Vilnius University
Institute of Mathematics and Informatics
L I T H U A N I A

INFORMATICS (09 P)

THE RESEARCH OF INTELLIGENT METHODS FOR OPINION MINING IN BIG DATA ARRAYS

Konstantinas Korovkinas

October 2017

Technical Report MII-DS-09P-17-19

VU Institute of Mathematics and Informatics, Akademijos str. 4, Vilnius LT-08663, Lithuania
www.mii.lt

Abstract

This report follows an important problem of sentiment recognition which may influence ones decisions or reviews about item and etc. This report contains introduction, a review of existing techniques and problem domain, methodology of the research and experimental research. A new method is introduced to improve classification performance in sentiment analysis, by combining SVM and Naïve Bayes classification results to recognize positive or negative sentiment, and test in on datasets with simple sentiments from ordinary speech and from movie reviews. This method is evaluated on a training dataset which consists positive and negative words, and hold-out testing dataset, as well as on its complement with additional training data.

Keywords: sentiment analysis, machine learning algorithms, SVM, Naïve Bayes classification

Contents

1 Introduction

The term opinion mining appears in a paper by Dave et al. [DLP03] that was published in the proceedings of the 2003 WWW conference; the publication venue may explain the popularity of the term within communities strongly associated with Web search or information retrieval. According to Dave et al. [DLP03], the ideal opinion-mining tool would "process a set of search results for given item, generating a list of product attributes (quality, features, etc.) and aggregating opinions about each of them (poor, mixed, good)". Much of the subsequent research self-identified as opinion mining fits this description in its emphasis on extracting and analyzing judgements on various aspects of given items, However, the term has recently also been interpreted more broadly to include many different types of analysis of text [Liu07, PL+08].

The history of the phrase sentiment analysis parallels that of "opinion mining" in certain respects. The term "sentiment" used in reference to the automatic analysis of evaluative text and tracking of the predictive judgments therein appears in 2001 papers by Das and Chen [DC01] and Tong [Ton01], due to these authors' interest in analyzing market sentiment. It subsequently occurred within 2002 papers by Turney [Tur02] and Pang et al. [PLV02], which were published in the proceedings of the annual meeting of the Association for Computational Linguistics (ACL) and the annual conference on Empirical Methods in Natural Language Processing (EMNLP). Moreover, Nasukawa and Yi [NY03] entitled their 2003 paper, "Sentiment analysis: Capturing favorability using natural language processing", and a paper in the same year by Yi et al. [YNBN03] was named "Sentiment Analyzer: Extracting sentiments about a given topic using natural language processing techniques". These events together may explain the popularity of "sentiment analysis" among communities self-identified as focused on NLP. A sizeable number of papers mentioning "sentiment analysis" focus on the specific application of classifying reviews as to their polarity (either positive or negative), a fact that appears to have caused some authors to suggest that the phrase refers specifically to this narrowly defined task. However, nowadays many construe the term more broadly to mean the computational treatment of opinion, sentiment, and subjectivity in text [PL+08].

Sentiment analysis, also called opinion mining, is the field of study that analyzes people's opinions, sentiments, appraisals, attitudes, and emotions toward entities and their attributes expressed in written text. The entities can be products,services, organizations, individuals, events, issues, or topics. The field represents a large problem space [Liu15].

Sentiment analysis research has been mainly carried out at three levels of granularity: document level, sentence level, and aspect level [Liu15]. This topic is considered as very challenging – although a lot of work has been done in this field, accuracy is still rather average due to comments, slang, smiles, sarcasm and etc.

2 A review of existing techniques and problem domain

Nowadays the sentiment analysis is one of the most research area. Author [Liu15] seperate the several reasons for it.

First, it has a wide arrange of applications, almost in every domain. The industry surrounding sentiment analysis has also flourished due to the proliferation of commercial applications. This provides a strong motivation for research [Liu15].

Second, it offers many challenging research problems, which had never been studied before. This book will systematically define and discuss these problems, and describe the current state-of-the-art techniques for solving them [Liu15].

Third, for the first time in human history, we now have a huge volume of opinionated data in the social media on the Web. Without this data, a lot of research would not have been possible [Liu15].

In this part are presented a short review of existing researches and techniques, which are done in this area.

2.1 Literature review

Bing Liu [Liu15], Tang et al. [TTC09] expressed an overview in sentiment analysis in which analyzed the strong points and the weak points of sentiment analysis and they gave many research ways of sentiment analysis. Pang et al. [PLV02], Dave et al. [DLP03] think that sentiment classification can be regarded as a binary-classification task. Dave et al. [DLP03] use structured reviews for testing and training, identifying appropriate features and scoring methods from information retrieval for determining whether reviews are positive or negative.

Pang et al. [PL04, PL+08] compared many classifiers on movie reviews and gave a vision of insight and comprehension in sentiment analysis and opinion mining. Authors also used star rating as a feature for classification [LN15]. In another paper Pang et al. [PLV02] evaluated the performance of Naïve Bayes, maximum entropy, and support vector machines in the specific domain of movie reviews, obtaining accuracy slightly above 80%. Go et al. [GBH09] later obtained similar results with unigrams by introducing a more novel approach to automatically classify the sentiment of Twitter messages as either positive or negative with respect to a query term. SVM proved to perform best.

Davidov et al. [DTR10] stated that SVM and Naïve Bayes are best techniques to classify the data and can be regarded as the baseline learning methods, by applying them for analysis based on the Twitter user defined hashtag in tweets. The features were obtained after preprocessing step using the ngrams, punctuation, single words and pattern as different feature types and then combined in to a single feature vector for the classification. K-nearest neighbor strategy was used to assign labels in each training and testing data set.

The ensemble idea in supervised learning has been investigated since the late seventies. Tukey [Tuk77] suggests combining two linear regression models. The first linear

regression model is fitted to the original data and the second linear model to the residuals. Two years later, Dasarathy, Sheela [DS79] suggested to partition the input space using two or more classifiers. The main progress in the field was achieved during the Nineties. Hansen, Salamon [HS90] suggested an ensemble of similarly configured neural networks to improve the predictive performance of a single one. At the same time Schapire [Sch90] laid the foundations for the award winning AdaBoost Freund, Schapire [FS+96] algorithm by showing that a strong classifier in the probably approximately correct (PAC) sense can be generated by combining "weak" classifiers (that is, simple classifiers whose classification performance is only slightly better than random classification). Ensemble methods can also be used for improving the quality and robustness of unsupervised tasks. Ensemble methods can be also used for improving the quality and robustness of clustering algorithms [DWH03], [Rok10a].

Xia et al. [XZL11] used an ensemble framework for sentiment classification. Ensemble framework is obtained by combining various feature sets and classification techniques. In that work, they used two types of feature sets and three base classifiers to form the ensemble framework. Two types of feature sets are created using Part-of-speech information and Word-relations. Naïve Bayes, Maximum Entropy and Support Vector Machines are selected as base classifiers. They applied different ensemble methods like Fixed combination, Weighted combination and Meta-classifier combination for sentiment classification and obtained better accuracy [NR13].

2.2 Ensemble methods overview

The goal of ensemble systems is to create several classifiers with relatively fixed (or similar) bias and then combining their outputs, say by averaging, to reduce the variance [ZM12].

