Vilnius University
Institute of Data Science and
Digital Technologies
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 2018

Technical Report MII-DS-09P-18-19

VU Institute of Data Science and Digital Technologies, 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 main goal of research related to sentiment analysis (abbr. as SA) is to obtain authors feelings expressed in positive or negative comments. This analysis is performed in multiple levels: document, sentence, word/term or aspect. According to Pang and Lee in [PL+08] the term "sentiment" appears in 2001 papers [DC01, Ton01] and subsequently in 2002 papers [Tur02, PLV02]. Opinion mining is another term in certain respects parallels to sentiment analysis, appeared in 2003 paper by Dave et. al. [DLP03]. They described the ideal opinion-mining tool like "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)" [DLP03].

Support vector machine (abbr. as SVM) were introduced in papers [BGV92, CV95]. According to a number of authors, who worked with SVM, this method proved its efficiency to solve difficult tasks in various domains: for recognition of regulatory DNA sequences (Damaševičius in [Dam10]), for EEG Data classification (Martisius et al. [MDJB12]), for classification of images ([TKF15]), for credit risk evaluation (Danenas and Garsva in [DG15]), for Sensor Multifault Diagnosis (Deng et al. [DGZC17]), for monitoring metal-oxide surge arrester conditions (Hoang et al. in [HCAV18]), for Multi-class parkinsonian disorders classification (Morisi et al. in [MMG+18]), for Forecasting Stock Market Movement Direction (Ren et al. in [RWL18]), for Sentiment Analysis (Liu and Lee in [LL18], Chen and Zhang in [CZ18]) and etc.

1.1 Research problem

Sentiment analysis from text is considered as very challenging area – although a lot of work has been done in this field, accuracy is still rather average due to comments, slang, smiles and etc. To increase accuracy, researchers use natural language processing (abbr. as NLP), various text analyzing techniques, combinations of different machine learning algorithms. When it goes to a large volume of data – big data text arrays, SVM performance decreases dependently on dataset size – the higher number of features is, the longer computation time it requires. There have been a number of efforts to speed up SVM including implementation on Graphics Processor Unit, using cloud computing technology, selecting only representative data for training.

Aforementioned problems led to development a method for sentiment polarity prediction.

1.2 The object of research

The main object of this research is to propose a hybrid K-means and SVM method for sentiment polarity classification in large scale text arrays.

1.3 The goal and objectives of the research

The aim of the research is to propose an approach to hybrid K-means and SVM method for sentiment polarity prediction in large scale text arrays.

The objectives of the dissertation:

1. Investigate related works in sentiment analysis and identify their advantages.
2. Analyze machine learning algorithms for sentiment analysis.
3. Propose hybrid sentiment polarity classification method for researched problem.
4. To perform experimental evaluation of developed techniques.

1.4 Research methodology and tools

The following methods were used: formulation of research tasks and goals; related works analysis; data collection, cleaning, preparation; comparison of techniques; experimental research; results analysis, presentation, comparison; formulation of conclusions.

For hybrid method development and performing experiments were used: Python programming language, scikit-learn [PVG+11]: library for machine learning. For graphical results presentation Matplotlib [Hun07]: 2D graphics environment. For preparing dissertation, diagrams were used LaTeX[2] – A document preparation system.

Experiments were performed with existing datasets: The Stanford Twitter sentiment corpus[3] and Amazon customer reviews dataset[4].

For experiments is used computer with processor Intel(R) Core(TM) i7-4712MQ CPU @ 2.30 GHz and 8.00 GB installed memory (RAM).

1.5 The statements of the thesis

1. Proposed hybrid K-means and SVM method can be applied on a small and huge datasets.
2. Small representative dataset for training can obtain the same or better results than the huge.
3. SVM is very sensitive for parameters tuning. The properly tuned hyperplane can perform better results.
4. Method can be applied for different domains.

1.6 Scientific novelty and practical significance

In this dissertation is proposed hybrid sentiment analysis method. Differently from other authors work, this method consists from two main parts: training data creation and SVM spead up. First part includes: cluster selection algorithm; features retrieving from text; dataset preparation, using various techniques. In another part of method presented SVM speed up technique and hyperplane tuning. Proposed method can be applied in different domains to predict sentiments from text.

Since sentiment analysis is still very challenging area and at the same time very useful, the proposed hybrid K-means and SVM method for sentiment polarity classification in large scale text arrays can be used in new models development or for improving existing. The sentiment analysis is widely used for product reviews, customer churn prediction, fraud detection, president election and etc.

[2]`https://www.latex-project.org/`

[3]`http://help.sentiment140.com/home`

[4]`https://www.kaggle.com/bittlingmayer/amazonreviews/`

1.7 Presentation and approbation of results

Conferences

1. Information Technologies (IT2018), The 23th Conference for Master and PhD students, Kaunas, Lithuania, 2018.
2. The Symposium for Young Scientists in Technology, Engineering and Mathematics (SYSTEM2018), The 23th Conference for Master and PhD students, Gliwice, Poland, 2018.
3. The 24th International Conference on Information and Software Technologies (ICIST 2018), Kaunas, Lithuania, 2018.

Publications

International journals, which are included in Scientific Master Journal List (ISI):

1. Korovkinas, K., Danėnas, P., Garšva, G., 2017. SVM and Naïve Bayes Classification Ensemble Method for Sentiment Analysis. Baltic Journal of Modern Computing, 5(4), pp. 398–409.

Proceedings of scientific conferences, indexed in Scientific Master Journal Proceeding List (ISI):

1. Korovkinas, K., Danėnas, P., Garšva, G., 2018. SVM accuracy and training speed trade-off in sentiment analysis tasks. In International Conference on Information and Software Technologies (pp. 227-239). Springer, Cham.

Proceedings of other conferences:

1. Korovkinas, K., Garšva, G., 2018. Selection of intelligent algorithms for sentiment classification method creation. Proceedings of the International Conference on Information Technologies, Vol–2145, Kaunas, Lithuania, pp. 152–157, ISSN 1613-0073, CEUR. Available: http://ceur-ws.org/Vol-2145/p26.pdf
2. Vaitonis, M., Masteika, S., Korovkinas, K. 2018. Algorithmic trading and machine learning based on GPU. Proceedings of the Symposium for Young Scientists in Technology, Engineering and Mathematics, Vol–2147, Gliwice, Poland, pp. 49–54, ISSN 1613-0073, CEUR. Available: http://ceur-ws.org/Vol-2147/p09.pdf

1.8 Thesis structure

The thesis contains introduction, 4 chapters, conclusion and list of references. The total volume of the dissertation is ?? pages. The list of references contains ?? various sources, including books, scientific papers, technical reports, Internet sources, patents. The work consists of five main parts: introduction, analytical, methodological, experimental and conclusions.

In introduction section are presented the research problem and object, the goal and objectives, methodology and tools, the statements, scientific novelty, practical significance and the lists of publications, where results were published.

Analytical section (review of existing techniques and problem domain) contains an overview of works in sentiment analysis; definitions and comparison of machine learning techniques; conclusion of the section.

2 A review of existing techniques and problem domain

2.1 Sentiment analysis

Sentiment analysis became very popular since people start using Internet, to be more concrete when appeared e-shops, social networks, Blogs and etc., where people can write their comments. Nowadays if you want to get opinion about hotel, movie, book, commodity, game, restaurant and so on, you can find all information what you need in Internet. But these tasks are not so easy, because of huge data amount. Then there is a need in application which can do it instead of us and give the final result.

Liu in [Liu15] gives a definition of sentiment analysis. He described it as “the field of study that analyzes people’s opinions, sentiments, appraisals, attitudes, and emotions toward entities and their attributes expressed in written text.”. According to him this field represents a huge problem space due “many related names and slightly different tasks”. “Sentiment analysis, opinion mining, opinion analysis, opinion extraction, sentiment mining, subjectivity analysis, affect analysis,emotion analysis, and review mining”, according to Liu are “all under the umbrella of sentiment analysis.”.

Basically sentiment analysis is divided into lexicon-based methods and machine learning methods [MF11]. For better results, authors also mixed aforementioned methods. Mainly SA tasks have two approaches: feature extraction and sentiment classification (prediction). In the further sections are presented an overview of works in this field.

2.1.1 Machine learning in sentiment analysis

Traditionally, sentiment classification can be regarded as a binary-classification task (Pang et al. in [PLV02]; Dave et al. in [DLP03]). According to Pang et al. in [PLV02] sentiment analysis is the extraction of positive or negative opinions from text. Dave et al. in [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. These results perform as well as traditional machine learning method then use the classifier to identify and classify review sentences from the web, where classification is more difficult (Khainar and Kinikar in [KK13]). Authors in [PL+08, TTC09, Liu15] 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.

Machine learning algorithms are one part of sentiment analysis. A lot of works are done and many classifiers are compared for SA tasks and many authors conclude that SVM proved to perform best. Pang et al. in [PLV02] evaluated the performance of Naïve Bayes, maximum entropy (abbr. as MaxEnt), and support vector machines in the specific domain of movie reviews, obtaining accuracy slightly above 80%. Go et al. in [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. The same techniques were also used in Kharde and Sonawane in [KS+16] to perform sentiment analysis on Twitter data, yet resulting in lower accuracy; again, SVM proved to perform best. Davidov et al. [DTR10] also 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. Kapočiut et al. in [KKK+13] used knowledge-based and machine learning approaches for sentiment classification into positive, negative and neutral on Lithuanian internet comments. Support Vector Machine and Naïve Bayes Multinomial significantly outperform their proposed knowledge-based method. Le and Nguyen in [LN15] proposed a sentiment analysis model based on Naïve Bayes and Support Vector Machine, for feature extraction they applied Information Gain, Bigram, Object-oriented extraction method in purpose to analyze sentiment more effectively. Gautam and Yadav in [GY14] applied Naïve Bayes, Maximum entropy and SVM along with the Semantic analysis for classifying the sentence and product reviews based on twitter data into positive and negative. Naïve Bayes shown the better accuracy 88.2% than SVM (85.5%) and Maximum entropy (83.8%). After Semantic analysis was applied, they improved accuracy from 88.2% to 89.9%. Kolchyna et al. in [KSTA15] presented a new ensemble method that uses a lexicon based sentiment score as input feature for the machine learning approach and applied it on the benchmark Twitter dataset using three machine learning algorithms: SVM, Decision trees (abbr. as DT) and Naïve Bayes. Results show that when was used only N-grams as the features, SVM achieved accuracy 86.62%, NB - 81.5% and DT - 80.57%. After a new method was applied results increased as follow: SVM - to 91.17%, Decision tree - to 89.9% and NB - to 88.54%. Kanakaraj et al. in [KG15] also presented the semantics based feature vector with ensemble classifier for Twitter data sentiment analysis. The new method was compared with widely used machine learning algorithms like Naïve Bayes, Maximum Entropy, SVM, Decision Tree, Random Forest (abbr. as RF), Extremely Randomized Trees and Decision Tree regression with Ada Boost. Proposed method outperformed single machine learning classifier by 3-5%. Wan and Gao in [WG15] used Naïve Bayes, SVM, Bayesian Network, C4.5 Decision Tree and Random Forest algorithms for creation ensemble method based on Majority Vote principle of multiple classification methods and applied it for sentiment classification on Twitter Data for Airline Services. Amolik et al. in [AJBV16] achieved 75% accuracy with SVM and 65% with Naïve Bayes on Twitter sentiment analysis of Movie reviews classifying tweets as positive, negative and neutral. Tripathy et al. in [TAR16] applied Naive Bayes, Maximum Entropy, Stochastic Gradient Descent and Support Vector Machine on Movie review dataset, using n-gram approach and various combination of it for sentiment classification into positive or negative. The best results were obtained when used: "Unigram+Bigram+Trigram" with SVM 88.94% or Naïve Bayes 86.23%; "Unigram+Bigram" with SVM 88.88% or Maximum entropy 88.42%; "Unigram" with Maximum entropy 88.48% or SVM 86.97%. Authors in [BDDN14, MHK14, AAMA17] did a review on machine learning techniques for sentiment analysis. Well known techniques like Maximum Entropy, SailAil Sentiment Analyzer, Multilayer Perceptron, Naïve Bayes, Multinomial Naïve Bayes, Support Vector Machine, Random forest were discussed and compared accuracy on different datasets. Pranckevičius and Marcinkevičius in [PM17] did investigation on Naïve Bayes, Random Forest, Decision Tree, Support Vector Machines, and Logistic Regression (abbr. as LR) classifiers implemented in Apache Spark and identify the optimal number of n-grams to get the best accuracy. Technique applied on Amazon customers' product-review data for Android Apps. Rathor et al. in [RAD18] also applied Support Vector Machines, Naïve Bayes and Maximum Entropy for classification Amazon reviews into positive, neutral and negative. Naïve Bayes shown the bests results with Unigrams - 66.84%, however SVM provide better with Weighted Unigrams - 81.20% . Manikandan and Sivakumar in [MS18] provided a review of the principles, advantages and applications of document classification, Document clustering and text mining, focusing on the existing literature. According approaches on machine learning the common algorithms for text classification are: Naïve Bayes Classifier, Support Vector Machine Learning, Decision Tree, Rocchio's Algorithm, K-Nearest Neighbor (K-NN), Decision Rules Classification, Artificial Neural Network, Fuzzy correlation and Genetic Algorithm. They concluded that SVM, NB, kNN and their hybrid system with the combination of different other algorithms are shown most appropriate.

