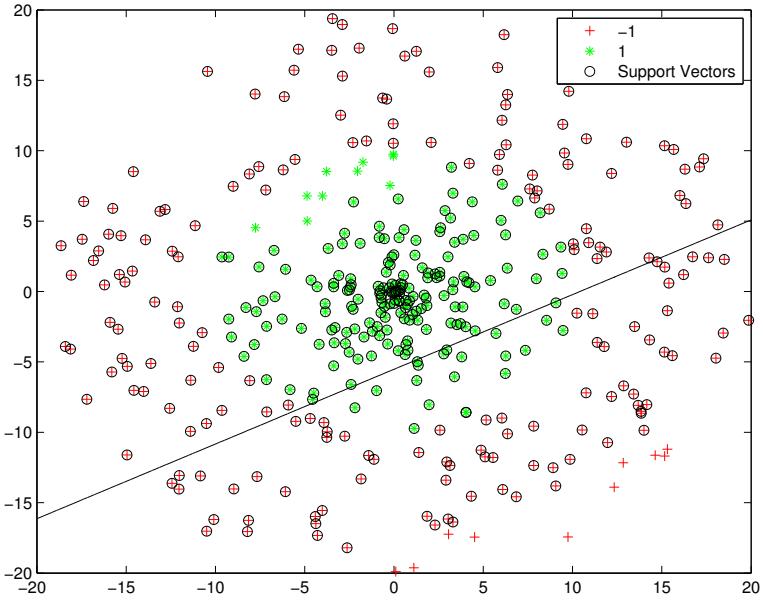


Figure 3.3.: Dataset for Figure 3.2 being classified using a linear kernel

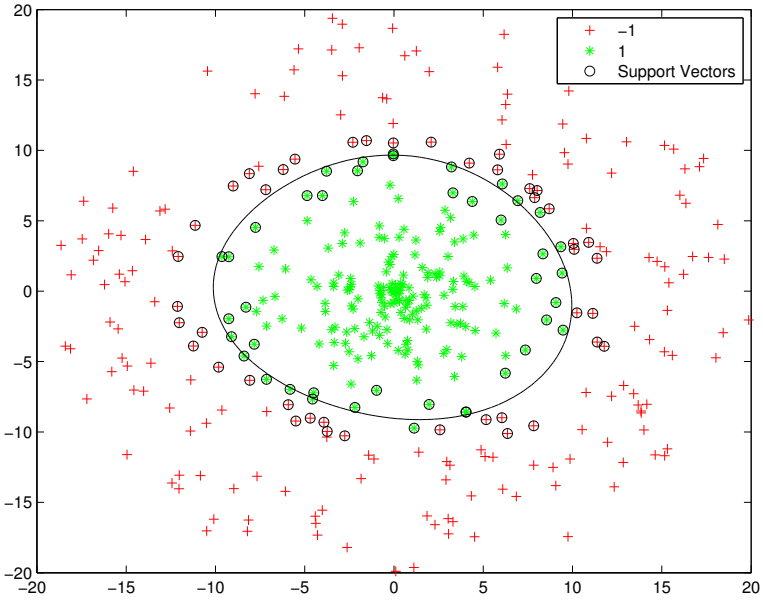


Figure 3.4.: Dataset for Figure 3.2 being classified using an RBF kernel

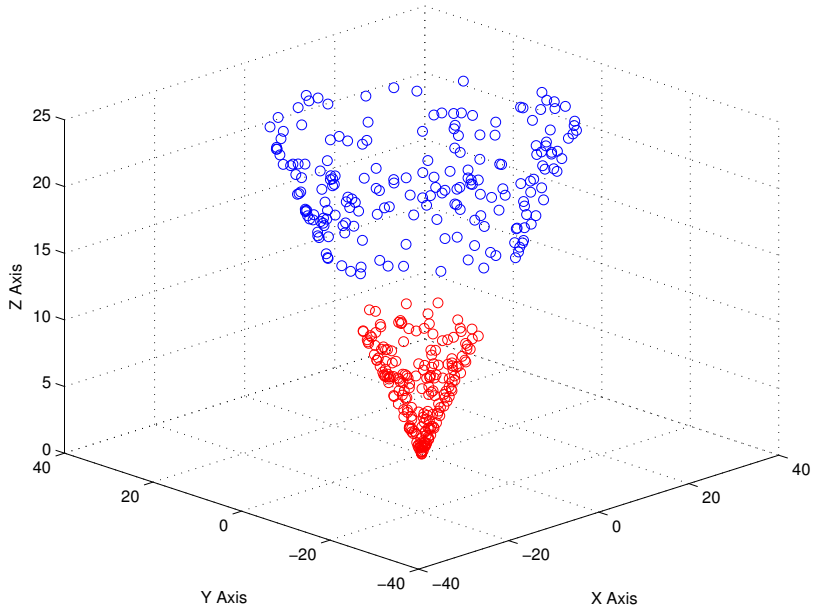


Figure 3.5.: The dataset from Figure 3.2, exported into a three dimensional space, which can now be separated by a linear plane

```
% constants
n = 200; radius = 10; theta = 2 * pi * rand(n, 1);

% points inside the circle
r_in = rand(n, 1) * radius;
x_in = r_in .* cos(theta); y_in = r_in .* sin(theta);
label_in = ones(n, 1);

% points outside the circle
r_out = radius + rand(n, 1) * radius;
x_out = r_out .* cos(theta); y_out = r_out .* sin(theta);
label_out = -1 * ones(n, 1);

% final points
instances = [x_in y_in; x_out y_out;];
labels = [label_in; label_out;];

% plot the points in 2D
figure(1);
scatter(instances(1:200, 1), instances(1:200, 2), 'r');
hold on;
```

```
22 scatter(instances(201:400, 1), instances(201:400, 2), 'b');
23 hold off;
24
25 % transform the input to 3D and plot again
26 figure(2);
27 instances = [instances sqrt(power(instances(:, 1), 2) + power(
      instances(:, 2), 2))];
28 scatter3(instances(1:n, 1), instances(1:n, 2), instances(1:n, 3),
       'r');
29 hold on;
30 scatter3(instances(n + 1:end, 1), instances(n + 1:end, 2),
      instances(n + 1:end, 3), 'b');
31 xlabel('X Axis'); ylabel('Y Axis'); zlabel('Z Axis'); grid on;
32 hold off;
```

Listing 3.1: Generating the dataset to be classified by a linear classifier

Chapter 4

Text Representation

Machine learning algorithms are designed to operate on real values. To classify documents, a prerequisite is to convert the natural language text to a format that such an algorithm can understand and work on. One such format is the vector space model. Processes to convert natural language text to such formats include obtaining text from documents, building a vocabulary, text preprocessing (such as tokenization, stemming, and stop words removal), and converting terms to actual usable feature values. The processes may differ slightly for different languages. For instance, tokenization for English text is different from tokenization for Chinese text because of differences in word boundaries. But these differences are minor and do not change the overall flow.