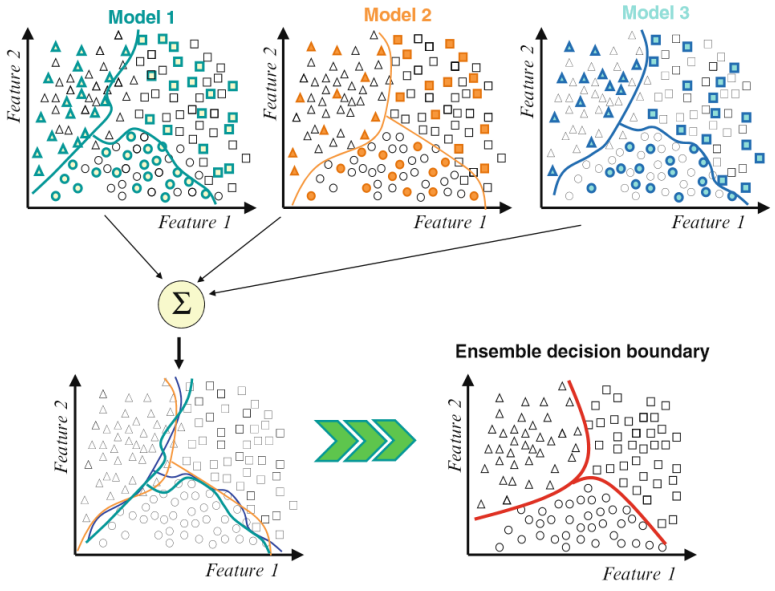

Figure 1: Variability reduction using ensemble systems. Source [ZM12]

The reduction of variability can be thought of as reducing high-frequency (high-variance) noise using a moving average filter, where each sample of the signal is averaged by a neighbor of samples around it. Assuming that noise in each sample is independent, the noise component is averaged out, whereas the information content that is common to all segments of the signal is unaffected by the averaging operation. Increasing classifier accuracy using an ensemble of classifiers works exactly the same way: assuming that classifiers make different errors on each sample, but generally agree on their correct classifications, averaging the classifier outputs reduces the error by averaging out the error components [ZM12].

2.2.1 Boosting

Boosting (also known as arcing — Adaptive Resampling and Combining) is a general method for improving the performance of a weak learner (such as classification rules or decision trees). The method works by repeatedly running a weak learner (such as classification rules or decision trees), on various distributed training data. The classifiers produced by the weak learners are then combined into a single composite strong classifier in order to achieve a higher accuracy than the weak learner's classifiers would have had [Rok10a].

2.2.1.1 AdaBoost

AdaBoost (Adaptive Boosting), which was first introduced in [FS+96], is a popular ensemble algorithm that improves the simple boosting algorithm via an iterative process. The main idea behind this algorithm is to give more focus to patterns that are harder to classify. The amount of focus is quantified by a weight that is assigned to every pattern in the training set. Initially, the same weight is assigned to all the patterns. In each iteration the weights of all misclassified instances are increased while the weights of correctly classified instances are decreased. As a consequence, the weak learner is forced to focus on the difficult instances of the training set by performing additional iterations and creating more classifiers. Furthermore, a weight is assigned to every individual classifier. This weight measures the overall accuracy of the classifier and is a function of the total weight of the correctly classified patterns. Thus, higher weights are given to more accurate classifiers. These weights are used for the classification of new patterns. Mathematically, it can be written as [Rok10a]:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t \cdot M_t(x)\right) \tag{1}$$

2.2.1.2 AdaBoost.M1

Algorithm AdaBoost.M1

Input: sequence of m examples $((x_1, y_1), \ldots, (x_m, y_m))$ with labels $y_i \in Y = \{1, \ldots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i.

Do for $t = 1, 2, \ldots, T$

1. Call **WeakLearn**, providing it with the distribution D_t
2. Get back a hypothesis $h_t : X \to Y$.
3. Calculate the error of $h_t : \epsilon_t = \sum_{i:h_t(x_i) \neq y_i} \cdot D_t(i)$. If $\epsilon_t > 1/2$, then $T = t - 1$ and abort loop.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update distribution $D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
   where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$h_{fin}(x) = arg \max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$$

Figure 2: *The algorithm **AdaBoost.M1*** [FS+96]

The boosting algorithm takes as input a training set of m examples $S = ((x_1, y1), \ldots, (x_m, y_m))$ where x_i is an instance drawn from some space X and represented in some manner (typically, a vector of attribute values), and $y_i \in Y$ is the class label associated with x_i. In addition, the boosting algorithm has access to another unspecified learning algorithm, called the weak learning algorithm, which is denoted generically as **WeakLearn**. The boosting algorithm calls **WeakLearn** repeatedly in a series of rounds. On round t, the booster provides **WeakLearn** with a distribution D_t over the training set S. In response, **WeakLearn** computes a classifier or hypothesis $h_t : X \to Y$ which should misclassify a non trivial fraction of the training examples, relative to D_t. That is, the weak learner's goal is to find a hypothesis h_t which minimizes the (training) error $\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$. Note that this error is measured with respect to the distribution D_t that was provided to the weak learner. This process continues for T rounds, and, at last, the booster combines the weak hypotheses $h_1, \ldots, h_T$ into a single final hypothesis h_{fin} [FS+96].

2.2.1.3 AdaBoost.M2

Algorithm AdaBoost.M2

Input: sequence of m examples $((x_1, y_1), \ldots, (x_m, y_m))$ with labels $y_i \in Y = \{1, \ldots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Let $B = \{(i, y) : i \in \{1, \ldots, m\}, y \neq y_i\}$

Initialize $D_1(i, y) = 1/|B|$ for $(i, y) \in B$.

Do for $t = 1, 2, \ldots, T$

1. Call **WeakLearn**, providing it with mislabel distribution D_t
2. Get back a hypothesis $h_t : X \times Y \to [0, 1]$.
3. Calculate the pseudo-loss of $h_t : \epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update $D_t : D_{t+1}(i, y) = \frac{D_t(i,y)}{Z_t} \cdot \beta_t^{(1/2)(1+h_t(x_i,y_i)-h_t(x_i,y))}$
   where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the hypothesis:

$$h_{fin}(x) = arg \max_{y \in Y} \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t} \right) h_t(x, y)$$

.

Figure 3: *The algorithm **AdaBoost.M2*** [FS+96]

AdaBoost.M2, is based on these ideas, achieves boosting if each weak hypothesis has pseudo-loss slightly better than random guessing. On each round t of boosting, **AdaBoost.M2** supplies the weak learner with a mislabel distribution D_t. In response, the weak learner computes a hypothesis h_t of the form $h_t : X \in Y \to [0, 1]$. The weak learner's goal is to find a weak hypothesis h_t with small pseudo-loss. Thus, standard "off-the-shelf" learning algorithms may need some modification to be used in this manner, although this modification is often straightforward. After receiving h_t, the mislabel distribution is updated using a rule similar to the one used in **AdaBoost.M1**. The final hypothesis h_{fin} outputs, for a given instance x, the label y that maximizes a weighted average of the weak hypothesis values $h_t(x, y)$ [FS+96].

2.3 The Bagging algorithm

The Bagging algorithm
Input: training set S, Inducer I, integer T (number of bootstrap samples).

1. for $i = 1$ to $T\{$
2. $\quad S' =$ bootstrap sample from S (i.i.d sample with replacement).
3. $\quad C_i = I(S')$
4. $\}$
5. $$C^*(x) = arg\max_{y \in Y} \sum_{i:C_i(x)=y} 1 \text{ (the most often predicted label y)}$$

Output classifier C^*.

Figure 4: *The Baggign algorithm* [BK99]

The Bagging algorithm (Bootstrap aggregating) by Breiman [Bre96] votes classifiers generated by different bootstrap samples (replicates). A *Bootstrap sample* [ET93] is generated by uniformly sampling m instances from the training set with replacement. T bootstrap samples $B_1, B_2, \ldots, B_T$ are generated and a classifier C_i is built from each bootstrap sample B_j. A final classifier C^* is built from $C_1, C_2, \ldots, C_T$ whose output is the class predicted most often by its sub-classifiers, with ties broken arbitrarily.
For a given bootstrap sample, an instance in the training set has probability $1-(1-1/m)^m$ of being selected at least once in the m times instances are randomly selected from the training set. For large m, this is about $1 - 1/e = 63.2\%$, which means that each bootstrap sample contains only about 63.2% unique instances from the training set. This perturbation causes different classifiers to be built if the inducer is unstable (e.g., neural networks, decision trees) [B+96] and the performance can improve if the induced classifiers are good and not correlated; however, Bagging may slightly degrade the performance of stable algorithms (e.g., k-nearest neighbor) because effectively smaller training sets are used for training each classifier [Bre96, BK99].