Literature review led to conclusions that Naïve Bayes, Random Forest, Decision Tree, Support Vector Machines and Maximum Entrophy (aka Logistic Regression) are still widely used Machine learning algorithms.

2.2 Machine learning algorithms

2.2.0.1 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. (2.1),

$$p(c|d) = \frac{p(c)p(d,c)}{p(d)} \tag{1}$$

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.(2.2),

$$p_{NB}(c,d) = \frac{p(c)\left(\prod_{i=1}^{m} p(f_i,c)^{n_i(d)}\right)}{p(d)} \tag{2}$$

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* $(f_i|c)$ (Pang et al. in [PLV02]).

2.2.1 Multinomial Naïve Bayes

Multinomial Naïve Bayes, presented by Pedregosa et al. in [PVG+11] implements the Naïve Bayes algorithm for multinomially distributed data, and is one of the two classic Naïve Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all features for class y.

The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing [PVG+11].

2.2.2 Support Vector Machines

Support vector machines were introduced in [BGV92,CV95] and basically attempt to find the best possible surface to separate positive and negative training samples in supervised manner.

2.2.3 *C*-Support Vector Classification

Given training vectors $x_i \in R_n$, $i = 1, \dots, l$, in two classes, and an indicator vector $y \in R^l$ such that $y_i \in \{1,-1\}$, $C - SVC$ (Boser et al. in [BGV92]; Cortes and Vapnik, in [CV95]) solves the following primal optimization problem (Chang et al. in [CL11]).

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \tag{3}$$

$$\text{subject to } \; y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i,$$

$$\xi_i \geq 0, \;\; i = 1, \dots, l$$

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.

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \tag{4}$$

$$\text{subject to } \; y^T \alpha = 0,$$

$$0 \leq \alpha \leq C, \;\; i = 1, \dots, l$$

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 $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ is the kernel function.

After problem (2.4) is solved, using the primal-dual relationship, the optimal w satisfies.

$$w = \sum_{i=1}^{l} y_i \alpha_i \phi(x_i) \tag{5}$$

and the decision function is

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

(Chang and Lin in [CL11])

2.2.4 Linear SVM

Linear SVM (Fan et al. in [FCH+08]) is optimized for large-scale learning.

Given training vectors $x_i \in R^n$, $i = 1, \dots, l$ in two class, and a vector $y \in R^l$ such that $y_i = \{1,-1\}$, a linear classifier generates a weight vector w as the model. The decision function is

$$sgn(w^T x)$$

$L2$-regularized $L1$-loss SVC solves the following primal problem:

$$\min_w \frac{1}{2}w^T w + C\sum_{i=1}^{l}(\max(0, 1 - y_i w^T x_i))$$

whereas L2-regularized L2-loss SVC solves the following primal problem:

$$\min_w \frac{1}{2}w^T w + C\sum_{i=1}^{l}(\max(0, 1 - y_i w^T x_i))^2 \tag{6}$$

Their dual forms are:

$$\min_\alpha \frac{1}{2}\alpha^T \bar{Q}\alpha - e^T\alpha$$
$$\text{subject to } 0 \le \alpha_i \le U, \quad i = 1, \dots, l$$

where e is the vector of all ones, $\bar{Q} = Q + D$, D is a diagonal matrix, and $Q_{ij} = y_i y_j x_i^T x_j$. For $L1$-loss SVC, $U = C$ and $D_{ii} = 0, \forall i$. For $L2$-loss SVC, $U = \infty$ and $D_{ii} = 1/(2C), \forall i$.

$L1$ regularization generates a sparse solution w. $L1$-regularized $L2$-loss SVC solves the following primal problem:

$$\min_w \|w\|_1 + C\sum_{i=1}^{l}(\max(0, 1 - y_i w^T x_i))^2 \tag{7}$$

where $\|\cdot\|_1$ denotes the 1-norm. (Fan et al. in [FCH+08])

2.2.5 Logistic Regression

The logistic regression model arises from the desire to model the posterior probabilities of the K classes via linear functions in x, while at the same time ensuring that they sum to one and remain in [0, 1]. The model has the form

$$\begin{aligned}
\log\frac{Pr(G = 1|X = x)}{Pr(G = K|X = x)} &= \beta_{10} + \beta_1^T x \\
\log\frac{Pr(G = 2|X = x)}{Pr(G = K|X = x)} &= \beta_{20} + \beta_2^T x \\
&\vdots \\
\log\frac{Pr(G = K-1|X = x)}{Pr(G = K|X = x)} &= \beta_{(K-1)0} + \beta_{K-1}^T x
\end{aligned} \tag{8}$$

The model is specified in terms of $K - 1$ log-odds or logit transformations (reflecting the constraint that the probabilities sum to one). Although the model uses the last class as the denominator in the odds-ratios, the choice of denominator is arbitrary in that the estimates are equivariant under this choice. A simple calculation shows that

$$\begin{aligned} Pr(G = k|X = x) &= \frac{\exp(\beta_{k0} + \beta_k^T x)}{1 + \sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x)}, \\ k &= 1, \dots, K-1, \\ Pr(G = K|X = x) &= \frac{1}{1 + \sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x)}, \end{aligned} \tag{9}$$

and they clearly sum to one. To emphasize the dependence on the entire parameter set $\theta = \{\beta_{10}, \beta_1^T, \dots, \beta_{(K-1)0}, \beta_{K-1}^T\}$, we denote the probabilities $Pr(G = k|X = x) = p_k(x; \theta)$ (Trevor et al. in [TRJ09]).

2.2.6 Random Forests

Random Forests were introduced by Leo Breiman in [Bre01] who was inspired by earlier work by Amit and Geman in [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 $X = (X_1, \dots, X_p)^T$ 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

$$E_{XY}(L(Y, f(X))) \tag{10}$$

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:

$$L(Y, f(X)) = I(Y \neq f(X)) = \begin{cases} 0 & \text{if } Y = f(X) \\ 1 & \text{otherwise.} \end{cases} \tag{11}$$

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

$$f(x) = E(Y|X = x) \tag{12}$$

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

$$f(x) = arg \max_{y \in \mathcal{Y}} P(Y = y|X = x) \tag{13}$$

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

$$f(x) = \frac{1}{J} \sum_{j=1}^{J} h_j(x) \tag{14}$$

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

$$f(x) = arg \max_{y \in \mathcal{Y}} \sum_{j=1}^{J} I(y = h_j(x)) \tag{15}$$

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 $j = 1, \ldots, J$ (Cutler et al. in [CCS12]).

2.2.7 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].

2.2.7.1 ID3 (Iterative Dichotomiser)

ID3 a decision tree algorithm is developed by J. Ross Quinlan, a researcher in machine learning. It 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

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i) \tag{16}$$

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

$$Info_A(D) = \sum_{j=1}^{\nu} \frac{|D_j|}{|D|} \times Info(D_i) \tag{17}$$

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],

$$Gain(A) = Info(D) - Info_A(D) \tag{18}$$

2.2.7.2 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

$$SplitInfo_A(D) = -\sum_{j=1}^{\nu} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right) \tag{19}$$

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]

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A D} \tag{20}$$

2.2.7.3 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

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 \tag{21}$$

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]

$$Gini(D) = \frac{|D_1|}{D} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \tag{22}$$

The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute A is [HPK11]

$$\Delta Gini(A) = Gini(D) - Gini_A(D) \tag{23}$$

2.2.8 Comparison of machine learning algorithms for sentiment analysis

Below is presented Naïve Bayes, Support vector machine, Logistic regression, Random forest and Decision tree comparison table.

Table 1: ML comparison

ML	Popularity*	Accuracy**	Performance**
`NB`	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆
`SVM`	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆
`LR (MaxEnt)`	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆
`RF`	⋆	⋆⋆⋆	⋆⋆
`DT`	⋆⋆	⋆⋆⋆	⋆⋆⋆⋆

Source: created by the author.
* applicability for sentiment analysis tasks, created based on K. Ravi and V. Ravi survey [RR15].
** based on researches in sentiment analysis.

2.3 Support Vector Machines

2.3.1 Researches in SVM optimization

According to a number of authors, who worked on SVM hyperparameter optimization, SVM proved its efficiency to solve difficult tasks in various domains. Damaševičius in [Dam10] used SVM for classification of DNA sequences and recognition of the regulatory sequences. The results demonstrated that selection of the kernel type and its parameters have direct impact on the performance of the SVM and accuracy of the results. Sunkad in [S$^+$16] proposed the best set of features and the SVM hyperparameters for obtaining the best results in human activity recognition. Osman et al. in [OGN17] tuned hyperparameters of two machine learning algorithms to improve bug prediction accuracy. They concluded that the k-nearest neighbours algorithm always significantly improved and the prediction accuracy of support vector machines either improved or was at least retained. Liu and Zio in [LZ17] used one synthetic dataset and two real time series data, related to prediction of wind speed in a region and leakage from the reactor coolant pump in a nuclear power plant and proposed the preferable choice for tuning SVM hyperparameters for recursive multi-step-ahead prediction.

However, despite all advantages, typical for SVM algorithm, it is characterized by slow performance in the big data arrays. The higher number of features is, the longer computation time it requires. There have been a number of efforts to speed up SVM, and most of them focus on reduction of the training set. Lee and Mangasarian in [LM01] proposed the Reduced Support Vector Machine (RSVM) algorithm which uses a randomly selected subset of the data that is typically 10% or less of the original dataset to obtain a nonlinear separating surface. RSVM performs better than a conventional SVM, sequential minimal optimization (SMO) and projected conjugate gradient chunking (PCGC). Lei and Govindaraju in [LG05] introduced a reduction of the feature space using principal component analysis (PCS) and Recursive Feature Elimination (RFE). PCA and RFE can speed up the evaluation of SVM by an order of 10 while maintaining comparable accuracy. Graf et al. in [GCB+05] proposed the Cascade SVM, where the training set is first divided into a number of subsets and then these subsets are optimized by multiple SVMs. The partial results were combined and filtered again in the "Cascade" of SVMs, until the global optimum was reached. Later, Meyer et al. in [MBW14] introduced a new stepwise bagging approach that exploits parallelization in a better way than the Cascade SVM and contains an adaptive stopping-time to select the number of stages for improving accuracy. Nandan et al. in [NKT14] used a linear time algorithm based on convex hulls and extreme points to select subset, the so-called representative set of the training data for SVM optimization. Wang et al. in [WLL+14] reduced SVM training time using only the most informative samples, obtained after removing most of the training data. Guo and Boukir in [GB15] proposed a new ensemble margin-based data selection approach based on a simple and efficient heuristic to provide support vector candidates: they selected the lowest margin instances that reduced SVM training task complexity while maintaining the accuracy of the SVM classification. Mao et al. in [MFWH16] trained number of kernel SVMs on the randomly selected small subsets of training data and concluded that it is more efficient than training a single kernel SVM on the whole training data especially for large datasets. Mourad et al. in [MTV17] proposed a computationally efficient subset selection algorithm for fast SVM training on large scale data. Liu et al. in [LZ17] proposed to train an approximate SVM by using the anchors obtained from non-negative matrix factorization (NMF) in a divide-and-conquer framework.

2.4 Heuristic optimization techniques

Heuristics optimization methods are very common in use combined with other machine learning algorithms to improve their accuracy, for feature selection tasks and etc. Elbeltagi et al. in [EHG05] compared five evolutionary-based optimization algorithms. According them five recent evolutionary-based algorithms are: genetic algorithms, memetic algorithms, particle swarm, ant-colony optimization, and shuffled frog leaping. They conclude that "the PSO method was generally found to perform better than other algorithms in terms of success rate and solution quality, while being second best in terms of processing time". Further is presented descriptions of each algorithm.

2.4.1 Genetic algorithms

The genetic algorithm (abbr. as GA) is an optimization and search technique based on the principles of genetics and natural selection. A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the “fitness” (i.e., minimizes the cost function). The method was developed by John Holland in [Hol75] over the course of the 1960s and 1970s and finally popularized by one of his students, David Goldberg, who was able to solve a difficult problem involving the control of gas-pipeline transmission for his dissertation (Holland and Goldberg in [HG89]) [HHH98].