4.1. Preprocessing

Preprocessing documents typically involves all the steps that are needed to be performed before proceeding on to extracting meaningful information from text that the classifier could handle. Text documents in their original forms normally contain a lot of redundant information that usually does not affect how a classifier behaves. Preprocessing 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.

- Extraction - Data extraction involves extracting relevant data out of documents. Often, documents are unstructured, poorly structured, or are in a format such as XML or HTML. In such cases, the documents in their original forms contain a lot of redundant information. Extracting relevant data by using methods like parsing brings documents to a state where they are ready to be tokenized.

- 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 and white spaces and linebreaks 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 some 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. One approach to solving such problems is using n-grams, which is 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. Consequently, since these words appear a lot in almost every document, it may lead to high and 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, the words *car*, *cars*, *car's* and *car's*, must all 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.

4.2. Representation and Vector Space Classification

After a document has been preprocessed, it needs to be represented in terms of the useful content that it has, such that the relative importance of each word has been taken into account. Since a preprocessed document is just a collection of tokens, it can be represented as a vector

$$\mathbf{x}_n = (\mathbf{x}_{n,1}, \mathbf{x}_{n,2}, ..., \mathbf{x}_{n,D})$$

where each dimension $\mathbf{x}_{n,d}$ corresponds to a token, 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, consider two simple HTML documents containing English text,

```
<html>
    <head> <title>Munich</title> </head>
    <body> We are headquartered in Munich, Germany. </body>
</html>
```

and

```
<html>
    <head> <title>Berlin</title> </head>
    <body> We also have an office in Berlin, Germany. </body>
```

```
</html>
```

After data extraction, the documents look like the following,

```
Munich We are headquartered in Munich, Germany
```

and

```
Berlin We also have an office in Berlin, Germany
```

It can be seen that all the HTML tags have been stripped off and only the relevant information remains. The documents are further subjected to token extraction, after which they look like the following,

```
Munich | We | are | headquartered | in | Munich | Germany
```

and

```
Berlin | We | also | have | an | office | in | Berlin | Germany
```

The next step in preprocessing involves removing stop words and stemming. After removing the stop words, the only tokens that are left in the corpus include “Munich”, “headquartered”, “office”, and “Germany”, forming a term dictionary over which will not really illustrate the vector space representation in a good way. Hence, ignoring the stop words removal for the sake of this example, the corpus looks like the following after stemming

```
{
    'We': 1,
    'are': 2,
    'headquarter': 3,
    'in': 4,
    'Munich': 5,
    'Germany': 6,
    'also': 6,
    'have': 8,
    'an': 9,
    'office': 10,
    'Berlin': 11
}
```

and the documents are then represented as the following vectors

```
{1, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0}
```

and

```
{1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 2}
```

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

In this example, the values represent 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. An even better improvement is to use the $\mathrm{tf} - \mathrm{idf}$ value, which (for a term in the document) is defined as

$$\mathbf{tfidf}_{t,d} = \mathbf{tf}_{t,d} * \mathbf{idf}_t$$

where $\mathrm{tf}_{t,d}$ is the term frequency of the term $\mathbf{t}$ in document $\mathbf{d}$, and idf_t is the inverse document frequency of the term $\mathbf{t}$ across the entire collection of documents, usually defined as $\log(N/\mathrm{df}_\mathrm{t})$, 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 tf-idf value 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 it provides a good representation of a document in the vector space format. This representation can now be used to calculate the similarity between two documents. The standard way for this is to compute the dot product between the two vector representations

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

Chapter 5

Ensemble Learning

Ensemble Learning is a class of supervised 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 of classifiers involve combinining multiple classifiers to construct an expectantly better classifier. The performance of an ensemble is not guaranteed to be better than the individual classifiers, but depending on the problem, an ensemble usually outperforms its constituent classifiers by a fair margin. But like almost all machine learning methods, a generalization cannot be made and it depends a lot on the problem at hand. Using ensemble learning requires performing more computation, but yields better results when the classifiers are not similar to each other.

5.1. Bagging

Bagging, also referred to as Bootstrap Aggregation, is one of the most basic forms of ensemble learning. Given a training dataset D containing N samples, each sample having D different features, the aim of bagging is to combine M classifiers, each trained using a different subset of the main data, such that every classifier learns about a different structure of the input data. When the outputs from all the constituent classifiers are combined, the final output is expected to be better than the individual predictions. Bagging requires that the entire system should not be stable, so that when the training data changes by even small amounts, the resulting classifiers are different [22]. A consequence of this was also presented in [3], where it was noted that for unstable classifiers, bagging introduces improvements in the net classifier accuracy. On the other hand, if the underlying classifiers are already strong and stable, the performance may degrade a little [3]. Bagging is often known as the less risky approach, in that there is a high chance that using bagging improves the performance of the classifier.

To train different classifiers 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, $N'(< N)$ samples are picked with replacement, and each time a new classifier is trained 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, every new classifier is trained 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.

5.2. Boosting

The original hypothesis behind Boosting was the notion that a set of weak learners can be trained to form a strong learner [16]. Boosting aims to build an ensemble of classifiers 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 instance 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. This way, each classifier has an incentive to focus more on classifying correctly those points which the previous classifier classified incorrectly.

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 Adaptive Boosting, originally presented in [8]. 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.

The algorithm used to train a model over a dataset is the following.

1. Initialize the sample weights $w_n = 1/N$, for $n = 1, 2, ..., N$
2. For $m = 1, 2, ..., M$:
   - Train a classifier $\mathbf{y_m}(\mathbf{x})$ on the training data using the sample weights $\mathbf{w}$
   - Calculate
   $$\epsilon_m = \frac{\sum_{n=1}^{N} w_n I_n}{\sum_{n=1}^{N} w_n}$$
   where $I(y_m(x_n) \neq y_n)$ is the indicator function and equals 1 when the predicted label $y_m(x_n) \neq$ the actual label y_n, and 0 otherwise
   - Set
   $$\alpha_m = \log(\frac{1 - \epsilon_m}{\epsilon_m})$$