2.3.1 Fusions Methods

Fusing methods aim at providing the classification by combining the out-puts of several classifiers. We assume that the output of each classifier i is a k-long vector $p_{i,1}, \ldots, p_{i,k}$. The value $p_{i,j}$ represents the support that instance x belongs to class j according to the

classifier *i*. For the sake of simplicity, it is also assumed that $\sum_{j=1}^{k} p_{i,j} = 1$. If we are dealing with a crisp classifier *i*, which explicitly assigns the instance *x* to a certain class *l*, then it can still be converted to *k*-long vector $p_{i,1}, \ldots, p_{i,k}$ such that $p_{i,l} = 1$ and $p_{i,j} = 0 \forall j \neq l$. Fusions methods can be furthered partitioned into weighting methodsand meta-learning methods [Rok10b].

2.3.1.1 Weighting Methods

The base members classification are combined using weights that are assigned to each member. The member's weight indicates its effect on the final classification. The assigned weight can be fixed or dynamically determined for the specific instance to be classified. The weighting methods are best suited for problems where the individual classifiers perform the same task and have comparable success or when we would like to avoid problems associated with added learning (such as overfitting or long training time) [Rok10b].

2.3.1.2 Majority Voting

In this combining scheme, a classification of an unlabeled instance is per-formed according to the class that obtains the highest number of votes. This method is also known as the plurality vote (PV)or the basic ensemble method (BEM). This approach has frequently been used as a combining method for comparing newly proposed methods.
Mathematically majority voting can be written as:

$$class(x) = arg \max_{c_i \in dom(y)} \left(\sum g(y_k(x), c_i) \right) \tag{2}$$

where $y_k(x)$ is the classification of the *k*'th classifier and *g(y,c)* is an indicator function defined as:

$$g(y, c) = \begin{cases} 1 & y = c \\ 0 & y \neq c \end{cases} \tag{3}$$

Note that in case of a probabilistic classifier, the crisp classification $y_k(x)$ is usually obtained as follows:

$$y_k(x) = arg \max_{c_i \in dom(y)} \hat{P}_{M_k}(y = c_i | x) \tag{4}$$

where M_k denotes classifier *k* and $\hat{P}_{M_k}(y = C|x)$ denotes the probability of *y* obtaining the value *c* given an instance *x* [Rok10b].

2.3.1.3 Performance Weighting

The weight of each classifier can be set proportional to its accuracy performance on a validation set [OS96].

$$w_i = \frac{(\alpha_i)}{\sum_{j=1}^{T}(\alpha_j)} \tag{5}$$

where α_i is a performance evaluation of classifier i on a validation set. Once the weights for each classifier have been computed, we select the class which receive the highest score:

$$class(x) = arg \max_{c_i \in dom(y)} \left(\sum_k \alpha_i g(y_k(x), c_i) \right) \tag{6}$$

Since the weights are normalized and are summed up to 1, it possible to interpret the sum in last equation as the probability that x_i is classified into c_j [Rok10b].
Moreno-Seco et al. [MSIDLM06] examined several variations of performance weighting methods:
Re-scaled weighted vote The idea is to weight values proportionally to some given ratio *N/M* as following:

$$\alpha_k = \max \left\{ 1 - \frac{M \cdot e_k}{N \cdot (M-1)}, 0 \right\} \tag{7}$$

where e_i is the number of misclassifications made by classifier i [Rok10b].

Best-worst weighted vote The idea is that the best and the worst classifiers obtain the weight of 1 and 0 respectively. The rest of classifiers are rated linearly between these extremes [Rok10b]:

$$\alpha_i = 1 - \frac{e_i - \min_i(e_i)}{\max_i(e_i) - \min_i(e_i)} \tag{8}$$

Quadratic best-worst weighted vote In order to give additional weight to the classifications provided by the most accurate classifiers, the values obtained by the best-worst weighted vote approach are squared [Rok10b]:

$$\alpha_i = \left(\frac{\max_i(e_i) - e_i}{\max_i(e_i) - \min_i(e_i)} \right)^2 \tag{9}$$

2.3.1.4 Distribution Summation

The idea of the distribution summation combining method is to sum up the conditional probability vector obtained from each classifier [CB91]. The selected class is chosen according to the highest value in the total vector. Mathematically, it can be written as [Rok10b]:

$$Class(x) = arg \max_{c_i \in dom(y)} \sum_k \hat{P}_{M_k}(y = c_i | x) \tag{10}$$

2.3.1.5 Bayesian Combination

In the Bayesian combination method the weight associated with each classifier is the posterior probability of the classifier given the training set [Bun90].

$$Class(x) = arg \max_{c_i \in dom(y)} \sum_k P(M_k|S) \cdot \hat{P}_{M_k}(y = c_i|x) \tag{11}$$

where $P(M_k|S)$ denotes the probability that the classifier M_k is correct given the training set S. The estimation of $P(M_k|S)$ depends on the classifier's representation [Rok10b].

2.3.1.6 Dempster-Shafer

The idea of using the Dempster–Shafer theory of evidence [BS+84] for combining classifiers has been suggested in [Shl90]. This method uses the notion of basic probability assignment defined for a certain class c_i given the instance x:

$$bpa(c_i, x) = 1 - \prod_k \left(1 - \hat{P}_{M_k}(y = c_i|x)\right) \tag{12}$$

Consequently, the selected class is the one that maximizes the value of the belief function:

$$Bel(c_i, x) = \frac{1}{A} \cdot \frac{bpa(c_i, x)}{1 - bpa(c_i, x)} \tag{13}$$

where A is a normalization factor defined as [Rok10b]:

$$A = \sum_{\forall c_i \in dom(y)} \frac{bpa(c_i, x)}{1 - bpa(c_i, x)} + 1 \tag{14}$$

2.3.1.7 Vogging

The idea of behind the vogging approach (Variance Optimized Bagging) is to optimize a linear combination of base-classifiers so as to aggressively reduce variance while attempting to preserve a prescribed accuracy [DEYM02]. For this purpose, Derbeko et al. [DEYM02] implemented the Markowitz Mean-Variance Portfolio Theory that is used for generating low variance portfolios of financial assets [Rok10b].

2.3.1.8 Entropy-Weighting

The idea in this combining method is to give each classifier a weight that is inversely proportional to the entropy of its classification vector.

$$Class(x) = arg \max_{c_i \in dom(y)} \sum_{k:c_i = arg \max_{c_j \in dom(y)} \hat{P}_{M_k}(y = c_j|x)} E(M_k, x) \tag{15}$$

where [Rok10b]:

$$E(M_k, x) = -\sum_{c_j} \hat{P}_{M_k}(y = c_j|x) \log\left(\hat{P}_{M_k}(y = c_j|x)\right) \tag{16}$$

2.3.1.9 Density-based Weighting

If the various classifiers were trained using datasets obtained from different regions of the instance space, it might be useful to weight the classifiers according to the probability of sampling x by classifier M_k, namely:

$$Class(x) = arg \max_{c_i \in dom(y)} \sum_{k:c_i = arg \max_{c_j \in dom(y)} \hat{P}_{M_k}(y = c_j|x)} \hat{P}_{M_k}(x) \tag{17}$$

The estimation of $\hat{P}_{M_k}(x)$ depends on the classifier representation and cannot always be estimated [Rok10b].

2.3.1.10 DEA Weighting Method

Recently there has been attempts to use the data envelop analysis (DEA) methodology Charnes et al [CCR78] in order to assign weights to different classifiers [SC01]. These researchers argue that the weights should not be specified according to a single performance measure, but should be based on several performance measures. Because there is a trade-off among the various performance measures, the DEA is employed in order to figure out the set of efficient classifiers. In addition, DEA provides in efficient classifiers with the benchmarking point [Rok10b].

2.3.1.11 Logarithmic Opinion Pool

According to the logarithmic opinion pool [Han00] the selection of the preferred class is performed according to:

$$Class(x) = arg \max_{c_j \in dom(y)} e^{\sum_k = \alpha_k \log\left(\hat{P}_{M_k}(y = c_j|x)\right)} \tag{18}$$

where α_k denotes the weight of the k-th classifier, such that [Rok10b]:

$$\alpha_k \geq 0; \sum \alpha_k = 1 \tag{19}$$

2.3.1.12 Order Statistics

Order statistics can be used to combine classifiers [TG01]. These combiners offer the simplicity of a simple weighted combination method together with the generality of meta-combination methods. The robustness of this method is helpful when there are significant variations among classifiers in some part of the instance space [Rok10b].