GAs work with a random population of solutions (chromosomes). The fitness of each chromosome is determined by evaluating it against an objective function. To simulate the natural survival of the fittest process, best chromosomes exchange information (through crossover or mutation) to produce offspring chromosomes. The offspring solutions are then evaluated and used to evolve the population if they provide better solutions than weak population members. Usually, the process is continued for a large number of generations to obtain a best-fit (nearoptimum) solution [EHG05].

Genetic Algorithm

```
Generate random population of P solutions (chromosomes);
For each individual $i \in P$: calculate fitness (i);
    For i=1 to number of generations;
        Randomly select an operation (crossover or mutation);
        If crossover;
            Select two parents at random $i_a$ and $i_b$;
            Generate on offspring $i_c$ = crossover($i_a$ and $i_b$);
        Else If mutation;
            Select one chromosome i at random;
            Generate an offspring $i_c$ = mutate (i);
        End If;
        Calculate the fitness of the offspring $i_c$;
        If $i_c$ is better than the worst chromosome then replace the worst chromosome by $i_c$;
    Next i;
Check if termination = true;
```

Figure 1: Source: Elbeltagi et al. [EHG05]

Based on Haupt et al. in [HHH98] advantages of a GA are as follow:

- Optimizes with continuous or discrete variables,
- Doesn’t require derivative information,
- Simultaneously searches from a wide sampling of the cost surface,
- Deals with a large number of variables,
- Is well suited for parallel computers,
- Optimizes variables with extremely complex cost surfaces (they can jump out of a local minimum),
- Provides a list of optimum variables, not just a single solution,

- May encode the variables so that the optimization is done with the encoded variables,
- Works with numerically generated data, experimental data, or analytical functions.

2.4.2 Memetic algorithms

According Elbeltagi et al. in [EHG05] "Memetic algorithms (abbr. as MA) are inspired by Dawkins' notion of a meme [Daw76]. MAs are similar to GAs but the elements that form a chromosome are called memes, not genes. The unique aspect of the MAs algorithm is that all chromosomes and offsprings are allowed to gain some experience, through a local search, before being involved in the evolutionary process [MF97]". Like genetic algorithms, memetic Algorithms are a population-based approach. They have shown that they are orders of magnitude faster than traditional genetic Algorithms for some problem domains. In a memetic algorithm the population is initialized at random or using a heuristic. Then, each individual makes local search to improve its fitness. To form a new population for the next generation, higher quality individuals are selected. The selection phase is identical inform to that used in the classical genetic algorithm selection phase. Once two parents have been selected, their chromosomes are combined and the classical operators of crossover are applied to generate new individuals. The latter are enhanced using a local search technique. The role of local search in memetic algorithms is to locate the local optimum more efficiently then the genetic algorithm [Gar10].

2.4.3 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.

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{24}$$

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

Memetic Algorithm

```
Generate random population of P solutions (chromosomes);
For each individual i $\in$ P: calculate fitness (i);
For each individual i $\in$ P: do local-search (i);
    For i=1 to number of generations;
        Randomly select an operation (crossover or mutation);
        If crossover;
            Select two parents at random $i_a$ and $i_b$;
            Generate on offspring $i_c$ = crossover($i_a$ and $i_b$);
            $i_c$ = local-search($i_c$);
        Else If mutation;
            Select one chromosome i at random;
            Generate an offspring $i_c$ = mutate (i);
            $i_c$ = local-search($i_c$);
        End If;
        Calculate the fitness of the offspring $i_c$;
        If $i_c$ is better than the worst chromosome then replace the worst chromosome by $i_c$;
    Next i;
Check if termination = true;
```

the memetic local search

```
Select an incremental value d = a*Rand(), where a is a constant that suits the variable values;
For a given chromosome i $\in$ P: calculate fitness (i);
For j = 1 to number of variables in chromosome i;
    Value (j) = value (j) - d;
    If chromosome fitness not improved then value (j) = value (j) - d;
    If chromosome fitness not improved then retain the original value (j);
Next j;
```

Figure 2: Source: Elbeltagi et al. [EHG05]

of the velocity equation [Eng08].

2.4.3.1 Global Best PSO

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

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t)\left[\hat{y}_j(t) - x_{ij}(t)\right] \tag{25}$$

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 [Eng08].

$$y_i(t+1) = \begin{cases} y_i(t) & \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{if } f(x_i(t+1)) < f(y_i(t)) \end{cases} \tag{26}$$

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) [Eng08].
The global best position, *ŷ(t)*, at time step *t*, is defined as

$$\hat{y}(t) \in \{y_0(t), \ldots, y_{n_s}(t)\} | f(\hat{y}(t)) = \min\{f(y_0(t)), \ldots, f(y_{n_s}(t))\} \tag{27}$$

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, Eng08]

$$\hat{y}(t) = \min\{f(x_0(t)), \ldots, f(x_{n_s}(t))\} \tag{28}$$

```
gbest 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 global best position
        if $f(y_i) < f(\hat{y})$ then
            $\hat{y} = y_i$;
        end
    end
    for each particle $i = 1, \ldots, n_s$ do
        update the velocity using equation (2.25);
        update the position using equation (2.24);
    end
until stopping condition is true;
```

Figure 3: *gbest PSO algorithm* [Eng08]

2.4.3.2 Local Best PSO

The velocity is calculated as

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t)\left[\hat{y}_{ij}(t) - x_{ij}(t)\right] \tag{29}$$

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

$$\hat{y}(t{+}1) \in \{\mathcal{N}_i | f(\hat{y}(t{+}1)) = \min\{f(x)\}, \forall x \in \mathcal{N}_i\} \tag{30}$$

with the neighborhood defined as

$$\mathcal{N}_i = \{y_{i-n_{\mathcal{N}_i}}(t), y_{i-n_{\mathcal{N}_i}+1}(t), \ldots, y_{i-1}(t), y_i(t), y_{i+1}(t), \ldots, y_{i+n_{\mathcal{N}_i}}(t)\} \tag{31}$$

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 [Eng08].

```
**lbest** 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 neighborhood best position
        if $f(y_i) < f(\hat{y}_i)$ then
            $\hat{y} = y_i$;
        end
    end
    for each particle i = 1, ..., $n_s$ do
        update the velocity using equation (2.29);
        update the position using equation (2.24);
    end
until stopping condition is true;
```

Figure 4: *lbest PSO algorithm* [Eng08]

2.4.4 Ant-colony system

Ant system [Dor92] is the progenitor of all our research efforts with ant algorithms and was first applied to the traveling salesman problem. Ant-colony system (abbr. as ACS) algorithms are stochastic search procedures. Their central component is the pheromone model, which is used to probabilistically sample the search space. Informally, the ACS works as follows: ants are initially positioned on cities chosen according to some initialization rule. Each ant builds a tour by repeatedly applying a stochastic greedy rule. While constructing its tour, an ant also modifies the amount of pheromone on the visited edges by applying the local updating rule. Once all ants have terminated their tour, the amount of pheromone on edges is modified again (by applying the global updating rule). As was the case in ant system, ants are guided, in building their tours, by both heuristic information (they prefer to choose short edges) and by pheromone information. An edge with a high amount of pheromone is a very desirable choice. The pheromone updating rules are designed so that they tend to give more pheromone to edges which should be visited by ants [DG97].

In the Ant-colony optimization algorithm, the process starts by generating *m* random ants (solutions). An ant *k* ($k = 1, 2, \dots, m$) represents a solution string, with a selected value for each variable. Each ant is then evaluated according to an objective function. Accordingly, pheromone concentration associated with each possible route (variable value) is changed in a way to reinforce good solutions, as follows [DG97] [EHG05]:

$$\tau_{ij}(t) = \rho\tau_{ij}(t-1) + \mathit{\Delta}\tau_{ij}; t = 1, 2, \dots, T \tag{32}$$

where T is the number of iterations (generation cycles); $\tau_{ij}(t)$ is the revised concentration of pheromone associated with option l_{ij} at iteration t, $\tau_{ij}(t-1)$ is the concentration of pheromone at the previous iteration $(t-1)$; $\mathit{\Delta}\tau_{ij}$ =change in pheromone concentration; and ρ = pheromone evaporation rate (0-1). The reason for allowing pheromone evaporation is to avoid too strong influence of the old pheromone to avoid premature solution stagnation [MMS02] [EHG05]. In Eq. (2.32), the change in pheromone concentration$\mathit{\Delta}\tau_{ij}$ = is calculated as [DG97] [EHG05]:

$$\mathit{\Delta}\tau_{ij} = \sum_{k=1}^{m} \begin{cases} R/\text{fitness}_k \text{ if option } l_{ij} \text{ is chosen by ant} k \\ 0 \text{ otherwise} \end{cases} \tag{33}$$

where R is a constant called the pheromone reward factor; and fitness$_k$ is the value of the objective function (solution performance) calculated for ant *k*. It is noted that the amount of pheromone gets higher as the solution improves. Therefore, for minimization problems, Eq. (2.33) shows the pheromone change as proportional to the inverse of the fitness. In maximization problems, on the other hand, the fitness value itself can be directly used. Once the pheromone is updated after an iteration, the next iteration starts by changing the ants' paths (i.e. associated variable values) in a manner that respects pheromone concentration and also some heuristic preference. As such, an ant *k* at iteration *t* will change the value for each variable according to the following probability [DG97] [EHG05]:

$$P_{ij}(k,t) = \frac{[\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{l_{ij}} [\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}} \tag{34}$$

where $P_{ij}(k,t)$ = probability that option l_{ij} is chosen by ant *k* for variable *i* at iteration *t*; $\tau_{ij}(t)$ =pheromone concentration associated with option l_{ij} at iteration *t*; η_{ij} =heuristic factor for preferring among available options and is an indicator of how good it is for ant *k* to select option l_{ij} (this heuristic factor is generated by some problem characteristics and its value is fixed for each option l_{ij}); and α and β are exponent parameters that control the relative importance of pheromone concentration versus the heuristic factor [MSZ+03] [EHG05]. Both α and β can take values greater than zero and can be determined by trial and error. Based on the previous discussion, the main parameters involved in ACO are: number of ants *m*; number of iterations *t*; exponents α and β; pheromone evaporation rate ρ; and pheromone reward factor *R* [EHG05].

Ant-colony optimization

```
Initialize the pheromone trails and parameters;
    Generate population of m solutions (ants);
    For each individual ant k ∈ m: calculate fitness (k);
    For each ant determine its best position;
    Determine the best global ant;
    Update the pheromone trail;
Check if termination = true;
```

Figure 5: Source: Elbeltagi et al. [EHG05]

2.4.5 Shuffled frog leaping

The Shuffled frog leaping (abbr. as SFL) has been designed as a meta-heuristic to perform an informed heuristic search using a heuristic function (any mathematical function) to seek a solution of a combinatorial optimization problem. It is based on evolution of memes carried by the interactive individuals, and a global exchange of information among themselves [ELP06].

Initial population of *P* frogs is created randomly. For *S*-dimensional problems (*S* variables), a frog *i* is represented as $X_i = (x_{i1}, x_{i2}, \ldots, x_{iS})$. Afterwards, the frogs are sorted in a descending order according to their fitness. Then, the entire population is divided into *m* memeplexes, each containing n frogs (i.e. $P = m \times n$). In this process, the first frog goes to the first memeplex, the second frog goes to the second memeplex, frog *m* goes to the *m*th memeplex, and frog $m + 1$ goes back to the first memeplex, etc.

Within each memeplex, the frogs with the best and the worst fitnesses are identified as X_b and X_w, respectively. Also, the frog with the global best fitness is identified as X_g. Then, a process similar to PSO is applied to improve only the frog with the worst fitness (not all frogs) in each cycle. Accordingly, the position of the frog with the worst fitness is adjusted as follows:

$$\text{Change in frog position}(D_i) = rand() \times (X_b - X_w) \tag{35}$$

$$\text{New position}X_w = \text{current position}X_w + D_i; D_{max} \geq D \geq -Dmax \tag{36}$$

where rand() is a random number between 0 and 1; and D_{max} is the maximum allowed change in a frog's position. If this process produces a better solution, it replaces the worst frog. Otherwise, the calculations in Eqs. (8) and (9) are repeated but with respect to the global best frog (i.e. X_g replaces X_b). If no improvement becomes possible in this case, then a new solution is randomly generated to replace that frog. The calculations then continue for a specific number of iterations [EL03]. Accordingly, the main parameters of SFL are: number of frogs P; number of memeplexes; number of generation for each memeplex before shuffling; number of shuffling iterations; and maximum step size [EHG05].