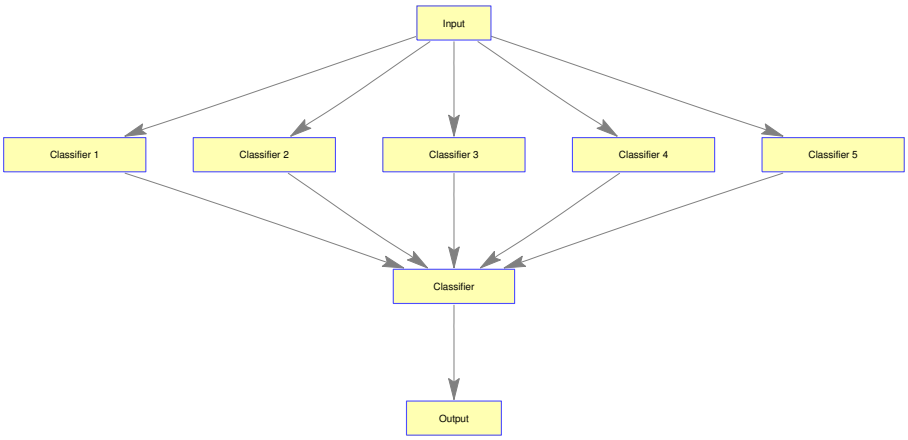


Figure 5.1.: Stacking

- Update **w** using

$$w_n := w_n \exp(\alpha_m I(y_m(x_n) \neq y_n))$$

After all the models have been trained, the prediction for an instance x_n is obtained by calculating

$$sign(\sum_{m=1}^{M} \alpha_m y_m(x_n))$$

5.3. Stacking

Stacking, also called Stacked Generalization was first introduced in [33], and obtains predictions by a two step process. It is assumed that there exists a training input matrix that has N rows for D samples, and D columns for D features. In the first step, M distinct classifiers are trained on the training data. After the classifiers have been trained, a new dataset of N rows and M columns is generated, each column being populated by predictions for each of the M classifiers. This new dataset forms the training data for a second level classifier, from which the final prediction is obtained. The process for prediction from a Stacking ensemble follows a similar path. Given a sample x_n of the form $[(\mathbf{x}_{n,1}, \mathbf{x}_{n,2}, ...\mathbf{x}_{n,D})]$, it is first run through the M classifiers to obtain M predictions (y_m) for $m = 1, 2, ..., M$. These values are then fed to the second level classifier, which yields the final label.

Figure 5.1 illustrates how stacking works when there are 5 classifiers at the first level. The node "Input" refers to the sample x_n for which the prediction is to be made. It is fed to each classifier, and the resulting predictions from the classifiers form an M dimensional vector, which is then fed to the main classifier at the second level, which then gives the final label. Two of the most common schemes for training classifiers at the first level include bootstrapping (training different classifiers using different subsets of data) and feature sampling (using different features as inputs for different classifiers). Stacking is one of the best ensemble learning methods that has proven to be successful in a number of works, as well as in the real world. The study done in [7] experimented with a number of datasets to compare stacking against multiple schemes for combining classifiers, to discover that a stacking ensemble performs much better than when classifiers are combined using the *voting* or *select-best* schemes. The nature of inputs for the second level classifier were studied in [29]. This work concluded that much better results can be obtained if the confidence values from the first level classifiers are included in the input for the higher level model. Stacking was also employed by the winning team at the Netflix prize [17], in which the solution included meta features as inputs (number of users and number of movie ratings) in addition to using the normal feature vectors.

Part III.

Experimental Results

Chapter 6

Experiments

6.1. Dataset

This work focuses on detecting emotional distress amongst the general public from publicly available tweets/blogs. To predict the label for a particular tweet, machine learning models first need to be trained on similar data. This means that availability of a collection of tweets, each having a positive/negative label assigned to it was a prerequisite.

Non availability of such data in the beginning of this work led to the following approach - conduct benchmarks and evaluations on a similar problem, and apply the results to build the final system that can perform the said task. Experiments were initially performed on a dataset made available by a machine learning competition website [15], where the aim is to predict whether a certain piece of text (a comment from a conversation on the internet) can come across as insulting to a user or not. The dataset was a list of 6182 comments, each with a binary label.

Preprocessing this text consisted of - cleaning the text (allowing only alphanumeric characters and removing unnecessary punctuation marks), building n-grams of size 2, and using tf-idf information as feature values. Performing these steps resulted in an input matrix consisting of 6182 rows and 23175 columns. Each row corresponds to a distinct comment and each column corresponds to a distinct feature (n-gram, in this case).

After the evaluations, the next step in the work required the availability of an unlabelled dataset which could be labelled using crowd intelligence, and which could then finally serve as a basis for the final system to identify emotional distress. This data was fetched from the website Reddit [23], which is an online community for people to interact with one another. This website hosts a number of subwebsites called subreddits, two of which were of particular interest - *"/r/happy"* [24] and *"/r/suicidewatch"* [25]. The first subreddit is where people submit content if they are feeling happy and want to share their happy moments with others. The second subreddit is where people post when they feel depression, loneliness, feelings of helplessness, and feelings of ending their own lives. On this website, one piece of content is called a "story", having a title and a full text. Since the

training data is supposed to resemble a tweet in length, only the shorter length "title" of the story was selected. Consolidation of dataset from these websites involved downloading and labelling a total of 2000 stories, 1000 each from *"/r/happy"* and *"/r/suicidewatch"*. This builds a strong foundation for the system, after which stories are left to be fetched as per a pre-defined crontab schedule.

Finally, the main data for identifying emotional distress is fetched from Twitter. Starting from January 17, Twitter's public streaming API [30] was used to fetch 100 tweets every 3 hours, for a total of 800 tweets everyday. Interruptions were faced in this process from February 5 until February 15. This gives an overall view of the general sentiment of the public, on which the analysis was performed .

6.2. Approach and Setup

The work done in this thesis was done in two phases. In the first phase, various machine learning techniques were evaluated, including support vector machines and ensemble learning methods. In the second phase, a system was built that monitors emotional distress on the internet.

In the evaluation phase, the comments dataset from Kaggle [15] was used. Bigrams (n-grams of length 2) were extracted and used as features in the vector space representation of comments. To assign values to these features, the tf-idf information as collected from across all comments was used. Once this representation was obtained, the dataset was then evaluated as per k-fold cross validation. To evaluate the performance of a particular model, the model was trained over 90% of the comments, tested over the remaining 10%, and the process was repeated 10 times to obtain an average value. This procedure was repeated for standalone support vector machines (using linear, polynomial, gaussian, and RBF kernels) and the ensemble learning methods which include bagging, boosting, and stacking.

In the second phase, a web based system was built that is able to identify emotional distress amongst the general public on Twitter, and allow anyone to contribute to the training data (using crowd intelligence). To incorporate crowd intelligence, the appropriate dataset (from Reddit) was downloaded and presented to the users of the system to label. This process resulted in consistent growth of the training data, which in turn improved the machine learning models continuously as and when they were retrained. These models were then used to make a more informed analysis about the level of distress on Twitter.

Chapter 7

Results

The final output of this work is twofold - an evaluation of support vector machines and ensemble learning methods in the domain of text classification, and a web based interface that implements the evaluated techniques and presents a system that can monitor the public feed from the Internet to find out people who are suffering emotional distress and may need help.

7.1. Evaluation

The evaluation of all algorithms is done on the comments dataset [15]. Scores from normal supervised learning and online supervised learning are calculated and presented. To report online learning scores, the learning continues in iterations. In each iteration, 100 more random samples are picked on which the models are trained and the accuracy is measured. This is continued until no more samples are left. While this does not conform precisely to the definition of online learning, this approach is still used because of two reasons. First, this is very similar to how the classifiers in the system are updated each time there is new training data available. Second, in the most elementary approach for online learning, a new and updated classifier is obtained by retraining on the support vectors from the original classifier and the new sample. Under the experimental settings for the problem at hand, this approach yielded very poor results which could not be used to implement the final system. All accuracies reported henceforth are calculated using 10-fold cross-validation, training the model on 90% of the data and testing on the remaining 10%, and repeating this process 10 times to get the average of all 10 measurements.