2.4 Relevant machine learning algorithms

2.4.1 Naïve Bayes Classification

A Naïve Bayes classifier is a simple probabilistic classifier based on Bayes' theorem and is particularly suited when the dimensionality of the inputs are high. In text classification, the given document is assigned a class

$$C^* = arg\max_c \; p(c|d)$$

Its underlying probability model can be described as an "independent feature model". The Naïve Bayes (NB) classifier uses the Bayes' rule Eq. (20),

$$p(c|d) = \frac{p(c)p(d,c)}{p(d)} \tag{20}$$

Where, *p(d)* plays no role in selecting C^*. To estimate the term *p(d | c)*, Naïve Bayes decomposes it by assuming the f_i*'s* are conditionally independent given *d's* class as in Eq.(21),

$$p_{NB}(c,d) = \frac{p(c)\left(\prod_{i=1}^{m} p(f_i,c)^{n_i(d)}\right)}{p(d)} \tag{21}$$

Where, m is the no of features and f_i is the feature vector. Consider a training method consisting of a relative-frequency estimation *p(c)* and *p* (f_i | *c)* [PLV02].

2.4.2 Support Vector Machines

Support vector machines were introduced in [BGV92] and basically attempt to find the best possible surface to separate positive and negative training samples. Support Vector Machines (SVMs) are supervised learning methods used for classification.

Given training vectors $x_i \in R_n$, $i = 1, \ldots, l$, in two classes, and an indicator vector $y \in R^l$ such that $y_i \in$ {1,-1}, $C - SVC$ [BGV92] solves the following primal optimization problem [CL11].

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \tag{22}$$

subject to $y_i\ (w^T \phi(x_i)$ + b) ≥ 1 - ξ_i, $\xi_i \geq 0$, *i = 1, . . . , l*

where $\phi(x_i)$ maps x_i into a higher-dimensional space and *C* > 0 is the regularization parameter. Due to the possible high dimensionality of the vector variable w, usually we solve the following dual problem.

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \tag{23}$$

subject to $y^T \alpha = 0$, $0 \leq \alpha \leq C$, *i = 1, . . . , l*

where $e = [1, ..., l]^T$ is the vector of all ones, *Q* is an *l* by *l* positive semidefinite matrix,

$Q_{ij} \equiv y_i y_j K(x_i, x_j)$, and $K(x_i, x_j) \equiv \varphi(x_i)^T \varphi(x_j)$ is the kernel function.

After problem (2.5) is solved, using the primal-dual relationship, the optimal w satisfies.

$$w = \sum_{i=1}^{l} y_i \alpha_i \phi(x_i) \tag{24}$$

and the decision function is [CL11]

$$sgn(w^T \phi(x) + b) = sgn\left(\sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b\right) \tag{25}$$

2.4.3 Maximum Entropy

Maximum entropy classification is an alternative technique which has proven effective in a number of natural language processing applications (Berger et al. [BPP96]). Nigam et al. [NLM99] show that it sometimes, but not always, outperforms Naive Bayes at standard text classification. Its estimate of *P(c | d)* takes the following exponential form:

$$P_{ME}(c|d) := \frac{1}{Z(d)} \exp\left(\sum_i \lambda_{i,c} F_{i,c}(d, c)\right) \tag{26}$$

where *Z(d)* is a normalization function. $F_{i,c}$ is a *feature/class* function for feature f_i and class *c*, defined as follows:

$$F_{i,c}(d, c') := \begin{cases} 1, & n_i(d) > 0 \text{ and } c' = c \\ 0 & \text{otherwise.} \end{cases} \tag{27}$$

For instance, a particular feature/class function might fire if and only if the bigram "still hate" appears and the document's sentiment is hypothesized to be negative. Importantly, unlike Naive Bayes, Maximum Entropy makes no assumptions about the relationships between features, and so might potentially perform better when conditional independence assumptions are not met [PLV02].

The $\lambda_{i,c}$'s are feature-weight parameters; inspection of the definition of P_{ME} shows that a large $\lambda_{i,c}$ means that f_i is considered a strong indicator for class *c*. The parameter values are set so as to maximize the entropy of the induced distribution (hence the classifier's name) subject to the constraint that the expected values of the feature/class functions with respect to the model are equal to their expected values with respect to the training data: the underlying philosophy is that we should choose the model making the fewest assumptions about the data while still remaining consistent with it, which makes intuitive sense. Authors [PLV02] use ten iterations of the improved iterative scaling algorithm (Della Pietra et al. [DPDPL97]) for parameter training (this was a sufficient number of iterations for convergence of training-data accuracy), together with a Gaussian prior to prevent overfitting (Chen and Rosenfeld, [CR00]) [PLV02].

2.4.4 Random Forests

Random Forests were introduced by Leo Breiman [Bre01] who was inspired by earlier work by Amit and Geman [AG97].
Random Forest is a tree-based ensemble with each tree depending on a collection of random variables. More formally, for a p-dimensional random vector $X = (X_1, \ldots, X_p)^T$ representing the real-valued input or predictor variables and a random variable Y representing the real-valued response, we assume an unknown joint distribution $P_{XY}(X, Y)$. The goal is to find a prediction function *f(X)* for predicting Y. The prediction function is determined by a loss function *L(Y, f(X))* and defined to minimize the expected value of the loss

$$E_{XY}(L(Y, f(X))) \tag{28}$$

where the subscripts denote expectation with respect to the joint distribution of X and Y [CCS12].
Intuitively, *L(Y, f(X))* is a measure of how close *f(X)* is to Y; it penalizes values of *f(X)* that are a long way from Y. Typical choices of L are *squared error loss* $L(Y, f(X)) = (Y - f(X))^2$ for regression and *zero-one loss* for classification:

$$L(Y, f(X)) = I(Y \neq f(X)) = \begin{cases} 0 & \text{if } Y = f(X) \\ 1 & \text{otherwise.} \end{cases} \tag{29}$$

It turns out that minimizing $E_{XY}(L(Y, f(X)))$ for squared error loss gives the conditional expectation

$$f(x) = E(Y|X = x) \tag{30}$$

otherwise known as the *regression function*. In the classification situation, if the set of possible values of Y is denoted by $\mathcal{Y}$, minimizing $E_{XY}(L(Y, f(X)))$ for zero-one loss gives

$$f(x) = arg \max_{y \in \mathcal{Y}} P(Y = y|X = x) \tag{31}$$

otherwise known as the *Bayes rule* [CCS12]. Ensembles construct f in terms of a collection of so-called "base learners" $h_1(x), \ldots, h_J(x)$ and these base learners are combined to give the "ensemble predictor" *f(x)*. In regression, the base learners are averaged

$$f(x) = \frac{1}{J} \sum_{j=1}^{J} h_j(x) \tag{32}$$

while in classification, *f(x)* is the most frequently predicted class ("voting")

$$f(x) = arg \max_{y \in \mathcal{Y}} \sum_{j=1}^{J} I(y = h_j(x)) \tag{33}$$

In Random Forests the *j*th base learner is a tree denoted $h_j(X, \Theta_j)$, where Θ_j is a collection of random variables and the Θ_j's are independent for $j = 1, \ldots, J$ [CCS12].

Algorithm Random Forests

Let $\mathcal{D} = (x_1, y_1), \ldots, (x_N, y_N)$ denote the training data, with $x_i = (x_{i,1}, \ldots, x_{i,p})^T$. For $j = 1$ to J:

1. Take a bootstrap sample $\mathcal{D}_j$ of size N from $\mathcal{D}_j$.
2. Using the bootstrap sample $\mathcal{D}_j$ as the training data, fit a tree using binary recursive partitioning:
   (a) Start with all observations in a single node.
   (b) Repeat the following steps recursively for each unsplit node until the stopping criterion is met:
      i. Select m predictors at random from the p available predictors.
      ii. Find the best binary split among all binary splits on the m predictors from step i.
      iii. Split the node into two descendant nodes using the split from step ii.

To make a prediction at a new point x,

- $\hat{f}(x) = \frac{1}{J}\sum_{j=1}^{J}\hat{h}_j(x)$ for regression
- $\hat{f}(x) = arg\max_y \sum_{j=1}^{J} I(\hat{h}_j(x) = y)$ for classification

where $\hat{h}_j(x)$ is the prediction of the response variable at x using the jth tree.