Shuffled frog leaping

```
Generate random population of P solutions (frogs);
For each individual $i \in P$: calculate fitness (i);
Sort the population P in descending order of their fitness;
    Divide P into m memeplexes;
    For each memeplex;
        Determine the best and worst frogs;
        Improve the worst frog position using:
        Improve the worst frog position using Eqs. (2.32) or (2.33)
        Repeat for a specific number of iterations;
    End;
    Combine the evolved memeplexes;
    Sort the population P in descending order of their fitness;
Check if termination = true;
```

Figure 6: Source: Elbeltagi et al. [EHG05]

2.5 Clustering techniques in sentiment analysis

k-Means (MacQueen et al. in [M+67]) is one of the most popular and widely known techniques, used as standalone technique or in combination with others. Gu and Han in [GH13] proposed Clustered Support Vector Machine (CSVM) method, using k-Means for data dividing in clusters, in each to train linear SVM. Yao et al. in [YLY+13] used k-Means clustering algorithm to select the most informative samples into small subset from original training set for SVM training. Kurasova et al. in [KMM+14] presented an overview of techniques used for big data clustering and also identified k-means as one of the most popular and efficient techniques. Gan et al. in [GLL+17] used k-Means to construct a pre-selection scheme, which obtains a subset of important instances as training set for SVM. They reported that proposed KA-SVM has the outstanding performance on both of classification accuracy and computation efficiency. Wang et al. in [WZC+18] improved the spam filtering speed and filtering accuracy by proposed a fast content-based spam filtering algorithm with fuzzy-SVM and k-Means. k-Means was used to compress data with retain most of the effective information.

2.5.1 k-Means

k-Means (MacQueen et al. in [M+67]) is one of the oldest and widely research clustering algorithm. It is often preferred due to its simplicity and generally very fast performance. The main idea is to partition the input dataset into *k* clusters, represented by adaptively-changing centroids (also called cluster centers); they are initialized using so-called seed-points. k-Means computes the squared distances between the input data points and centroids, and assigns inputs to the nearest centroid. Formally, to solve problem of clustering *N* input data points $x_1, x_2, \dots, x_N$ into *k* disjoint subsets C_i, *i* = 1,…,*k*, each containing n_i data points, $0 < n_i < N$, the following mean-square-error (MSE) cost-function is minimized:

$$J_{MSE} = \sum_{i=1}^{k} \sum_{x_t \in C_i} \|x_t - c_i\|^2 \tag{37}$$

x_t is a vector representing the *t*-th data point in the cluster C_i and c_i is the geometric centroid of the cluster C_i. Finally, this algorithm seeks to minimize J_{MSE}, where $\|x_t - c_i\|^2$ is a chosen distance measurement between data point x_t and the cluster centre c_i. An input data point x_t is assigned to cluster i if it satisfies the following condition:

$$i = \arg\min(\|x_t - c_j\|^2) \;\; j = 1, \dots, k \tag{38}$$

Cluster centers $c_1, c_2, c_j, \dots, c_k$ can be obtained with the following steps [Ž08]:

Step 1: Initialize *k* cluster centres $c_1, c_2, \dots, c_k$ by some initial values called seed-points, using random sampling.
For each input data point x_t and all *k* clusters, repeat steps 2 and 3 until all centres converge.

Step 2: Calculate cluster membership function $I(x_t, i)$ by Eq. (4) and decide the membership of each input data point in one of the *k* clusters whose cluster centre is closest to that point.

Step 3: For all k cluster centres, set c_i to be the centre of mass of all points in cluster C_i.

2.6 Natural language processing

Natural language processing (abbr. as NLP) defined by Liddy in [lid01] is " a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications". According Sun et al. [SLC17] opinion mining requires several preprocessing steps for structuring the text and extracting features, including tokenization, word segmentation, Part of Speech (POS) tagging, parsing.

Tokenization is a fundamental technique for most NLP tasks. It splits a sentence or document into tokens which are words or phrases. For Chinese, Japanese or other languages which do not have explicit word boundary markers, tokenization is not as trivial as English and word segmentation is required. The word segmentation is a sequential labeling problem [SLC17].

Part of Speech (abbr. as POS) tagging and parsing are techniques that analyze the lexical and syntactic information. POS tagging is used to determine the corresponding POS tag for each word. The POS tags, such as adjective, noun, are quite helpful because opinion words are usually adjectives and opinion targets (i.e., entities and aspects) are nouns or combination of nouns. While POS tagging provides lexical information, parsing obtains syntactic information. Parsing produces a tree which represents the grammatical structure of a given sentence with the corresponding relationship of different constituents [SLC17]. Neethu and Rajasree in [NR13] proposed feature vector creation technique, which includes the 8 features: part of speech (pos) tag, special keyword, presence of negation, emoticon, number of positive keywords, number of negative keywords, number of positive hash tags and number of negative hash tags. According them "keywords that are adjective, adverb or verb shows more emotion than others." Ortigosa et al. [OMC14] combined lexicon based approach with ML algorithms and achieved the better results. Sentiment extraction contains these steps: lower-case, idiom detection; segmentation into sentences; tokenization; emoticon detection; interjection detection; token score assignation; POS tagging and syntactical analysis; polarity calculation. Pak and Paroubek in [PP10] for feature extraction used: filtering, which includes URL links, Twitter user names, Twitter special words and emoticons removal; tokenization; stopwords removal; n-grams constructing. Boiy and Moens in [BM09] for feature selection used unigrams, stems, negation, discourse features, depth difference, path distance and simple distance. Abdi et al. in [ASHP18] also used sentiment lexicon feature, negation features, sentence types, punctuation feature, POS feature, sentiment score feature. Yousefpour in [YIH17] used n-gram features, POS, negation, sentiment lexicon, syntactic and semantic dependency. Bharti and Singh in [BS15] applied stop words removal, stemming, tokenization, term weighting, relevance score computation, term variance, document frequency and merge feature sublists.

2.7 Features extraction techniques

Features extraction one of the NLP techniques and another very important part of sentiment analysis. Properly extracted features can increase accuracy of machine learning algorithm [KSTA15, KG15]. Tommasel and Godoi in [TG18] did a review of features selection techniques for short texts. According to summary of these techniques, the most commonly used data preprocessing are the following:

- Tokenisation [WHYL12, LYC+10, TXW+14, VVC+11, TWY+14, OSO12];
- Stopword removal [HSZC09, VVC+11, OSO12, THGL13, FZYL14, TL12];
- Stemming [OSO12, THGL13, TL12];
- TF-IDF irrelevant feature removal [TL12];
- Special symbol removal [VVC+11, FZYL14];
- URL's removal [VVC+11, TWY+14, SHA12, AAM+14];
- Remove of usernames [TWY+14, SHA12, AAM+14];
- Remove tweets with less than 7 tokens [TWY+14, JYZ+11];
- Remove hashtags and non-alphabetic characters [SHA12];
- Nouns, verbs, and adjectives are kept [WHYL12, LYC+10, TXW+14];
- Tweet segmentation [LSD12].

N-grams is another feature extracting technique which are comonly used in NLP. The mostly use are: unigrams, bigrams, trigrams and combination of them [PLV02, DLP03, NNVP14]. Agarwal and Mittal in [AM+16] gave definition of Unigrams and Bigrams. According them Unigram features are "simply bag-of-words(BoW) features extracted by eliminating extra spaces and noisy characters between two words". Respectively Bigrams are "the features, consisting of every two consecutive words in the text". Then we can define Trigrams as the features, consisting of every three consecutive words in the text. From the definitions we can see, that the prefix before "gram" points to number of consecutive words in the text.

2.8 Conclusions

In this section is presented related works in sentiment polarity classification field using machine learning algorithms.

1. The analysis has shown that sentiment analysis area is very challenging and interesting for researchers.

2. The literature analysis shown what Naïve Bayes Classification, Support vector machine, Logistic Regression, Random Forest and Decision Tree are the mostly used in sentiment analysis. Comparison of aforementioned techniques shown that SVM proved the best with its efficiency to solve diffcult tasks in various domains with higher accuracy.
3. Further SVM approaches characterized it by slow performance in the big data arrays. The higher number of features is, the longer computation time it requires. However SVM is very sensitive for parameter tuning. The accuracy can be significantly increased by correct parameter selecting. However literature still does not provide any heuristic rules or rules of thumb for SVM parameter selection and it is still one of the biggest issues, related to practical SVM research and application.
4. Researches by combining machine learning algorithms and NLP also shown promising results when the higher accuracy is needed.
5. The most common heuristic optimization techniques are overviewed and PSO is selected for methodology, which is presented in chapter "Methodology of the research".

3 Methodology of the research

3.1 Proposed method

Our introduced methodology is focused on combine SVM and Naïve Bayes classification algorithms to get better results. It is presented in paper Korovkinas et al. [KDG17]. In the figure below is presented system algorithm which show us principle of data processing from training data up to obtaining the results. Training and testing data had been preprocessed and cleaned before it was passed as the input of machine learning algorithms. It included removing redundant tokens such as hashtag symbols @, numbers, "http" for links, punctuation symbols, etc. Below are presented algorithms which are used in *"Combination"*. *"Results"* is the final results set with classified sentiments: "positive" or "negative".

Algorithm for words

Input: Let us denote the probability of word selection as p, and the threshold for its selection as th_1.The threshold values were selected by manually investigating the results. We found that the performance was optimal when word selection probability for was set to $p \geq 0.8$, therefore it was selected as threshold value th_1 for the algorithm for words.
$D_{test} = \{S_1, S_2, \ldots, S_n\}$ - set of testing data;
$S = \{w_1, w_2, \ldots, w_n\}$ - sentences;
w - words which are contained in sentence;
$R_{SVM} = \{SVMsent, v\}$ - set of SVM results, $SVMsent$ - sentiment;

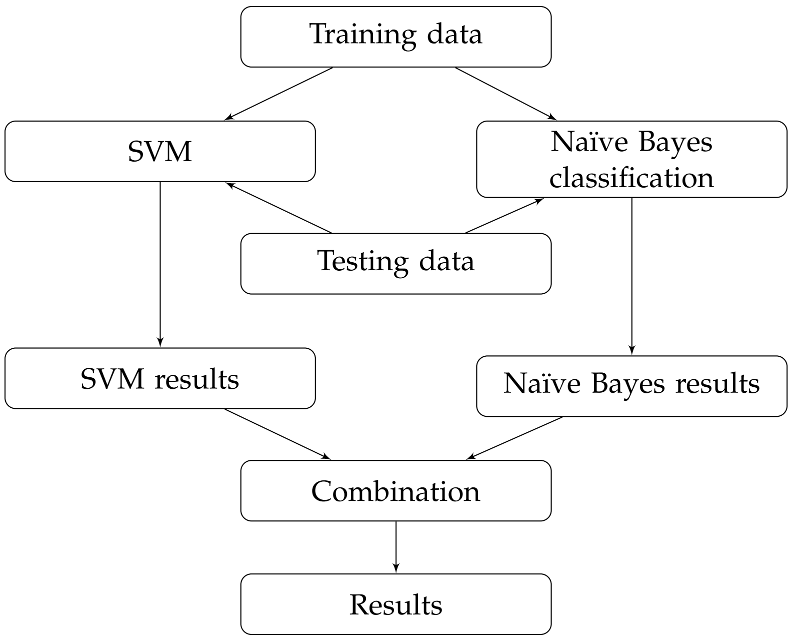

Figure 7: Proposed method for combining results.

$R_{NB} = \{NBsent, v\}$ - set of Naïve Bayes classification results, $NBsent$ - sentiment; v - value for Sum results.