7.1.1. Support Vector Machine

The behavior of support vector machine based classification depends heavily on the kernel being used. Using a linear kernel yields a cross validation accuracy of 79.06%, while using either of polynomial/RBF/sigmoid kernels gives an accuracy of only 34.39%. This can be attributed to the notion of how kernel functions operate. A kernel function calculates the similarity between two vectors when the vectors have been transformed from their original low dimensional space into a high dimensional space. In the current scenario, the text

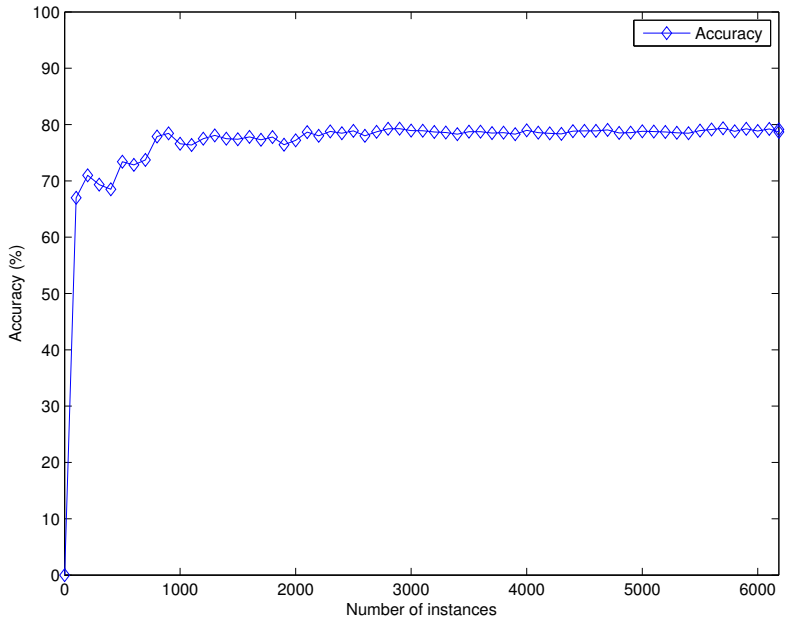


Figure 7.1.: Accuracy of a linear kernel support vector machine, measured against the total number of instances

corpus contains 23175 features, which is already a very high number. Any transformation from this space into an even higher space results in a decrease in the final accuracy calculated, as can be noted in the results.

When using a linear kernel, the accuracy of an SVM shows an increase as the number of input instances increases. The score ranges from 67.00% when measuring on 100 samples, to 79.35% when measuring on 5800 samples, as shown in Figure 7.1.

Support vectors is a term for the subset of data points that the main classifier uses to establish the margin between the two class of points. Measuring the number of support vectors against the total number of instances gives an idea about the performance of the classifier - the more the number of support vectors (relative to the number of instances), the harder it is to perform a classification. Figure 7.2 shows this variation for the problem at hand. The number ranges from 90 for 100 samples to 3771 for 6182 samples, the increase being almost linear in the number of instances. It can be noted that the ratio of support vectors to the total number of points decreases from 0.9 to around 0.6, which shows that the more the information available to the classifier, the easier the task at hand becomes.

7.1.2. Ensemble Learning

Ensemble learning algorithms are essentially a combination of individual classifiers. In this work, these individual classifiers are support vector machines, which means that ir-

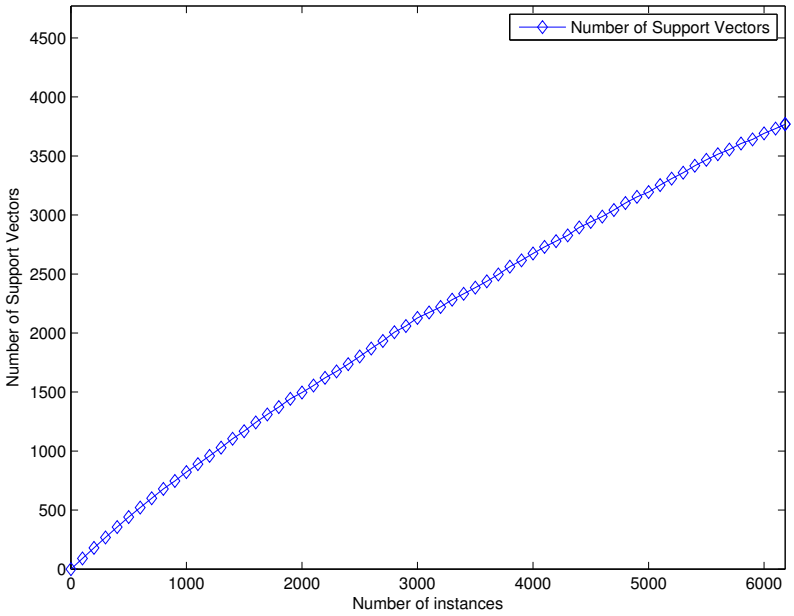


Figure 7.2.: Variation of the number of support vectors in a linear kernel SVM, measured against the total number of instances

respective of which class of ensemble methods is used, the behavior is going to depend on the kernel being used. Since the maximum accuracy in the section on support vector machines was found with the linear kernel, the same is used in the following experiments. In all experiments presented in this section, except for measuring the accuracy of a bagging ensemble against the number of models, the total number of models comprising an ensemble is set to 9.

Bagging

The increase in the accuracy observed by a bagging based ensemble is similar to the increase observed in simple support vector machines. This can be attributed to the fact that bagging is essentially just a combination of SVM classifiers, with a majority vote deciding the final label of the instance. This can be observed in Figure 7.3. A peak value of 79.97% is observed when the ensemble has to classify 6000 data points. Similarly, when the classifier was trained on all 6182 points without any iterations, an average accuracy of 79.65% was observed, which is a small improvement over the SVM classifier.

On the other hand, Figure 7.4 suggests that on varying the number of underlying classifiers from 1 to 20, the accuracy of a bagging based ensemble does not vary too much with the total number of models being used underneath. A maximum accuracy of 80.02% is observed for 7 classifiers, while the minimum value is 73.66% for 2 classifiers.