Figure 5: *The algorithm* ***Random Forests*** [CCS12]

2.4.5 Particle Swarm Optimization (PSO)

A PSO algorithm maintains a swarm of particles, where each particle represents a potential solution. In analogy with evolutionary computation paradigms, a *swarm* is similar to a population, while a particle is similar to an individual. In simple terms, the particles are "flown" through a multidimensional search space, where the position of each particle is adjusted according to its own experience and that of its neighbors.
Let $x_i(t)$ denote the position of particle i in the search space at time step t; unless otherwise stated, t denotes discrete time steps. The position of the particle is changed by adding a velocity, $v_i(t)$, to the current position, i.e.

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{34}$$

with $x_i(0) \sim U(x_min, x_max)$.
It is the velocity vector that drives the optimization process, and reflects both the experiential knowledge of the particle and socially exchanged information from the particle's neighborhood. The experiential knowledge of a particle is generally referred to as the cognitive component, which is proportional to the distance of the particle from its own best position (referred to as the particle's personal best position) found since the first time step. The socially exchanged information is referred to as the social component of the velocity equation [Eng07].

2.4.5.1 Global Best PSO

For *gbest* PSO, the velocity of particle *i* is calculated as.

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t)\left[\hat{y}_j(t) - x_{ij}(t)\right] \tag{35}$$

where $v_{ij}(t)$ is the velocity of particle *i* in dimension $j = 1, \ldots, n_x$ at time step *t*, $x_{ij}(t)$ is the position of particle *i* in dimension *j* at time step *t*, c_1 and c_2 are positive acceleration constants used to scale the contribution of the cognitive and social components respectively, and $r_{1j}(t)$, $r_{2j}(t) \sim U(0,1)$ are random values in the range [0, 1], sampled from a uniform distribution. These random values introduce a stochastic element to the algorithm.
The personal best position, y_i, associated with particle *i* is the best position the particle has visited since the first time step. Considering minimization problems, the personal best position at the next time step, $t+1$, is calculated as [Eng07].

$$y_i(t+1) = \begin{cases} y_i(t) & \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{if } f(x_i(t+1)) < f(y_i(t)) \end{cases} \tag{36}$$

where $f : \mathbb{R}^{n_x} \to \mathbb{R}$ is the fitness function. As with EAs, the fitness function measures how close the corresponding solution is to the optimum, i.e. the fitness function quantifies the performance, or quality, of a particle (or solution) [Eng07].
The global best position, $\hat{y}(t)$, at time step *t*, is defined as

$$\hat{y}(t) \in \{y_0(t), \ldots, y_{n_s}(t)\} | f(\hat{y}(t)) = \min\{f(y_0(t)), \ldots, f(y_{n_s}(t))\} \tag{37}$$

where n_s is the total number of particles in the swarm. $\hat{y}$ is the best position discovered by any of the particles so far – it is usually calculated as the best personal best position. The global best position can also be selected from the particles of the current swarm, in which case [ZCL+98, Eng07]

$$\hat{y}(t) = \min\{f(x_0(t)), \ldots, f(x_{n_s}(t))\} \tag{38}$$

***gbest* PSO algorithm**
Create and initialize an n_x-dimensional swarm;
repeat
  for *each particle i = 1, . . . , n_s* **do**
    //set the personal best position
    if $f(x_i) < f(y_i)$ **then**
      $y_i = x_i$;
    end
    //set the global best position
    if $f(y_i) < f(\hat{y})$ **then**
      $\hat{y} = y_i$;
    end
  end
  for *each particle i = 1, . . . , n_s* **do**
    update the velocity using equation (2.28);
    update the position using equation (2.27);
  end
until *stopping condition is true*;

Figure 6: *gbest PSO algorithm* [Eng07]

2.4.5.2 Local Best PSO

The velocity is calculated as

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t) \left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t) \left[\hat{y}_{ij}(t) - x_{ij}(t)\right] \tag{39}$$

where $\hat{y}_i j$ is the best position, found by the neighborhood of particle i in dimension j. The local best particle position, $\hat{y}_i$, i.e. the best position found in the neighborhood $\mathcal{N}_i$, is defined as

$$\hat{y}(t+1) \in \{\mathcal{N}_i | f(\hat{y}(t+1)) = \min\{f(x)\}, \forall x \in \mathcal{N}_i\} \tag{40}$$

with the neighborhood defined as

$$\mathcal{N}_i = \{y_{i-n_{\mathcal{N}_i}}(t), y_{i-n_{\mathcal{N}_i}+1}(t), \ldots, y_{i-1}(t), y_i(t), y_{i+1}(t), \ldots, y_{i+n_{\mathcal{N}_i}}(t)\} \tag{41}$$

for neighborhoods of size $n_{\mathcal{N}_i}$. The local best position will also be referred to as the neighborhood best position.
Particles within a neighborhood have no relationship to each other. Selection of neighborhoods is done based on particle indices. However, strategies have been developed where neighborhoods are formed based on spatial similarity [Eng07].

***lbest* PSO algorithm**
Create and initialize an n_x-dimensional swarm;
repeat
  for *each particle* $i = 1, \ldots, n_s$ **do**
    //set the personal best position
    if $f(x_i) < f(y_i)$ **then**
      $y_i = x_i$;
    end
    //set the neighborhood best position
    if $f(y_i) < f(\hat{y}_i)$ **then**
      $\hat{y} = y_i$;
    end
  end
  for *each particle* $i = 1, \ldots, n_s$ **do**
    update the velocity using equation (2.32);
    update the position using equation (2.27);
  end
until *stopping condition is true*;

Figure 7: *lbest PSO algorithm* [Eng07]

2.4.6 Decision Tree

Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node [HPK11].

2.4.6.1 ID3 (Iterative Dichotomiser)

ID3 a decision tree algorithm is developed by J. Ross Quinlan, a researcher in machine learning. **ID3** uses information gain as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. Let node *N* represent or hold the tuples of partition *D*. The attribute with the highest information gain is chosen as the splitting attribute for node *N*. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found [HPK11].

The expected information needed to classify a tuple in D is given by

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i) \tag{42}$$

where p_i is the nonzero probability that an arbitrary tuple in D belongs to class C_i and is estimated by $|C_i, D|/|D|$. A log function to the base 2 is used, because the information is encoded in bits. *Info(D)* is just the average amount of information needed to identify the class label of a tuple in D [HPK11].

Now, suppose we were to partition the tuples in D on some attribute A having ν distinct values, $\{a_1, a_2, ..., a_\nu\}$, as observed from the training data. If A is discrete-valued, these values correspond directly to the ν outcomes of a test on A. Attribute A can be used to split D into ν partitions or subsets, $\{D1, D2, ..., D_\nu\}$, where D_j contains those tuples in D that have outcome a_j of A. These partitions would correspond to the branches grown from node N. Ideally, we would like this partitioning to produce an exact classification of the tuples. That is, we would like for each partition to be pure. However, it is quite likely that the partitions will be impure (e.g., where a partition may contain a collection of tuples from different classes rather than from a single class).

How much more information would we still need (after the partitioning) to arrive at an exact classification? This amount is measured by

$$Info_A(D) = \sum_{j=1}^{\nu} \frac{|D_j|}{|D|} \times Info(D_i) \tag{43}$$

The term $\frac{|Dj|}{|D|}$ acts as the weight of the$j$th partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions.