1. SVM classification is performed:

```
$R_{SVM} = \{\}$
*for* $\forall S_i \in D_{test}$ :
    $v_i = 0$;
    *for* $\forall w_j \in S_i$ :
        *pass* $w_j$ *to SVM input (output* $SVMsent_j$ *and* $p_j$*)*
        if $|p_j| \geq th_1$
            if $SVMsent_j <> \text{“}positive''$
                $p_j = -(p_j)$
        $v_i = v_i + p_j$
    $R_{SVM} = R_{SVM} \cup \{SVMsent_i, v_i\}, \;\; SVMsent_i = \begin{cases} \text{“}positive'', \text{if } v_i \geq 0 \\ \text{“}negative'', \text{if } v_i < 0 \end{cases}$
```

2. Naïve Bayes classification is performed:

```
$R_{NB} = \{\}$
*for* $\forall S_i \in D_{test}$ :
    $v_i = 0$;
```

```
for $\forall w_j \in S_i$ :
    pass $w_j$ to Naïve Bayes input (output $v_j$)
    $v_j = \begin{cases} 1, \text{for “positive” words} \\ -1, \text{for “negative” words} \end{cases}$
    $v_i = v_i + v_j$
$R_{NB} = R_{NB} \cup \{NBsent_i, v_i\},\ \ NBsent_i = \begin{cases} \text{“}positive\text{''}, \text{if } v_i \geq 0 \\ \text{“}negative\text{''}, \text{if } v_i < 0 \end{cases}$
```

3. Results are combined as following:

   (a) Find results which are the same in both *SVM* and *Naïve Bayes classification*. $Results = R_{SVM} \cap R_{NB} = \{x : x \in R_{SVM}\{SVMsent\} \text{and } x \in R_{NB}\{NBsent\}\}$

   (b) Find results which are different between *SVM* and *Naïve Bayes classification*. $R_{SVM}\{SVMsent\}\Delta R_{NB}\{NBsent\}$

   (c) Find *coefficient of difference* for $\forall(R_{SVM}\{SVMsent\}\Delta R_{NB}\{NBsent\})$, using our proposed formula (we need to unify $R_{NB}\{v\}$ values, so we used $\log_{10}$ for it):
   $$difference = R_{SVM}\{p\} + \log_{10}(|R_{NB}\{v\}|)$$

   (d) Find $average$ of all *coefficients of difference*.

   (e) for $\forall difference_i \in difference$:
   $$Results = \begin{cases} Results \cup R_{SVM}, \text{if } difference_i \leq average \\ Results \cup R_{NB}, \text{if } difference_i > average \end{cases}$$

Output: set of classification results $Results = \{S, sentiment\}$ and Accuracy (see subsect 4.3).

Algorithm for sentences This technique is applied on the whole sentence without splitting into words.
Input: Let us denote th_2 ($th_2 = 0.8$ threshold value was manually selected for the algorithm for sentences.) as the threshold value for $R_{SVM}\{p\}$ selection in algorithm step (b) and th_3 as the threshold to select $R_{SVM}\{p\}$ in algorithm step (c).
$R_{SVM} = \{SVMsent, p\}$ - set of SVM results obtained after performing *SVM* classification; $SVMsent$ - sentiment
p - the probability of sentence classification
$R_{NB} = \{NBsent, v\}$ - set of Naïve Bayes classification results obtained after performing *Naïve Bayes classification*; $NBsent$ - sentiment
v - Naïve Bayes results value, contains “1” for “positive” sentence and “-1” for “negative” sentence

$th_3 = \min(R_{SVM}\{p\}) + \sigma_{R_{SVM}\{p\}}/2 - 0.01$ (used our proposed formula), where $\sigma_{R_{SVM}\{p\}}$ is the standard deviation of $R_{SVM}\{p\}$

Algorithm for results combining is performed:

1. Find results which are the same in both *SVM* and *Naïve Bayes classification*.
   $Results = R_{SVM} \cap R_{NB} = \{x : x \in R_{SVM}\{SVMsent\}$and $x \in R_{NB}\{NBsent\}\}$
2. Find results which are different between *SVM* and *Naïve Bayes classification*.
   $R_{SVM}\{SVMsent\} \Delta R_{NB}\{NBsent\}$ and $R_{SVM}\{p\} < th_2$
3. $Results = \begin{cases} Results \cup R_{SVM}, \text{if } |R_{SVM}\{p\}| < th_3 \\ Results \cup R_{NB}, \text{if } |R_{SVM}\{p\}| \geq th_3 \end{cases}$

Output: set of classification results $Results = \{S, sentiment\}$ and Accuracy (Korovkinas et al. in [KDG17]).

In paper Korovkinas and Garšva [KG18], we modified this algorithm for using it with different machine learning algorithms. This algorithm is presented below.

Algorithm for combining results

Input: Let us denote $ML1$ as the strongest classifier and $ML2$ as the weakest classifier.

$R_{ML1} = \{ML1\_sent, p\}$ - set of the first algorithm results, obtained after performing machine learning algorithm $ML1$ classification; $ML1\_sent$ - sentiment

p - the probability of classification

$R_{ML2} = \{ML2\_sent, v\}$ - set of the second machine learning $ML2$ classification results obtained after performing $ML2$; $ML2\_sent$ - sentiment

v - $ML2$ results value, contains "positive" or "negative" sentiment

$th_2 = 0.8$. The threshold value was selected by manually investigating the results.

$th_3 = \min(R_{ML1}\{p\}) + (\sigma_{R_{ML1}\{p\}} \setminus 2) - 0.01$ (used our proposed formula), where $\sigma_{R_{ML1}\{p\}}$ is the standard deviation of $R_{ML1}\{p\}$

Algorithm for results combining:

1. Find results which are the same in both $ML1$ and $ML2$.
   $Results = R_{ML1} \cap R_{ML2} = \{x : x \in R_{ML1}\{ML1\_sent\}$ and $x \in R_{ML2}\{ML2\_sent\}\}$
2. Find results which are different between $ML1$ and $ML2$.
   $R_{ML1}\{ML1\_sent\} \Delta R_{ML2}\{ML2\_sent\}$ and
   $R_{ML1}\{p\} < th_2$
3. $Results = \begin{cases} Results \cup R_{ML1}, \text{if } |R_{ML1}\{p\}| < th_3 \\ Results \cup R_{ML2}, \text{if } |R_{ML1}\{p\}| \geq th_3 \end{cases}$

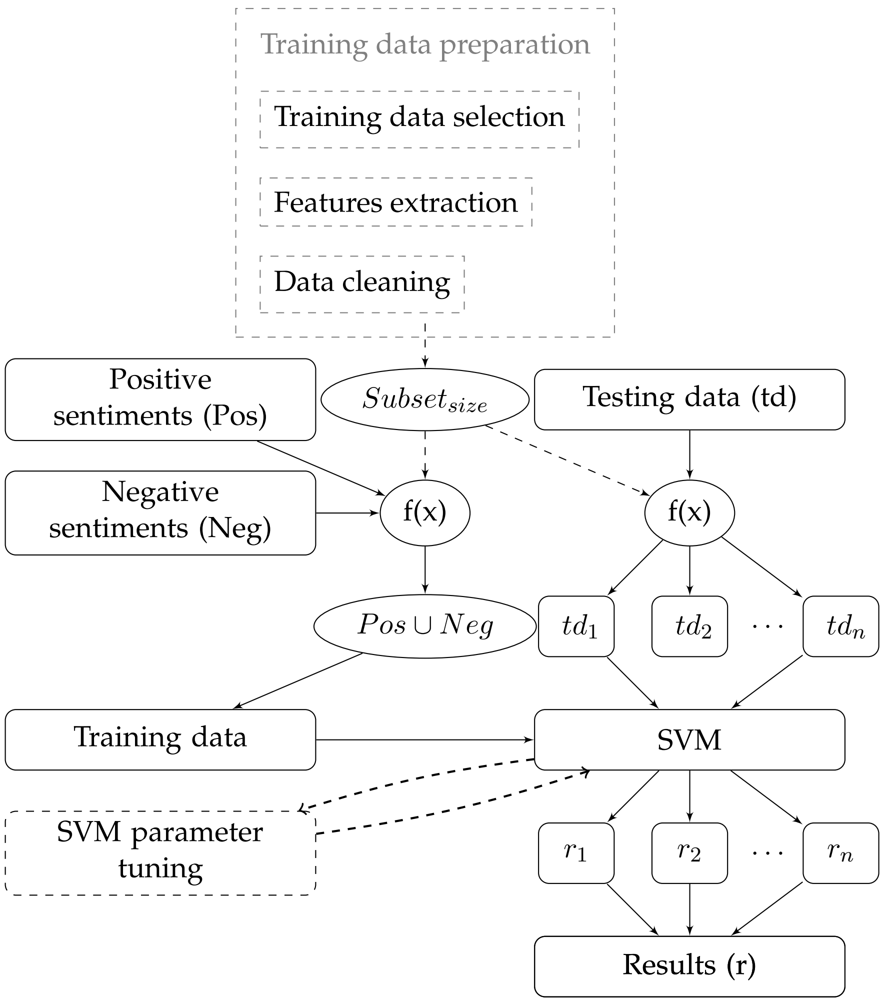

Figure 8: Proposed method

Output: set of classification results $Results = \{S, sentiment\}$ and Accuracy (Korovkinas and Garšva in [KG18]).

As was mentioned in chapter 2 "Review of existing techniques and problem domain", SVM has slow performance with large scale datasets - big data arrays. In paper [KDG18] we presented the method to improve the speed of SVM classification in sentiment analysis by reducing the training set. The method is based on selection of the training data size subject to the subset of split testing data. Thus, the testing data is split into equal subsets and training data size is calculated on the basis of the size of the first subset. It is assumed that the testing subset is 30%, therefore, the training data should be 70%. "Results" is the final results set with the following classified sentiment: "positive" or "negative". The diagram (see Fig. 3.2) and algorithm of the proposed method are presented below.

Algorithm

Input: Pos_{data} – set of positive sentiments for training;

Neg_{data} – set of negative sentiments for training;
td – set of testing data subsets;
$Subset_{size}$ – size of testing data subset is divided into;
D_{train} – set of training data;
D_{test} – set of testing data;
$Train_{count}$ – count of sentiments should be selected from Pos_{data} and Neg_{data} sets. This value is calculated by formula:
$Train_{count} = ((Subset_{size}/2) * (train_{size}/test_{size})))).$ $Subset_{size}$ is divided by 2, because we need to select equal parts from Pos_{data} and Neg_{data} sets;
POS_{train} – set of randomly selected sentiments from Pos_{data} set;
NEG_{train} – set of randomly selected sentiments from Neg_{data} set;
$R_{SVM} = \{SVMsent\}$ – set of SVM results, $SVMsent$ – sentiment;

```
$R_{SVM} = \{\}$
$D_{train} = \{\}$
$POS_{train} = (random.sample(Pos_{data}, Train_{count}))$
$NEG_{train} = (random.sample(Neg_{data}, Train_{count}))$
$D_{train} = POS_{train} \cup NEG_{train}$
train SVM with $D_{train}$
n = 0;
for i = 1 : $trunc(len(D_{test})/Subset_{size})$
        $td_i = D_{test}[(n+1) : (Subset_{size} * i),]$
        pass $td_i$ to SVM input (output $SVMsent_i$)
        $R_{SVM} = R_{SVM} \cup \{SVMsent_i\}$
        $n = (Subset_{size} * i)$
if $(len(D_{test})\ \%\ Subset_{size}) > 0$
        $td_{i+1} = D_{test}[(n+1) : (len(D_{test}),]$
        pass $td_{i+1}$ to SVM input (output $SVMsent_{i+1}$)
        $R_{SVM} = R_{SVM} \cup \{SVMsent_{i+1}\}$
```

Output: set of classification results $Results = \{sentiment\}$

3.2 Conclusions

In this section is presented the proposed hybrid method for sentiment polarity classification, its diagram, pseudo code. The method contains 5 main parts: selecting of representative dataset, features extracting, SVM parameter tuning, SVM speed-up technique, combination Naïve Bayes and SVM results.

4 Experimental research

4.1 Dataset

In this thesis are used existing datasets: *"A list of English positive and negative opinion words or sentiment words"* [5], Movie Review (polarity dataset v2.0 [6]), the Stanford Twitter sentiment corpus dataset[7] and Amazon customer reviews dataset[8].

"A list of English positive and negative opinion words or sentiment words", was compiled by authors Hu and Liu in [HL04]. It is actually a list of opinion lexicon. List of positive words contains 2006 words and negative list contains 4783 words. We add additional column in this list, which named "Sentiment". This column contains two values: "positive" for positive words list and "negative" for negative words list. After we combine these two lists in to one. The prepared training dataset contains 6789 words.

Movie Review dataset, was created by Pang and Lee in [PL04] and contains 1000 positive and 1000 negative processed reviews. Dataset was splitted into training data (70%), which was used in second experiment as the training dataset (1400 movie reviews) and testing data (30%).

The Stanford Twitter sentiment corpus dataset is introduced by Go et al. in [GBH09] and contains 1.6 million tweets automatically labeled as positive or negative based on emotions. The dataset is splitted in training dataset (560K positive and 560K negative tweets, in total 1.12M tweets) and testing dataset (480K tweets). Amazon customer reviews dataset contains 4 million reviews and star ratings. The dataset is splitted into training dataset (1.4M positive and 1.4M negative reviews, in total 2.8M reviews) and testing dataset (1.2M reviews).

Training and testing data has been preprocessed and has been cleaned before it was passed as the input of SVM algorithm. It included removing redundant tokens such as hashtag symbols @, numbers, "http" for links, punctuation symbols, etc. After cleaning was performed all datasets were checked and empty strings were removed.