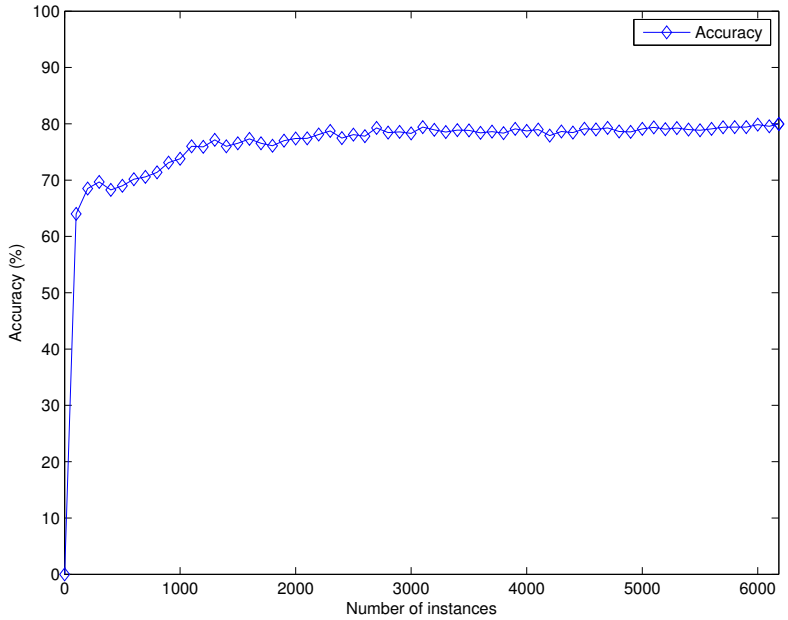


Figure 7.3.: Accuracy of a bagging ensemble (comprised of 9 linear kernel support vector machines) as measured against the number of instances

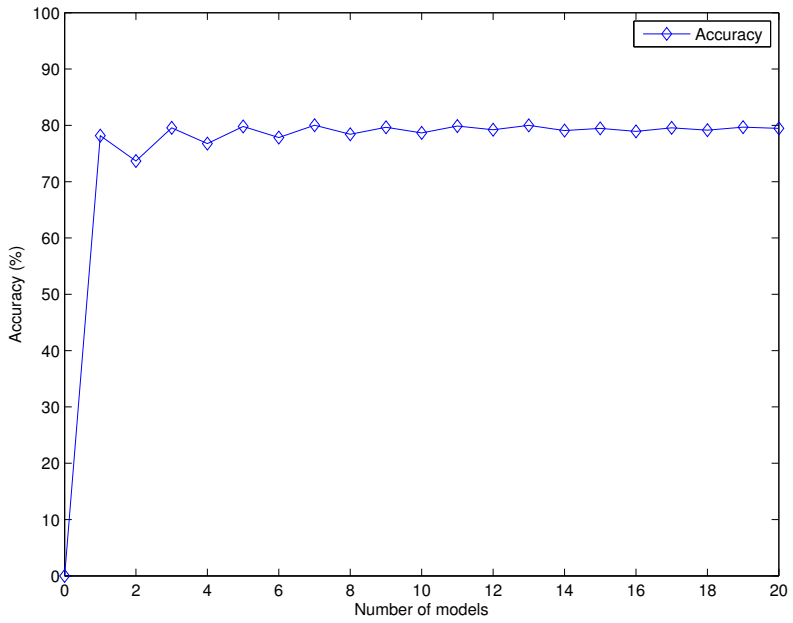


Figure 7.4.: Variation of accuracy of a bagging ensemble with the number of constituent models

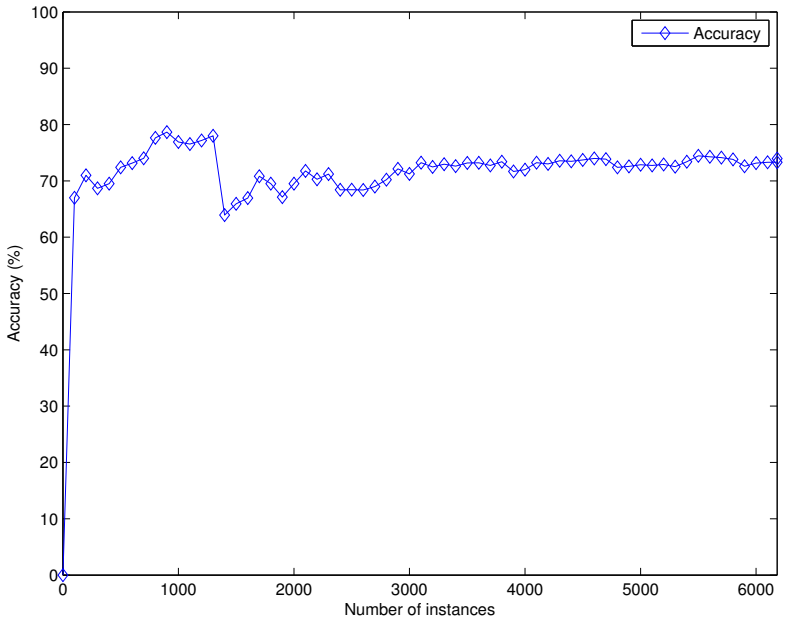


Figure 7.5.: Accuracy of a boosting ensemble (constituted by 9 linear kernel support vector machines) as measured against the number of instances

Boosting

Boosting follows a different approach to ensemble learning, by combining classifiers in a way that each successive classifier gets more incentive to classify correctly the points which the previous classifier misclassified. As shown in Figure 7.5, the performance of a 9-classifier ensemble peaks at 78.66%, and does not vary a lot as the number of instances increases. However, the average performance observed was 72.84%, which is considerably less than that in case of support vector machines.

Stacking

Stacking follows a more meta-level appraoch to classification, by training two levels of classifiers to perform the final classification. It can be seen in Figure 7.6 that the performance of stacked classifiers increases with the total number of instances, similar to the other two methods of ensemble learning. A maximum performance of 79.76% is seen at the point when the model has to classify all 6182 instances, while the minimum (67%) is seen when measurements are done on 100 instances. Similarly, when the classifier was trained on all 6182 instances without any iterations, an average accuracy of 79.48% was observed, which is again a very small increase over the performance of a single support vector machine.

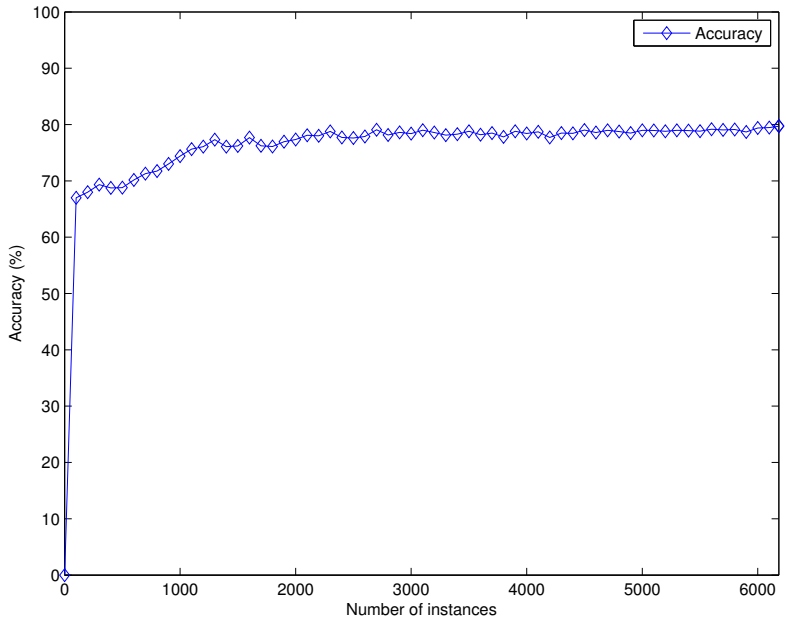


Figure 7.6.: Accuracy of linear kernel based stacking ensemble as measured against the number of instances

7.2. System Details

The second major contribution of this thesis is a web based system that allows users to perform various functions including helping building the training data, evaluating a level of distress amongst the general public by monitoring the status of the classifier, and keeping a check on particular instances of content that may appear to be distressed and qualified as needing-attention.