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is [HPK11],

$$Gain(A) = Info(D) - Info_A(D) \tag{44}$$

2.4.6.2 C4.5

C4.5 also presented by Quilan. **C4.5**, a successor of **ID3**, uses an extension to information gain known as *gain ratio*, which attempts to overcome this bias. It applies a kind of normalization to information gain using a "split information" value defined analogously with *Info(D)* as

$$SplitInfo_A(D) = -\sum_{j=1}^{\nu} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right) \tag{45}$$

This value represents the potential information generated by splitting the training data set, D, into ν partitions, corresponding to the ν outcomes of a test on attribute A. Note

that, for each outcome, it considers the number of tuples having that outcome with respect to the total number of tuples in D. It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning. The gain ratio is defined as [HPK11]

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A D} \tag{46}$$

2.4.6.3 CART

CART - Classification and Regression Trees. The Gini index is used in CART. Using the notation previously described, the Gini index measures the impurity of D, a data partition or set of training tuples, as

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 \tag{47}$$

where p_i is the probability that a tuple in D belongs to class C_i and is estimated by $|Ci,D|/|D|$. The sum is computed over m classes. [HPK11]

The Gini index considers a binary split for each attribute. Let's first consider the case where A is a discrete-valued attribute having v distinct values, $\{a_1, a_2, \ldots, a_\nu\}$, occurring in D. To determine the best binary split on A, we examine all the possible subsets that can be formed using known values of A. Each subset, S_A, can be considered as a binary test for attribute A of the form "$A \in S_A$?" Given a tuple, this test is satisfied if the value of A for the tuple is among the values listed in S_A. If A has ν possible values, then there are 2^ν possible subsets. For example, if income has three possible values, namely *{low, medium, high}*, then the possible subsets are *{low, medium, high}*, *{low, medium}*, *{low, high}*, *{medium, high}*, *{low}*, *{medium}*, *{high}*, and *{}*. We exclude the power set, *{low, medium, high}*, and the empty set from consideration since, conceptually, they do not represent a split. Therefore, there are $(2^\nu - 2)/2$ possible ways to form two partitions of the data, D, based on a binary split on A.

When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on A partitions D into D_1 and D_2, the Gini index of D given that partitioning is [HPK11]

$$Gini(D) = \frac{|D_1|}{D} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \tag{48}$$

The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute A is

$$\Delta Gini(A) = Gini(D) - Gini_A(D) \tag{49}$$

3 Research methodology and tools

The following methods were used in the research: theoretical analysis based on previous works; data collection, preparation, analysis, structuring and grouping; proposed method testing and result analysis.

A free software environment for statistical computing and graphics R ([R C16]) and its package package e1071 ([MDH+17]), were used to implement the algorithms and techniques presented in this research.

3.1 Twitter research review

The main characteristic of the Twitter messages is their length, 140 characters, which determines the text that the users post in the platform. Characteristics of the tweets:

1. The linguistic style of tweets is usually informal, with a lot of abbreviations, idioms, and the use of jargon is very common.

2. The users do not care about the correct use of grammar, which increases the difficulty of carrying out a linguistic analysis.

3. Because the maximum length of a tweet is 140 characters, the users usually refer to the same concept with a large variety of short and irregular forms. This problem is known as data sparsity, and it is a challenge for the sentiment-topic task.

4. The lack of context is a very difficult problem that the SA systems have to deal with.

Since 2009 the Sentiment Analysis research community has started to face the problem of the computational treatment of opinions, sentiments and subjectivity in the short texts of Twitter [MCMVULMR14].
Concerning the study of polarity in Twitter, most experiments assume that tweets are subjective. One of the first studies on the classiffication of polarity in tweets was carried out by Go, Bhayani and Huang [GBH09]. They conducted a supervised classification study on tweets in English. If anything characterizes Twitter, it is the vast amount of information published and the wide variety of topics on which users write. This makes very difficult and expensive the construction and manual tagging of a corpus for the supervised classification of polarity. Thus, the authors use the emoticons that usually appear in tweets to differentiate between positive and negative tweets. The validity of this technique was demonstrated by Read [Rea05]. Through Twitter Search APIs, authors generated a corpus of positive tweets, with positive emoticons ":)", and tweets with negative emoticons ":(". The corpus is used to study which features and which classification algorithm are best for the classification of polarity on Twitter. The algorithms analysed are the same as used by Pang et al. [PLV02], i.e. Support Vector Machine (SVM), Naïve Bayes and maximum entropy. The authors obtained good results with the three algorithms and the different features they tested.

They drew some interesting conclusions, such as that the use of POS-TAGS does not provide valuable information for the classification of polarity on Twitter, that the simple use of unigrams to represent tweets provides very good results, comparable to those obtained in the classification of polarity on long texts and, finally, that the results obtained with unigrams can be slightly improved by the combination of unigrams and bigrams [MCMVULMR14].

One of the problems of Sentiment Analysis in Twitter stressed in Aisopos et al. [APTV12] was the sparsity of the texts due to the large variety of short and irregular forms found in tweets because of the 140-character limit. Saif, He and Alani [SHA12] assessed two methods for solving the sparsity problem. The first method consists of mapping some words to semantic concepts, e.g. Mac, Ipod, Iphone and Ipad match with Apple Product, and then applying an interpolation method. The other proposal is based on the joint sentiment/topic model (JST) (Lin and He [LH09]) that instead of mapping semantic concepts, clusters sentiment concepts. The two different models for representing the tweets are used to train the Naïve Bayes algorithm. The corpus used for the experiment is generated by Go et al. [GBH09]. The first evaluation concludes that the method based on semantic interpolation is the best, but then, with the aim of comparing their results with other works, Saif et al. tested their system with the test set of the corpus. In this case, the JST model performed better than the model proposed by Saif et al. Moreover, the JST model reached better results than those obtained with the same corpus in Go et al. [GBH09] and Speriosu et al. [SSUB11] [MCMVULMR14].

Table 1: Work in Sentiment Analysis in Twitter. *Source [MCMVULMR14], modified by author*

Authors	Objective	Method	Model	Features	Accuracy
Go et al. (2009) [GBH09]	Polarity classification	Supervised	SVM, NB, Maximum Entropy	Unigrams, bigrams Bigrams Unigrams + Bigrams Unigrams + POS	81.3 %–82.2 % 78.8 %–81.6 % 81.6 %–83.0 % 79.9 %–81.9 %
Bifetand Frank (2010) [BF10]	Polarity classification	Supervised, data stream mining methods	Multinomial Naïve Bayes SGD Hoeffding tree	Unigrams	82.45 % 78.55 % 69.36 %
Zhang et al. (2011) [ZGD+11]	Sentiment Analysis	Hybrid	LMS (Method proposed)	Unigrams (negation considered)	88.8 % 91.0 % 88.2 % 81.0 % 78.0 %
Jiang et al. (2011) [JYZ+11]	Subjective classification	Supervised	SVM	Unigrams +Sentiment Lexicon Features +Target-dependent features	61.1 % 63.8 % 68.2 %
	Polarity classification		SVM	Unigrams +Sentiment Lexicon Features +Target-dependent features	78.8 % 84.2 % 85.6 %

4 Methodology of the research

4.1 Implemented method

The proposed methodology is focused on combine SVM and Naive Bayes Classification algorithms to get better results. Below (Fig.3.1) is presented system algorithm which shows the principle of data processing from training data up to obtaining the results.