4.2 Experiments

Number of experiments are performed with aforementioned datasets.

In the first four experiment was used *"A list of English positive and negative opinion words or sentiment words"*, Movie Reviews dataset, Stanford Twitter sentiment corpus dataset (sentiment140) and Amazon customer reviews dataset (Amazon reviews). After the results are compared with results when was used standalone machine learning algorithms NB and SVM.

[5] `https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html`

[6] `http://www.cs.cornell.edu/people/pabo/movie-review-data/`

[7] `http://help.sentiment140.com/`

[8] `https://www.kaggle.com/bittlingmayer/amazonreviews/`

For the first experiment training, we used dataset "*A list of English positive and negative opinion words or sentiment words*", which contains 6789 words (see subsec 4.1).

The dataset for testing was used Movie review dataset. Dataset was splitted into training data (70%), which was used in second experiment as the training dataset (1400 movie reviews) and testing data (30%). We used the same testing dataset (600 movie reviews) in first and the second experiments.

For the third experiment was used The Stanford Twitter sentiment corpus dataset, which contains 50K positive and 50K negative (total 100K) randomly selected tweets. The dataset was splitted into training (70%) and testing (30%) datasets.

For the fourth experiment was used Amazon customer reviews dataset, which contains 200K positive and 200K negative (total 400K) randomly selected reviews. The dataset was splitted into training (70%) and testing (30%) datasets.

Experimental settings for aforementioned experiments are presented in Table 4.1 (see section 4.3).

Another four experiments were performed to evaluate and compare the proposed method with combination of various classifiers. In first two experiments we compared four standalone classifiers and applied them on stanford140 and Amazon reviews dataset. In the third and fourth experiments the best three machine learning algorithms are selected, depending on results of the previous experiments, for the creating various combinations of two different single methods and apply them on aforementioned datasets. For experiments were used: Logistic Regression, Naïve Bayes classification, Support Vector Machines and Random Forest algorithms. Datasets were splitted into 70% for training and 30% for testing (see Table 4.2).

To evaluate the SVM speed performance, another ten experiments are performed in this thesis: five experiments with the Stanford Twitter sentiment corpus dataset (sentiment140) and five experiments with Amazon customer reviews dataset (Amazon reviews).

Original linear SVM technique is used in the first and the second experiments, using typical split into 70% for training and 30% for testing, and applying it to the Stanford Twitter sentiment corpus dataset and to Amazon customer reviews dataset. The main goal of the aforementioned experiments is to compare the efficiency of an ordinary SVM technique and our proposed method, which is applied in the further experiments of this thesis.

In the third and seventh experiments, the testing data is used from the first (480K tweets) and second experiments (1.2M reviews). Furthemore, it is divided into subsets, which contain 30K rows of a dataset. The last subset contains the remainder after division, if the testing dataset cannot be divided into equal subsets without the remainder. Similar splitting into subsets is used in other experiments, using different size of subsets: 60K rows of a dataset are used in the fourth and eight experiments; in the fifth and ninth experiments – 120K rows of a dataset; in the sixth and tenth experiments

– 180K rows of a dataset.

It is important to note that all experiments are performed 20 times to get more accurate results; the MIN, the MAX and the average are taken as the final results. Training data is calculated subject to the subset of testing data as it is described in the proposed method (see subsec 2.2) and new training set is randomly selected for each experiment and for each execution time.

Detailed experimental settings are presented in Table 4.2 and Table 4.3.

4.3 Experimental settings

Data cleaning, preparing and the experiments are implemented with Python programming language and scikit-learn [PVG+11]: library for machine learning.

Machine learning algorithms are used with their default parameters. They are described below.

Logistic Regression default parameters [PVG+11]:

- *C* (Inverse of regularization strength): float, default: 1.0.
- *dual* (Dual or primal formulation): bool, default: False
- *fit_intercept* (Specifies if a constant should be added to the decision function): bool, default: True
- *intercept_scaling*: float, default 1
- *max_iter*(Maximum number of iterations taken for the solvers to converge): int, default: 100
- *multi_class*: str, default: 'ovr'. With 'ovr' a binary problem is fit for each label.
- *n_jobs* (Number of CPU cores used when parallelizing over classes if multi_class='ovr'): int, default: 1
- *penalty* (Used to specify the norm used in the penalization): str, 'l1' or 'l2', default: 'l2'
- *solver* (Algorithm to use in the optimization problem): 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga', default: 'liblinear'.
- *tol* (Tolerance for stopping criteria): float, default: 0.0001

Naïve Bayes default parameters [PVG+11]:

- *alpha* (Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing)): float, optional (default=1.0)
- *fit_prior* (Whether to learn class prior probabilities or not): boolean, optional (default=True)
- *class_prior* (Prior probabilities of the classes): array-like, size (n_classes), optional (default=None)

Random Forest default parameters [PVG+11]:

- *n_estimators* (The number of trees in the forest): integer, optional (default=10)
- *max_features* (The number of features to consider when looking for the best split): int, float, string or None, optional (default="auto")
- *max_depth* (The maximum depth of the tree): integer or None, optional (default=None)
- *min_samples_split* (The minimum number of samples required to split an internal node): int, float, optional (default=2)
- *min_samples_leaf* (The minimum number of samples required to be at a leaf node): int, float, optional (default=1)
- *min_weight_fraction_leaf* (The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node): float, optional (default=0.0)
- *max_leaf_nodes* (Grow trees with *max_leaf_nodes* in best-first fashion. Best nodes are defined as relative reduction in impurity): int or None, optional (default=None)
- *min_impurity_decrease* (A node will be split if this split induces a decrease of the impurity greater than or equal to this value): float, optional (default=0.0)
- *bootstrap* (Whether bootstrap samples are used when building trees): boolean, optional (default=True)
- *oob_score* (Whether to use out-of-bag samples to estimate the generalization accuracy): bool (default=False)
- *n_jobs* (The number of jobs to run in parallel for both fit and predict): integer, optional (default=1)
- *verbose* (Controls the verbosity of the tree building process): int, optional (default=0)
- *warm_start* : bool, optional (default=False)
- *criterion* (The function to measure the quality of a split): string, optional (default="gini")

In the case of SVM, was used LinearSVC module for SVM classification with this default parameters (all parameters are selected as they are in LinearSVC module). It is similar to SVC (implementation of conventional SVM) with parameter kernel="linear", but implemented in terms of LibLinear (A Library for Large Linear Classification [9]) rather than LibSVM (A Library for Support Vector Machines [10]), so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples [PVG+11]. The main goal of this research is to compare training speed between our method and ordinary SVM on equal terms, consequently we don't need to change values of SVM parameters, cause it is enough to compare accuracy of methods obtained with default parameters.

[9] `https://www.csie.ntu.edu.tw/~cjlin/liblinear/`

[10] `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`

Table 2: Experimental settings

Exp No.	Training features	Training dataset	Testing features	Testing dataset
1	words	6789	movie reviews	600
2	movie reviews (70%)	1400	movie reviews (30%)	600
3	tweets (70%)	70000	tweets (30%)	30000
4	Amazon Reviews (70%)	280000	Amazon Reviews (30%)	120000

Table 3: Typical dataset split

Exp. No.	Dataset	Training data 70%	Testing data 30%
1	sentiment140	1.12M	480K
2	Amazon reviews	2.8M	1.2M

Support Vector Machines parameters:

- kernel: linear.
- *C* (Penalty parameter of the error term. It is the only parameter for linear classification.). Type: float, optional (default=1.0) [PVG+11]

Table 4: Experimental settings for proposed method

Exp. No.	Dataset	Testing data size (TDs)	Subset size (SubS)	Subsets quantity (SQ) trunc(TDs/Ss)	Remainder TDs-(SubS*SQ)	Calculated training data dependently on SubS
3	sentiment 140	480K	30K	16	0	70K
4		480K	60K	8	0	140K
5		480K	120K	4	0	280K
6		480K	180K	2	120K	420K
7	Amazon reviews	1.2M	30K	40	0	70K
8		1.2M	60K	20	0	140K
9		1.2M	120K	10	0	280K
10		1.2M	180K	6	120K	420K

For experiments is used computer with processor Intel(R) Core(TM) i7-4712MQ CPU @ 2.30 GHz and 16.00 GB installed memory (RAM).

4.4 Effectiveness

Effectiveness is measured using statistical measures: accuracy (ACC), precision (PPV - positive predictive value and NPV - negative predictive value), recall (TPR - true positive

rate and TNR - true negative rate) and F_1 (Harmonic mean of PPV and TPR). Formulas are presented below (Sammut and Webb in [SW11]):
Accuracy (ACC):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Positive predictive value (PPV):

$$PPV = \frac{TP}{TP + FP}$$

Negative predictive value (NPV):

$$NPV = \frac{TN}{TN + FN}$$

True positive rate (TPR):

$$TPR = \frac{TP}{TP + FN}$$

True negative rate (TNR):

$$TNR = \frac{TN}{TN + FP}$$

Harmonic mean of PPV and TPR (F_1):

$$F_1 = \frac{2}{\frac{1}{PPV} + \frac{1}{TPR}}$$

where *TP* - count of correctly classified "positive" sentiments, *TN* - count of correctly classified "negative" sentiments. *FP* - count of incorrectly classified "positive" sentiments. *FN* - count of incorrectly classified "negative" sentiments.

4.5 Results

Table 4.4 shows that the best results we got in forth experiment when was recognizing Amazon reviews sentiments. Our introduced method gave accuracy (ACC) 89,19%, while accuracy of SVM (ACC) was 89,05% and Naïve Bayes (ACC) 84,35 %. Not far away from the best results is the second experiment when was recognizing movie reviews with accuracy (ACC): our introduced method 88,66 %, SVM 88,50 % and Naïve Bayes 81,67 %. The first experiment where was used a list of English positive and negative opinion words or sentiment "*A list of English positive and negative opinion words or sentiment words*" for recognize movie reviews, showed the lowest recognize accuracy, but our introduced method still outperform SVM and Naïve Bayes and gave accuracy (ACC) 72,00%. To compare the first and the second experiments, where was used the same testing dataset, we found that the better accuracy is obtained when the sentences is not tokenized and the training dataset is from the same domain. In third experiment, when was recognizing sentiments from tweets, our introduced method shown accuracy (ACC) 78,31% and again outperform SVM with accuracy (ACC) 78,08 % and Naïve Bayes with accuracy 75,77 %. As we can see the best results are when recognizing movie reviews and Amazon reviews, the results of tweets recognizing is not very high, but still good enough if we don't need a very high accuracy. This happen because tweets are very short and tweets contain noises, slangs, acronyms and etc.

To compare with SVM and Naïve Bayes classification, our introduced method provided more uniform recognition of both classes (exept the first experiment where we gave almost the same), compared to other methods. $PPV, NPV, TPR, TNR, F_1 score$, are almost even in our introduced method, while Naïve Bayes have spread from 76,33% till 87,00% in second experiment, from 70,75% till 80,79% in third experiment and from 79,96% till 88,75% in fourth experiment. It can be indicated that Naïve Bayes classifier performed weekly in all experiments, but its combination with stronger classifier, such as SVM, can improve performance of the latter.

Results suggest that training and testing datasets should come from the same domain, which limits the direct transfer of the pretrained classifier to other domains. Also, Naïve Bayes classifier did not perform well while recognizing sentiments in all experiments to compare with SVM and our introduced method.

Fig. 4.1 – 4.4 graphically depict the results of the accuracy and $F_1 score$. In Fig. 4.5 – 4.8 – the results of *PPV, NPV, TPR, TNR*.