Users help build the training data by assigning labels to stories fetched from Reddit. This accounts for the crowd intelligence part of the whole system, and constitutes the *Ratings* module on the site. The *Monitoring* module then uses this data to train the classifiers and present information about a general level of distress amongst people who are posting on Twitter (grouped by date) as monitored by each class of classifiers (SVM, Bagging, Boosting, and Stacking). The module also presents information about individual instances of content that are most likely to be depressed. This may then serve as a first step towards providing further attention (in the form of human help) to the authors of these tweets.

As soon as a user visits the front page of the web interface, they are presented with a choice between the *Ratings* and *Monitoring* modules, as shown in Figure 7.7.

The division of the system into the two mentioned modules is from a user's perspective. From an infrastructure perspective, the system can be seen as being powered by two

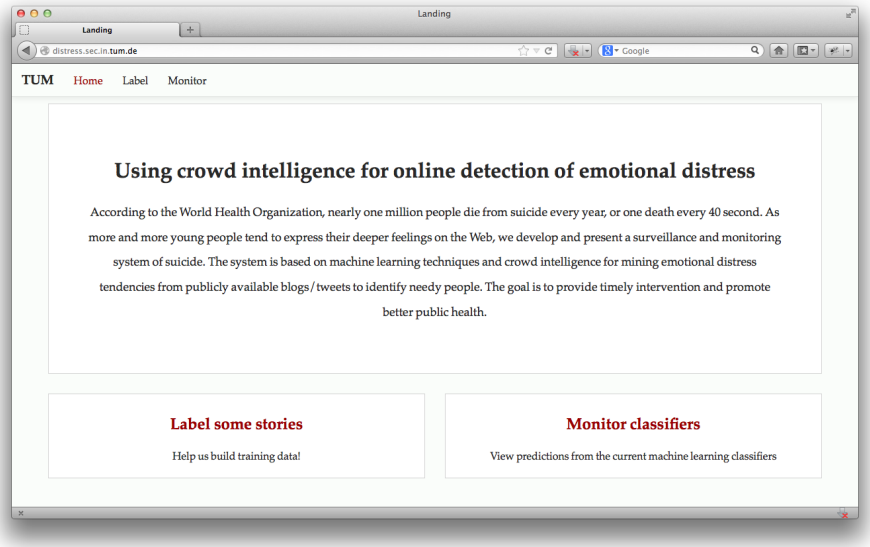


Figure 7.7.: Landing page

components, a manual one and an automatic one. The manual component includes users assigning labels to posts fetched from Reddit, which ultimately serves as the training data. The automatic component includes cron jobs (pieces of code that are executed at specific intervals of time) and involves fetching 800 tweets every day from the public stream of Twitter, collecting and consolidating statistics about the total number of depressed and not-depressed content posted each day, and updating the labels of all fetched tweets according to the updated models. In the next two sections, a discussion is presented about the system components from the user's perspective.

7.2.1. Ratings

The *Ratings* module is built to help consolidate training data. As mentioned earlier, 1000 posts from Reddit are fetched every 2 days (500 each from "/r/happy", and "/r/suicidewatch"), which do not contain any label information. As shown in Figure 7.8, this module presents a user with a random piece of text fetched from reddit, and two options for assigning this story a label. The label could be either "depressed", or "not depressed". If in case there are no unlabelled stories left, a simple message saying "No unlabelled stories left" is displayed.

7.2.2. Monitoring

The *Monitoring* module is responsible for performing mainly two functions. The first function is to display a list of the top few tweets that are most likely to be depressed. This calculation is done based on the number of classifiers that classified a tweet as depressed. This is illustrated in Figure 7.9 and Figure 7.10.