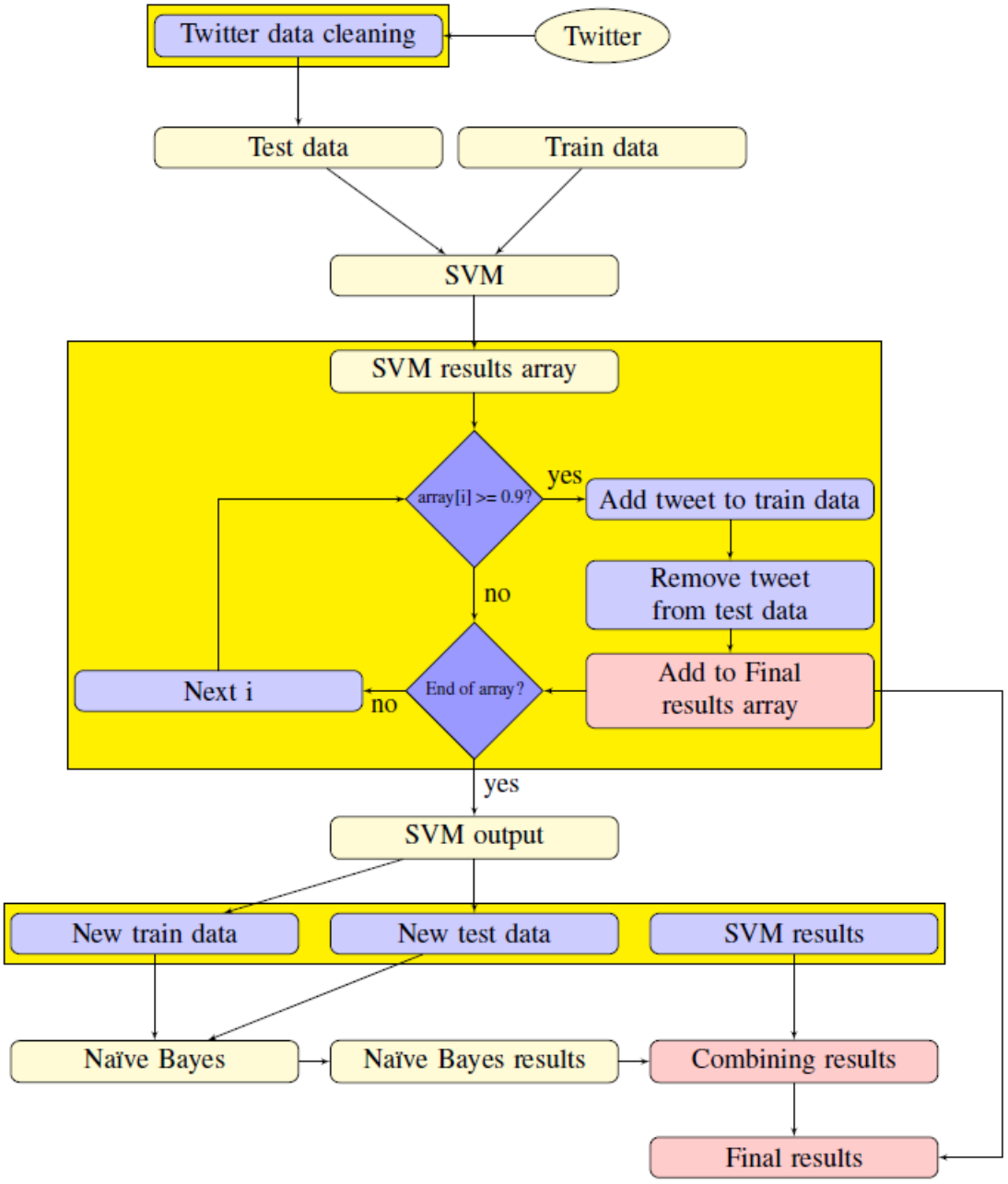

Figure 8: Proposed ensemble method. Created by author

1. Collecting Twitter data. Tweets are collected using created R script. In this script are specified date intervals and searched keyword. After tweets are collected, they are exported to csv file.

2. Twitter data cleaning:

   - Replace negative mentions (don't -> do not and etc.).
   - Replace smiles with meanings (" :)" - happy).
   - Remove repeated letters.
   - Replace acronyms with abbreviations.
   - Remove ampersands.
   - Remove retweet entities.
   - Remove @.
   - Remove punctuation symbols.
   - Remove punctuation numbers.
   - Remove links http.
   - Remove tabs.
   - Remove line breaks.
   - Remove new line characters.
   - Remove trailing and leading.
   - Convert to lowercase.

3. Cleaned tweets are exported to csv file.

4. Trained data are cleaned and labeled tweets, which are downloaded from internet.

5. Train data are passed to SVM for learning and Test data - for results.

6. SVM results array. The results in array are stored as probabilities. If probability is with minus sign - then it is negative sentiment, and if probability is with positive sign - then it is positive sentiment. Each item of results array are checked if it is equal or bigger than 0.9 (absolute values of a numbers are used), it means that this is very strong probability that sentiment are classified correct. Those results directly are storied in result file and tweets are added in train data with label positive or negative and removed from test data. By continuing with each result item - the train data file are increased and the test data file are decreased. Values which no met described requirements are left in SVM result file. After all items of array are checked new train and test data files are presented.

7. New train and new test data are passed to Naïve Bayes algorithm.

8. Combining results are presented in section 4.2.1 (Experimental steps).

4.2 Planning a Research Experiment for implemented method

Research is contained three experiments. All experiments have the same steps, but different datasets.

Experiment No 1.

Dataset is downloaded from *https://github.com/victorneo/Twitter-Sentimental-Analysis*. Files "happy.txt" and "happy_test.txt" are combined into one file "positive.txt" and files "sad.txt" and "sad_test.txt" are combined into one file "negative.txt". Files "positive.txt" and "negative.txt" contain 90 records per file.

Experiment No 2.

Dataset is downloaded from *http://www.cs.cornell.edu/people/pabo/movie-review-data/*. File name "polarity dataset v1.0". File contains 700 positive and 700 negative processed reviews (Released July 2002). Positive reviews are added into file "positive.txt" and negative reviews are added into file "negative.txt"

Experiment No 3.

Third experiment is performed with real twitter data. for training is used dataset downloaded from *https://www.cs.uic.edu/ liub/FBS/sentiment-analysis.html* [LHC05]. It is actually a list of opinion lexicon: a list of English positive and negative opinion words or sentiment words. List of positive words contains 2006 words and negative list contains 4783 words.

The steps of experiment

This steps are similar for all experiments.

1. Files "positive.txt" and "negative.txt" are spitted in training data ("positive_train.txt" and "negative_train.txt") and testing data ("positive_test.txt" and "negative_test.txt"). Train/Test: 50/50; 60/40; 70/30; 80/20.

2. After splitting "positive_train.txt" and "negative_train.txt" are combined into one file "train.txt" and "positive_test.txt" and "negative_test.txt" are combined into one file "test.txt"

3. Three experiments will be performed, results will be compared and accuracy will be calculated.

5 Results

Four experiments were executed to evaluate the performance of proposed techniques:

1. In first experiment we used training dataset, which contains a list of English positive and negative opinion words or sentiment words. List of positive words contains 2006 words and negative list contains 4783 words [LHC05], in total 6789

words. For testing we used dataset which contains 80 happy and 80 sad emotions, resulting in total 160 sentences. These sentences were split into words and used as input for machine learning algorithms.

2. In second experiment we used the same training dataset as in first experiment, with the testing dataset including 700 positive and 700 negative movie reviews (1400 movie reviews in total). These training dataset were also split into words before using as machine learning algorithms input.

3. In third experiment we used the same training dataset as in experiments above, but additionally 60% data was added from movie review dataset and we left 40% data for testing. Training dataset contains 7629 words and reviews, and testing dataset contains 560 movie reviews.

4. Finally, in the last experiment we used the same training and testing datasets, but changed the proportion of training-testing dataset split to 80% and 20%, respectively. The final training dataset contains 7909 words and reviews, while testing dataset contains 280 movie reviews.

Table 2: Results. *Creatied by author*

Exp. No.	Training features	Training dataset	Testing features	Testing dataset	SVM results	Naïve Bayes	New method
1	words	6789	sentences	160	85.63%	83.13%	89.38%
2	words	6789	sentences	1400	68.21%	68.86%	69.43%
3	words+ sentences	7629	sentences	560	75.71%	52.68%	74.46%
4	words sentences	7909	sentences	280	77.86%	53.57%	78.21%

The table 4.1 show that the best results we got when using a list of English positive and negative opinion words or sentiment words for recognize sentiments in ordinary speech. Our introduced method gave results 89,38%. When we use this above described dataset for recognize movie reviews sentiments, we got better results 69,43% than SVM and Naive Bayes, but they are lower to compare with previous experiment results. In third experiment SVM shown the better results than our introduced method, when was added 60% of movie review dataset for training to recognize the remaining 40%. Our introduced method shown 78,21% in the last experiment, when was added 80% of movie review dataset for training to recognize the remaining 20%. As we can see the best results are when testing dataset is the smallest and the lowest (except Naive Bayes) - when the testing dataset is the biggest. Also results shown that the training dataset should be from the same area like the testing dataset. Naive Bayes classification with default parameters

are not very good for recognize sentiments from whole sentence if we don't split it in the words.

6 Conclusions

In this section, we compared two supervised machine learning algorithms of SVM and Naive Bayes classification with our introduced method for the ordinary speech and movie reviews sentiment recognizing. SVM and Naive Bayes classification were used with their default parameters. The experimental results show that our introduced method gave us higher accuracy 89,38% when we use ordinary speech testing dataset than SVM and Naive Bayes classification. However for the movie reviews, the accuracy are lower but better than both others alghorithms, except third experiment, where SVM gave better accuracy 75,71%. We have found out that it is very important to use testing data which are from the same area like training data, as we can see in first experiment.