Table 5: Results

Classifiers	ACC	PPV	NPV	TPR	TNR	$F_1 score$
Experiment 1						
SVM	71,33%	64,33%	78,33%	74,81%	68,71%	69,16%
Naïve Bayes classification	67,83%	71,67%	64,00%	66,56%	69,31%	69,02%
Our introduced method	72,00%	65,67%	78,33%	75,19%	69,53%	70,11%
Experiment 2						
SVM	88,50%	89,67%	87,33%	87,62%	89,42%	88,63%
Naïve Bayes classification	81,67%	76,33%	87,00%	85,45%	78,61%	80,63%
Our introduced method	88,66%	90,00%	87,33%	87,66%	89,73%	88,82%
Experiment 3						
SVM	78,08%	78,74%	77,43%	77,72%	78,46%	78,23%
Naïve Bayes classification	75,77%	70,75%	80,79%	78,65%	73,42%	74,49%
Our introduced method	78,31%	77,82%	78,80%	78,59%	78,04%	78,20%
Experiment 4						
SVM	89,05%	88,11%	89,99%	89,80%	88,33%	88,95%
Naïve Bayes classification	84,35%	79,96%	88,75%	87,66%	81,58%	83,64%
Our introduced method	89,19%	88,10%	90,32%	90,10%	88,33%	89,07%

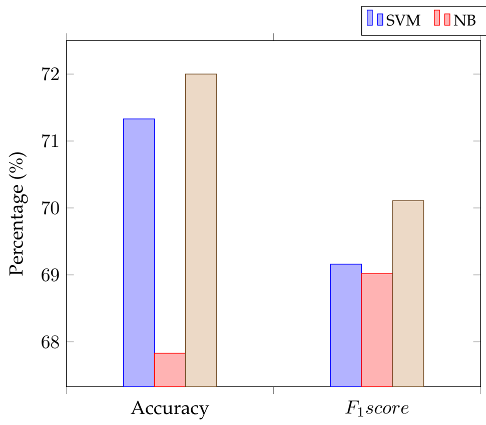

Figure 9: Experiment 1 results

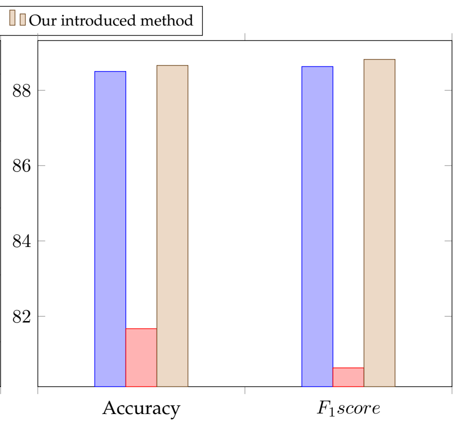

Figure 10: Experiment 2 results

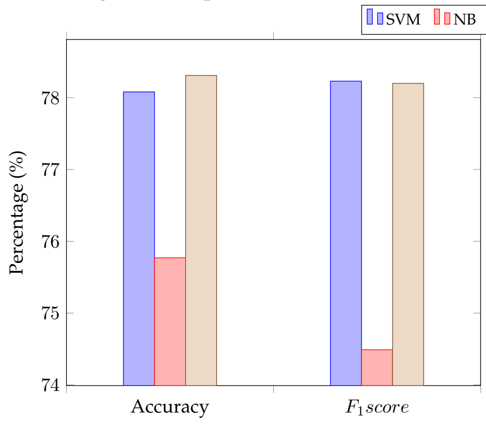

Figure 11: Experiment 3 results

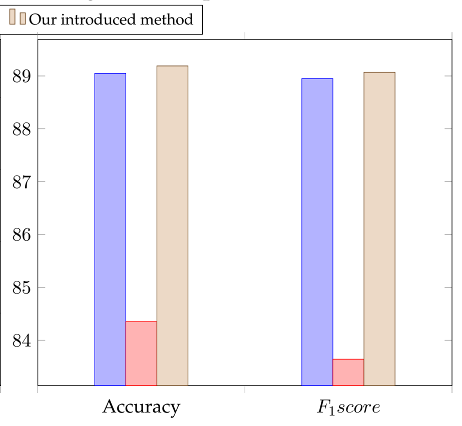

Figure 12: Experiment 4 results

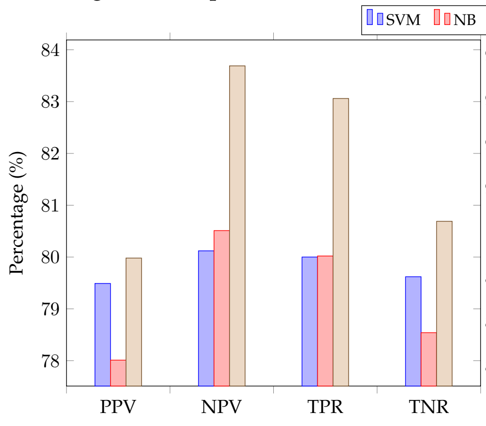

Figure 13: Experiment 1 results

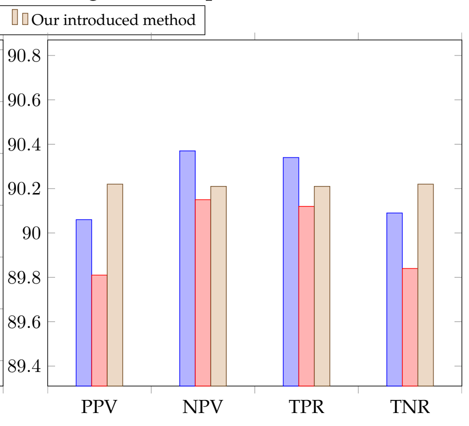

Figure 14: Experiment 2 results

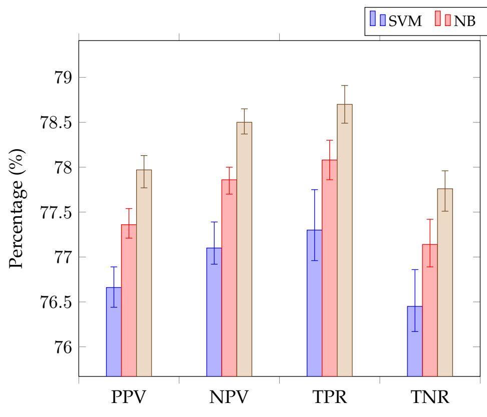

Figure 15: Experiment 3 results

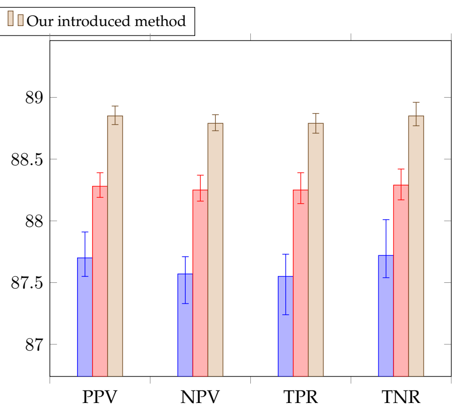

Figure 16: Experiment 4 results

Table 4.5 contains the second results of standard single machine learning algorithms with their default parameters. Results show that Logistic Regression (LR) obtained the best accuracy (ACC) in both experiments 79,67% and 90,21%. Other methods are arranged in the following order: SVM (ACC) - 79,16% and 90,00%, Naïve Bayes classification (ACC) - 76,72% and 84,18%, Random Forest (ACC) - 75,81% and 80,15%.

Table 6: the single methods experiment results

Classifiers	ACC	PPV	NPV	TPR	TNR	$F_1 score$
Experiment 1						
LR	79,67%	80,19%	79,16%	79,38%	79,98%	79,78%
NB	76,72%	73,18%	80,26%	78,76%	74,95%	75,87%
SVM	79,16%	79,49%	78,82%	78,97%	79,35%	79,23%
RF	75,81%	70,12%	81,49%	79,13%	73,17%	74,35%
Experiment 2						
LR	90,21%	90,19%	90,24%	90,24%	90,19%	90,21%
NB	84,18%	81,46%	86,89%	86,14%	82,42%	83,74%
SVM	90,00%	90,03%	89,98%	89,98%	90,03%	90,01%
RF	80,15%	73,05%	87,25%	85,14%	76,40%	78,63%

The better accuracy obtained when was used Amazon reviews dataset, while it significantly bigger than sentiment140 dataset. This happened because tweets are very short, contain noises, slangs, acronyms and etc.

Logistic Regression and SVM provided more uniform recognition of both classes; PPV, NPV, TPR, TNR, F_1, are almost even, compared to other methods.

Depending on results presented in Table 4.5, for the further experiments were selected Logistic Regression, SVM and Naïve Bayes. Various combinations of two different single algorithms were performed in these experiments.

Table 4.6 shows that using proposed method for combination of two single methods, let us to obtain the better accuracy to compare with a single method.

LR-SVM (Logistic Regression and SVM combination) shows the better accuracy

Table 7: the single methods experiment results

Classifiers	ACC	PPV	NPV	TPR	TNR	$F_1 score$
Experiment 1						
LR-NB	79,81%	79,49%	80,12%	80,00%	79,62%	79,75%
SVM-NB	79,26%	78,01%	80,51%	80,02%	78,54%	78,99%
LR-SVM	81,83%	79,98%	83,69%	83,06%	80,69%	81,49%
Experiment 2						
LR-NB	90,22%	90,06%	90,37%	90,34%	90,09%	90,20%
SVM-NB	89,98%	89,81%	90,15%	90,12%	89,84%	89,96%
LR-SVM	90,22%	90,22%	90,21%	90,21%	90,22%	90,22%

(ACC) 81,83% and 90,22%, while (ACC) of other combinations are smaller: LR-NB (Logistic Regression and Naïve Bayes combination) - 79,81% and 90,22%, SVM-NB (SVM and Naïve Bayes combination) - 79,26% and 89,98%. Our introduced method also outperformed single LR algorithm in all experiments, except the fourth experiment where SVM-NB obtained accuracy (ACC) 89,98% to compare with Logistic Regression 90,21%.

Our method also provided more uniform recognition of both classes PPV, NPV, TPR, TNR, F_1.

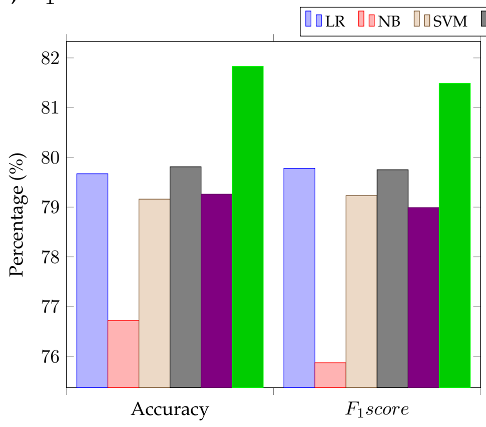

Figure 17: Experiment 1 results

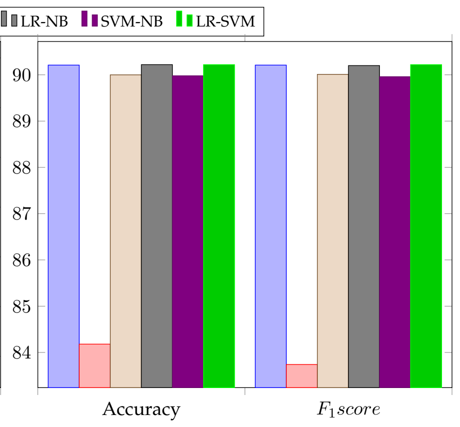

Figure 18: Experiment 2 results

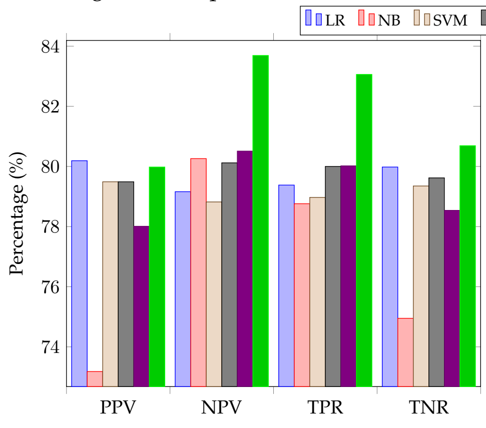

Figure 19: Experiment 1 results

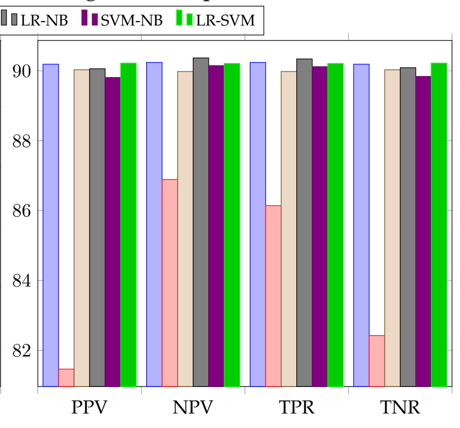

Figure 20: Experiment 2 results

The results of an ordinary SVM technique (see Table 4.7) revealed that in case of sentiment140 dataset the execution time average for is 407,26 sec., meanwhile the accuracy (ACC) average is 79,10%. While referring to Amazon customer reviews dataset, the execution time average is 1031,64 sec. and the accuracy (ACC) average is 89,44%.