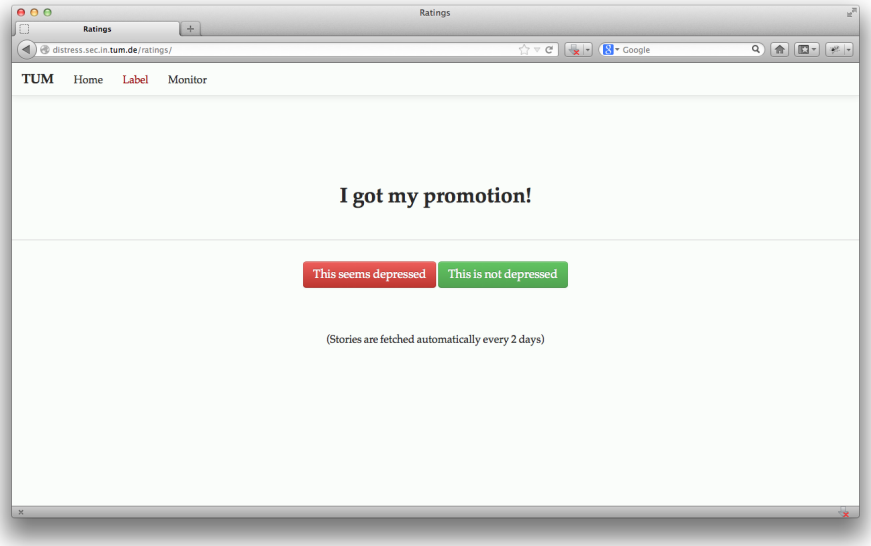


Figure 7.8.: Ratings module

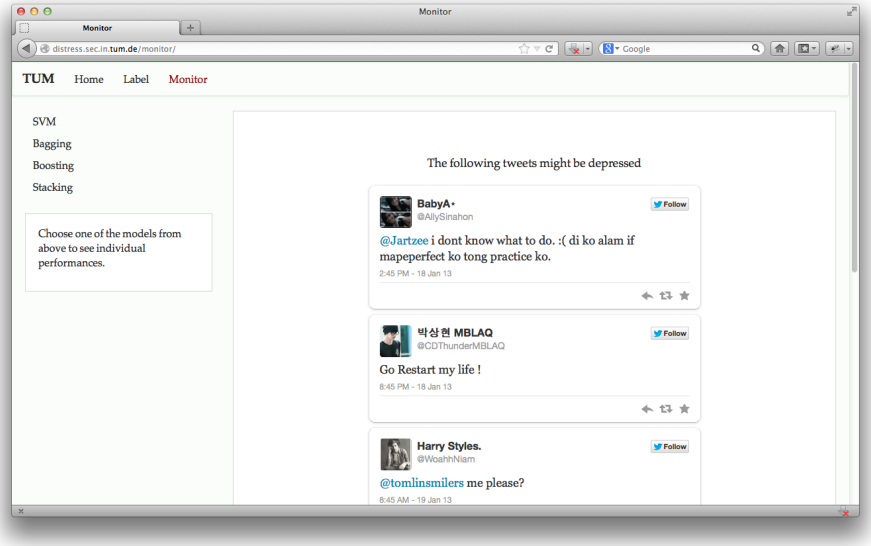


Figure 7.9.: Landing page of Monitoring module

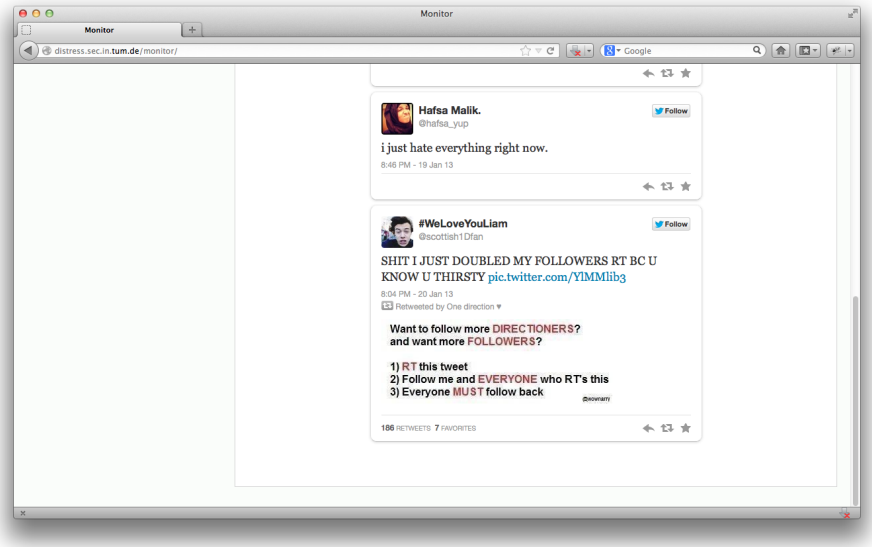


Figure 7.10.: Landing page of Monitoring module

As can be seen in Figure 7.10, one of the tweets displayed has the text *"i just hate everything right now"*. Such content can be associated with feelings of resentment, and so this is a good candidate to pass on to a human for a final call on whether the person posting this is depressed or not. Similarly, Figure 7.9 shows a tweet with the content that includes the phrase *"i dont know what to do"*. A lot of content like this was found on "/r/suicidewatch", where people often post when they feel confused in a very negative connotation. Hence, this tweet alone gives somewhat of an indication that this person might be feeling depressed, and this belief could be further solidifed (or not) after looking at the entire stream of this person.

The second function that the *Monitoring* module performs is that of displaying statistics about how many tweets were classified as depressed on a particular day. Figure 7.11 shows the page that is displayed when a user chooses to view the statistics for the bagging classifier.

As can be seen in the figure, the statistics are displayed for a time period beginning from January 17 (this is when the process of collecting tweets from Twitter began) until March 18, with intervals of 15 days between two consecutive sections in the graph. The graph can also be zoomed in on a smaller period of time, as shown in Figure 7.12, in which case only the statistics for that period of time are displayed. The zoom level can be reset using the *"Reset Zoom"* button.

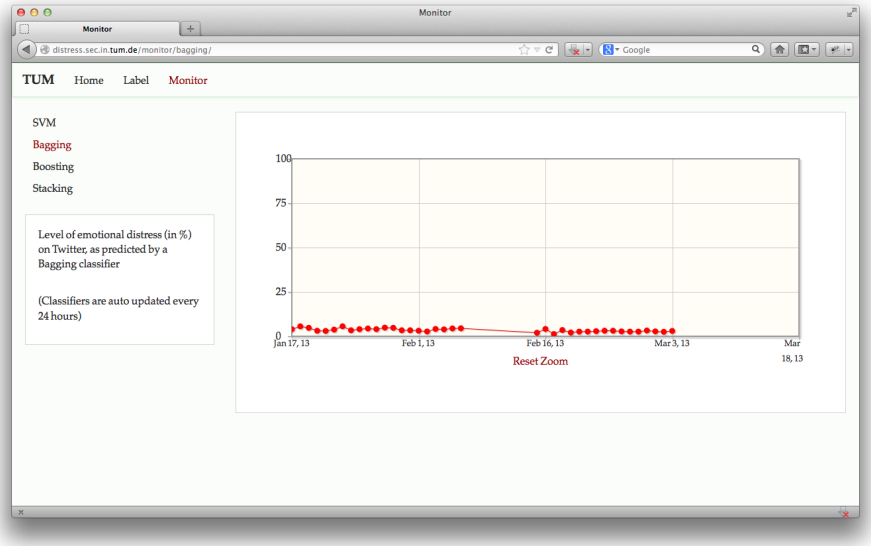


Figure 7.11.: Statistics for the bagging classifier

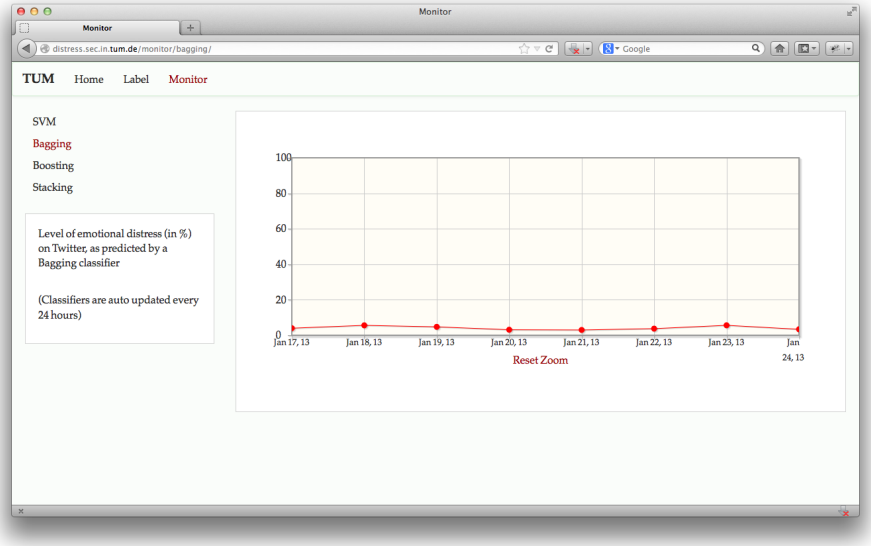


Figure 7.12.: Zooming in on a particular interval of time for Bagging statistics

Part IV.