References

[AG97] Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997.

[APTV12] Fotis Aisopos, George Papadakis, Konstantinos Tserpes, and Theodora Varvarigou. Content vs. context for sentiment analysis: a comparative analysis over microblogs. In *Proceedings of the 23rd ACM conference on Hypertext and social media*, pages 187–196. ACM, 2012.

[B+96] Leo Breiman et al. Heuristics of instability and stabilization in model selection. *The annals of statistics*, 24(6):2350–2383, 1996.

[BF10] Albert Bifet and Eibe Frank. Sentiment knowledge discovery in twitter streaming data. In *International conference on discovery science*, pages 1–15. Springer, 2010.

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[BK99] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1):105–139, 1999.

[BPP96] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[Bre96] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[BS+84] Bruce G Buchanan, Edward Hance Shortliffe, et al. *Rule-based expert systems*, volume 3. Addison-Wesley Reading, MA, 1984.

[Bun90] Wray Buntine. A theory of learning classification rules. In *Doctoral dissertation.* School of Computing Science, University of Technology. Sydney. Australia, 1990.

[CB91] Peter Clark and Robin Boswell. Rule induction with cn2: Some recent improvements. In *European Working Session on Learning*, pages 151–163. Springer, 1991.

[CCR78] Abraham Charnes, William W Cooper, and Edwardo Rhodes. Measuring the efficiency of decision making units. *European journal of operational research*, 2(6):429–444, 1978.

[CCS12] Adele Cutler, D Richard Cutler, and John R Stevens. Random forests. In *Ensemble machine learning*, pages 157–175. Springer, 2012.

[CL11] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

[CR00] Stanley F Chen and Ronald Rosenfeld. A survey of smoothing techniques for me models. *IEEE transactions on Speech and Audio Processing*, 8(1):37–50, 2000.

[DC01] Sanjiv Das and Mike Chen. Yahoo! for amazon: Extracting market sentiment from stock message boards. In *Proceedings of the Asia Pacific finance association annual conference (APFA)*, volume 35, page 43. Bangkok, Thailand, 2001.

[DEYM02] Philip Derbeko, Ran El-Yaniv, and Ron Meir. Variance optimized bagging. In *European Conference on Machine Learning*, pages 60–72. Springer, 2002.

[DLP03] Kushal Dave, Steve Lawrence, and David M Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528. ACM, 2003.

[DPDPL97] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE transactions on pattern analysis and machine intelligence*, 19(4):380–393, 1997.

[DS79] Belur V Dasarathy and Belur V Sheela. A composite classifier system design: concepts and methodology. *Proceedings of the IEEE*, 67(5):708–713, 1979.

[DTR10] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd international conference on computational linguistics: posters*, pages 241–249. Association for Computational Linguistics, 2010.

[DWH03] Evgenia Dimitriadou, Andreas Weingessel, and Kurt Hornik. A cluster ensembles framework, design and application of hybrid intelligent systems, 2003.

[Eng07] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[ET93] Bradley Efron and Robert J Tibshirani. An introduction to the bootstrap: Monographs on statistics and applied probability, vol. 57. *New York and London: Chapman and Hall/CRC*, 1993.

[FS+96] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156, 1996.

[GBH09] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12), 2009.

[Han00] Jakob Vogdrup Hansen. *Combining predictors: Meta machine learning methods and bias/variance & ambiguity decompositions*. PhD thesis, Aarhus University, Computer Science Department, 2000.

[HPK11] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[HS90] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

[JYZ+11] 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. Association for Computational Linguistics, 2011.

[LH09] Chenghua Lin and Yulan He. Joint sentiment/topic model for sentiment analysis. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 375–384. ACM, 2009.

[LHC05] Bing Liu, Minqing Hu, and Junsheng Cheng. Opinion observer: analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web*, pages 342–351. ACM, 2005.

[Liu07] Bing Liu. *Web data mining: exploring hyperlinks, contents, and usage data*. Springer Science & Business Media, 2007.

[Liu15] Bing Liu. *Sentiment analysis: Mining opinions, sentiments, and emotions*. Cambridge University Press, 2015.

[LN15] Bac Le and Huy Nguyen. Twitter sentiment analysis using machine learning techniques. In *Advanced Computational Methods for Knowledge Engineering*, pages 279–289. Springer, 2015.

[MCMVULMR14] Eugenio Martínez-Cámara, M Teresa Martín-Valdivia, L Alfonso Urena-López, and A Rturo Montejo-Ráez. Sentiment analysis in twitter. *Natural Language Engineering*, 20(1):1–28, 2014.

[MDH+17] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2017. R package version 1.6-8.

[MSIDLM06] Francisco Moreno-Seco, José M Inesta, Pedro J Ponce De León, and Luisa Micó. Comparison of classifier fusion methods for classification in pattern recognition tasks. In *SSPR/SPR*, pages 705–713. Springer, 2006.

[NLM99] Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67, 1999.

[NR13] MS Neethu and R Rajasree. Sentiment analysis in twitter using machine learning techniques. In *Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on*, pages 1–5. IEEE, 2013.

[NY03] Tetsuya Nasukawa and Jeonghee Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 70–77. ACM, 2003.

[OS96] David W Opitz and Jude W Shavlik. Generating accurate and diverse members of a neural-network ensemble. In *Advances in neural information processing systems*, pages 535–541, 1996.

[PL04] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

[PL+08] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

[PLV02] 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.

[R C16] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[Rea05] Jonathon Read. Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL student research workshop*, pages 43–48. Association for Computational Linguistics, 2005.

[Rok10a] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.

[Rok10b] Lior Rokach. *Pattern classification using ensemble methods*, volume 75. World Scientific, 2010.

[SC01] So Young Sohn and Hong Choi. Ensemble based on data envelopment analysis. In *ECML Meta Learning workshop*, volume 53, 2001.

[Sch90] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[SHA12] Hassan Saif, Yulan He, and Harith Alani. Alleviating data sparsity for twitter sentiment analysis. CEUR Workshop Proceedings (CEUR-WS. org), 2012.

[Shl90] Seymour Shlien. Multiple binary decision tree classifiers. *Pattern Recognition*, 23(7):757–763, 1990.

[SSUB11] Michael Speriosu, Nikita Sudan, Sid Upadhyay, and Jason Baldridge. Twitter polarity classification with label propagation over lexical links and the follower graph. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 53–63. Association for Computational Linguistics, 2011.

[TG01] Kagan Tumer and Joydeep Ghosh. Robust order statistics based ensembles for distributed data mining. Technical report, TEXAS UNIV AT AUSTIN DEPT OF ELECTRICALAND COMPUTER ENGINEERING, 2001.

[Ton01] Richard M Tong. An operational system for detecting and tracking opinions in on-line discussion. In *Working Notes of the ACM SIGIR 2001 Workshop on Operational Text Classification*, volume 1, page 6, 2001.

[TTC09] Huifeng Tang, Songbo Tan, and Xueqi Cheng. A survey on sentiment detection of reviews. *Expert Systems with Applications*, 36(7):10760–10773, 2009.

[Tuk77] John W Tukey. Exploratory data analysis. 1977.

[Tur02] Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[XZL11] Rui Xia, Chengqing Zong, and Shoushan Li. Ensemble of feature sets and classification algorithms for sentiment classification. *Information Sciences*, 181(6):1138–1152, 2011.

[YNBN03] Jeonghee Yi, Tetsuya Nasukawa, Razvan Bunescu, and Wayne Niblack. Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 427–434. IEEE, 2003.

[ZCL+98] He Zhenya, Wei Chengjian, Yang Luxi, Gao Xiqi, Yao Susu, Russell C Eberhart, and Yuhui Shi. Extracting rules from fuzzy neural network by particle swarm optimisation. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 74–77. IEEE, 1998.

[ZGD+11] L Zhang, R. Ghosh, M. Dekhil, M. Hsu, and B Liu. Combining lexicon-based and learning-based methods for twitter sentiment analysis. In *Technical Report HPL-2011-89*, 2011.

[ZM12] Cha Zhang and Yunqian Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.