Conclusion and Future Work

Chapter 8

Conclusion and Future Work

This work presented two results. The first was an evaluation of support vector machines and ensemble learning methods in the domain of text classification. It is well known that not all machine learning techniques suit every problem. Indeed, while Boosting is one of the best ensemble learning methods having been found to show significant improvements in accuracy in a number of works, it was outperformed by a support vector machine classifier, a bagging ensemble, and a stacked classifier. For the dataset used in this thesis, the accuracy obtained by a linear kernel support vector machine was found to be 79.06%, which was chosen as a baseline against which all other performances were compared. When 9 such SVMs were combined to form a bagging ensemble, the accuracy obtained was 79.65%. This is a small improvement over the baseline classifier. Similarly, when the same number of SVMs were combined to form a boosting ensemble, the accuracy obtained was 72.84%, which is significantly less than the simple support vector machine. The third and final ensemble learning method that was evaluated in this thesis was stacking, which gave an average accuracy of 79.48%, which is again a small improvement over the baseline.

The second contribution of this work is a web based system that is able to detect distress amongst people who choose to post their inner feelings on the web. A system is presented that uses a support vector machine, a bagging ensemble, a boosting ensemble, and a stacking ensemble to make its predictions on public tweets on the Internet to classify them as depressed or not depressed. The top few tweets are then listed on a section on the website, so that the end user can check whether the system is calculating reasonable results or not. In the experiments performed under the scope of this work, the results were found very reasonable. Text classified as depressed included tweets like "'Scared and don't know what to do" and "i just hate everything right now". On comparing it to the text posted on `/r/reddit`, it was found that the classification was not wrong. Another component of the system was a real time detection of the level of distress present on the Internet. It is difficult to qualitatively evalute this component, since this measure will only increase when a worldwide event happens which causes widespread depression. Hence, the only way to evaluate this component was to make sure that it was not predicting high levels of depression amongst the general public when absolutely nothing happened. This was found to be true.

The main scope for future work in this thesis is presented by the web based system. The

following list describes some of the improvements and additions that could be performed in order to improve.

- **Fetch more tweets** Currently, the system fetches 100 tweets every 3 hours, or 800 tweets per day. This is a very safe number, so as to not exceed the API limits of Twitter. While in the performed experiments, this number gave sufficiently reasonable results, the prediction quality could be further increased by increasing this number and fetching more number of tweets per day.
- **Increase the crowd intelligence involved** The system trains the machine learning models based on data collected from reddit and labels collected from users assigning them, the labelling being powered by crowd intelligence. Currently, only 1000 posts from reddit are fetched every day, and a lesser number of them are labelled every day. The system could benefit from having a widespread adoption as well as more reddit posts being fetched every day.
- **Relabelling process** Every day at 2AM, all the models are retrained, and all the tweets are re-classified. This is a slow process, and could be done away with. Attention worthy text needs to be acted upon immediately, hence re-classifying old tweets is important from an evaluation perspective, but not really helpful in a real world deployment of this system. Removing this piece of code execution will result in a more efficient system.
- **Select one model** Currently, the system uses 4 models to make its predictions, SVM, bagging, boosting, and stacking. The best performance was observed using SVM, Bagging, and Stacking. For a real world deployment of the system, eliminating the classifiers which are not needed and selecting the one which is most accurate is going to improve the system in the long run.

Appendix

Appendix A

Appendix

All the code used is this thesis can be downloaded from http://github.com/siddhantgoel/thesis. The root directory contains 4 directories - "MATLAB", "Python", "Report", and "Presentation". The "MATLAB" folder contains all the MATLAB code that was used to evaluate the machine learning models. The "Python/Code/" folder contains two subfolders - "models" and "web". The "models" folder contains the Python version of the MATLAB code used for the evaluations, and the "web" folder contains all the code for the web based system that was developed. The code is written using the Django (http://www.djangoproject.com) web framework, and contains all the scripts necessary to run the system. To run the project, visit the "Python/Code/web/" directory and run "supervisord", after making the appropriate changes to the "supervisord" configuration file, which should be located at "/etc/-supervisord.conf". The configuration files under use can be found in the same folder. The "Report" folder contains the LaTeXcode for this thesis, and the "Presentation" folder contains the LaTeXcode for the presentation accompanying this thesis.

Bibliography

[1] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1):105–139, 1999.

[2] Stephan Bloehdorn and Andreas Hotho. Text classification by boosting weak learners based on terms and concepts. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 331–334. IEEE, 2004.

[3] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[4] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

[5] David D. Lewis. Reuters-21578, Distribution 1.0. `http://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection`.

[6] Harris Drucker, Donghui Wu, and Vladimir N Vapnik. Support vector machines for spam categorization. *Neural Networks, IEEE Transactions on*, 10(5):1048–1054, 1999.

[7] Saso Džeroski and Bernard Ženko. Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004.

[8] Yoav Freund and Robert Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[9] Yoav Freund, Robert Schapire, and N Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[10] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, pages 1–12, 2009.

[11] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[12] Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, and Tiejun Zhao. Target-dependent twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 151–160, 2011.

[13] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137–142, 1998.

[14] Thorsten Joachims. *Learning to classify text using support vector machines: Methods, theory and algorithms*, volume 186. Kluwer Academic Publishers Norwell, MA, USA:, 2002.

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

[16] Michael Kearns. Thoughts on hypothesis boosting. *Unpublished manuscript, December*, 1988.

[17] Yehuda Koren. The bellkor solution to the netflix grand prize. 2009.

[18] Larry M Manevitz and Malik Yousef. One-class svms for document classification. *the Journal of machine Learning research*, 2:139–154, 2002.

[19] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *Proceedings of LREC*, volume 2010, 2010.

[20] Bo Pang and Lillian Lee. *Opinion mining and sentiment analysis*. Now Pub, 2008.

[21] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[22] J Ross Quinlan. Bagging, boosting, and c4. 5. In *Proceedings of the National Conference on Artificial Intelligence*, pages 725–730, 1996.

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

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

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

[26] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(3):660–674, 1991.

[27] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[28] Bharath Sriram, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu, and Murat Demirbas. Short text classification in twitter to improve information filtering. In *Proceeding of the 33rd international ACM SIGIR conference on research and development in information retrieval*, pages 841–842. ACM, 2010.

[29] Kai Ming Ting and Ian H. Witten. Issues in stacked generalization. *arXiv preprint arXiv:1105.5466*, 2011.

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

[31] Giorgio Valentini and Francesco Masulli. Ensembles of learning machines. *Neural Nets*, pages 3–20, 2002.

[32] Giorgio Valentini, Marco Muselli, and Francesca Ruffino. Bagged ensembles of support vector machines for gene expression data analysis. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pages 1844–1849. IEEE, 2003.

[33] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.