Table 8: Results of ordinary SVM technique, using typical dataset split

Dataset	Function	time (sec)	ACC	PPV	NPV	TPR	TNR	$F_1 score$
sentiment140	MIN	372,79	79,09%	78,81%	79,31%	79,46%	78,59%	79,17%
	MAX	464,71	79,11%	78,89%	79,38%	79,59%	78,72%	79,21%
	AVG	407,26	79,10%	78,84%	79,36%	79,56%	78,64%	79,20%
Amazon	MIN	1004,93	89,30%	91,49%	86,74%	85,81%	91,89%	88,91%
reviews	MAX	1053,14	89,62%	92,31%	87,85%	87,28%	92,85%	89,37%
	AVG	1031,64	89,44%	91,97%	87,21%	86,44%	92,45%	89,11%

Table 4 shows the results of the proposed method for the sentiment140 dataset. Average execution time of our method ranges from 7,47 sec. to 51,44 sec. Therefore, it outperforms an ordinary SVM technique with the average of 407,26 sec.; however, the accuracy (ACC) average of our method ranges from 76,87% to 78,55% and is slightly lower than the ordinary SVM technique (ACC 79,10%). Moreover, the results clearly demonstrate that the performance in terms of *PPV, NPV, TPR, TNR, F_1score* is very similar in all experiments.

Table 9: Results of the proposed method applied on the Stanford Twitter sentiment corpus dataset

Subset	Function	time (sec)	ACC	PPV	NPV	TPR	TNR	$F_1 score$
30K	MIN	7,39	76,72%	76,44%	76,92%	76,96%	76,17%	76,85%
	MAX	7,67	77,03%	76,89%	77,39%	77,75%	76,86%	77,16%
	AVG	7,47	76,87%	76,66%	77,10%	77,30%	76,45%	76,98%
60K	MIN	10,06	77,52%	77,21%	77,70%	77,86%	76,89%	77,59%
	MAX	10,86	77,69%	77,54%	78,00%	78,30%	77,42%	77,79%
	AVG	10,19	77,61%	77,36%	77,86%	78,08%	77,14%	77,72%
120K	MIN	21,27	78,08%	77,77%	78,37%	78,49%	77,51%	78,20%
	MAX	25,78	78,35%	78,13%	78,65%	78,91%	77,96%	78,46%
	AVG	23,28	78,23%	77,97%	78,50%	78,70%	77,76%	78,34%
180K	MIN	47,73	78,47%	78,22%	78,69%	78,86%	77,96%	78,56%
	MAX	55,28	78,62%	78,37%	78,95%	79,20%	78,21%	78,74%
	AVG	51,44	78,55%	78,29%	78,82%	79,02%	78,08%	78,65%

Table 4.8 presents the results for Amazon customer reviews dataset. From here, it is evident that the proposed method again shows very good execution time: the average from 67,17 sec. to 105,08 sec. while it took 1031,64 sec. for ordinary SVM. The accuracy (ACC) average from 87,63% to 89,09% was slightly smaller than in case of an ordinary SVM method with the average of 89,44%. The performance in terms of *PPV, NPV, TPR,*

TNR, F_1score is almost identical in all experiments.

Table 10: Results of the proposed method applied on Amazon customer reviews dataset

Subset	Function	time (sec)	ACC	PPV	NPV	TPR	TNR	F_1score
30K	MIN	65,41	87,55%	87,55%	87,33%	87,24%	87,54%	87,55%
	MAX	70,02	87,70%	87,91%	87,71%	87,73%	88,01%	87,68%
	AVG	67,17	87,63%	87,70%	87,57%	87,55%	87,72%	87,62%
60K	MIN	71,67	88,21%	88,19%	88,16%	88,14%	88,17%	88,20%
	MAX	75,24	88,34%	88,39%	88,37%	88,39%	88,42%	88,33%
	AVG	73,73	88,27%	88,28%	88,25%	88,25%	88,29%	88,27%
120K	MIN	85,31	88,79%	88,78%	88,73%	88,71%	88,77%	88,79%
	MAX	97,50	88,86%	88,93%	88,86%	88,87%	88,96%	88,85%
	AVG	88,54	88,82%	88,85%	88,79%	88,79%	88,85%	88,81%
180K	MIN	102,96	89,06%	89,00%	88,99%	88,98%	88,99%	89,06%
	MAX	109,63	89,12%	89,17%	89,12%	89,14%	89,19%	89,12%
	AVG	105,08	89,09%	89,11%	89,07%	89,06%	89,11%	89,08%

Fig. 4.13 and Fig. 4.14 graphically depict the results of the accuracy and F_1score. In Fig. 4.15 and Fig. 4.16 – the results of *PPV*, *NPV*, *TPR*, *TNR*. When Amazon customer reviews dataset was used, our method performed better with higher *NPV* and *TPR* (see Fig. 4.16).

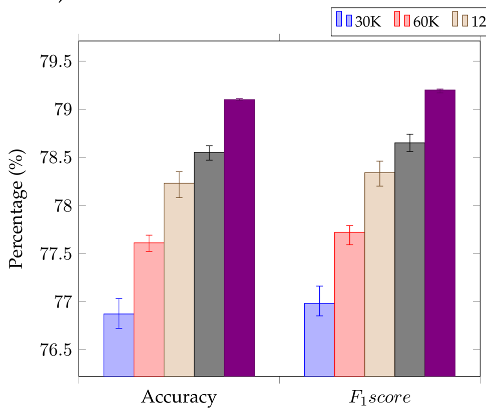

Figure 21: sentiment140 results

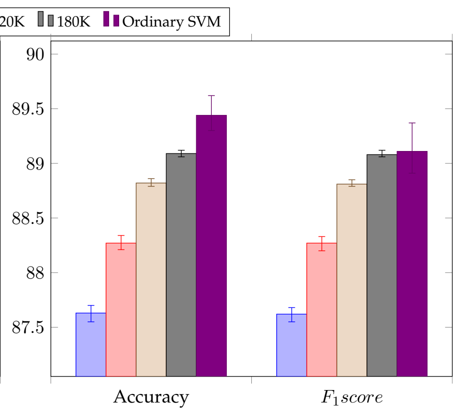

Figure 22: Amazon reviews results

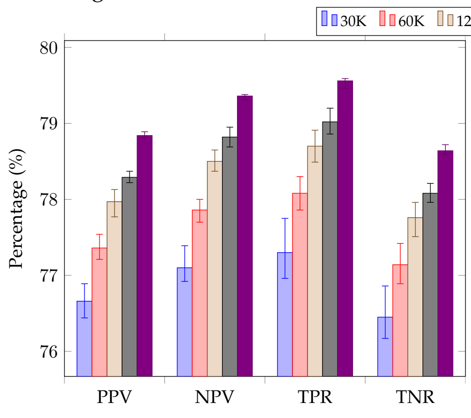

Figure 23: sentiment140 results

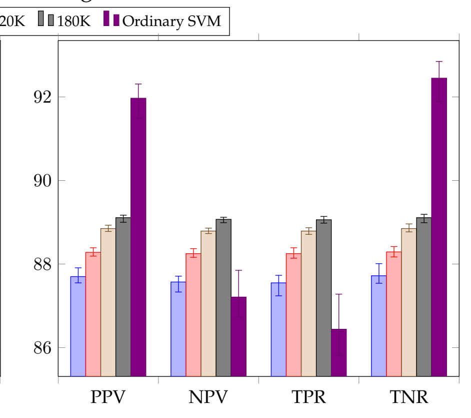

Figure 24: Amazon reviews results

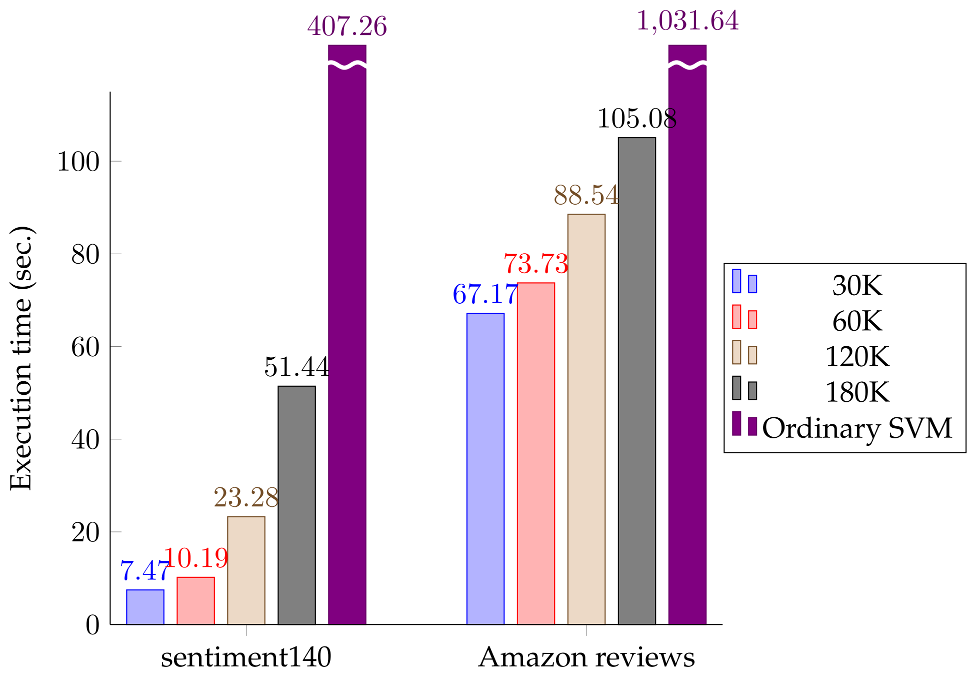

Figure 25: Execution time comparison

Fig. 4.17 compares the execution time between an ordinary SVM technique and our method, when the Stanford Twitter sentiment corpus and Amazon customer reviews datasets are used. It is important to conclude that the proposed technique outperformed an ordinary SVM technique in all cases.

The achieved results are not objective in comparison with other works as different hardware and methods (if implementations are not presented), datasets, parameters, tasks, etc. are used.

4.6 Conclusions

Training data size has significant impact on SVM classification speed. Properly selected features can improve executing time with no losing or similar accuracy. This thesis proposes the method to improve SVM classification speed by reducing the training set. The experimental results show that our method is characterized by significantly higher speed than an ordinary SVM. Although typical use of SVM is still superior in terms of accuracy or other tested metrics, the difference is not significant. However, the proposed technique outperformed ordinary SVM when applied to Amazon customer reviews dataset with higher *NPV* and *TPR*. Execution time for the Stanford Twitter sentiment corpus dataset was 7.9-54x faster, and for Amazon customer reviews dataset – 9.8-15.35x faster than an ordinary SVM.

5 Conclusions

The results of our proposed hybrid method has proved its efficiency in sentiment polarity classification tasks when large scale datasets are used.

References

[AAM+14] Silvio Amir, Miguel B Almeida, Bruno Martins, Joao Filgueiras, and Mário J Silva. Tugas: Exploiting unlabelled data for twitter sentiment analysis. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 673–677, 2014.

[AAMA17] Munir Ahmad, Shabib Aftab, Syed Shah Muhammad, and Sarfraz Ahmad. Machine Learning Techniques for Sentiment Analysis: A Review. *Int. J. Multidiscip. Sci. Eng*, 8(3):27–32, 2017.

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

[AJBV16] Akshay Amolik, Niketan Jivane, Mahavir Bhandari, and M Venkatesan. Twitter sentiment analysis of movie reviews using machine learning techniques. *International Journal of Engineering and Technology*, 7(6):1–7, 2016.

[AM+16] Basant Agarwal, Namita Mittal, et al. *Prominent feature extraction for sentiment analysis*. Springer, 2016.

[ASHP18] Asad Abdi, Siti Mariyam Shamsuddin, Shafaatunnur Hasan, and Jalil Piran. Machine learning-based multi-documents sentiment-oriented summarization using linguistic treatment. *Expert Systems with Applications*, 109:66–85, 2018.

[BDDN14] Sagar Bhuta, Avit Doshi, Uehit Doshi, and Meera Narvekar. A review of techniques for sentiment analysis Of Twitter data. In *Issues and challenges in intelligent computing techniques (ICICT), 2014 international conference on*, pages 583–591. IEEE, 2014.

[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.

[BM09] Erik Boiy and Marie-Francine Moens. A machine learning approach to sentiment analysis in multilingual Web texts. *Information retrieval*, 12(5):526–558, 2009.

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

[BS15] Kusum Kumari Bharti and Pramod Kumar Singh. Hybrid dimension reduction by integrating feature selection with feature extraction method for text clustering. *Expert Systems with Applications*, 42(6):3105–3114, 2015.

[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.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[CZ18] Yuling Chen and Zhi Zhang. Research on text sentiment analysis based on CNNs and SVM. In *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, 2018.

[Dam10] Robertas Damaševičius. Optimization of SVM parameters for recognition of regulatory DNA sequences. *Top*, 18(2):339–353, 2010.

[Daw76] Richard Dawkins. The selfish gene Oxford university press. *New York*, 1976.

[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.

[DG97] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[DG15] Paulius Danenas and Gintautas Garsva. Selection of support vector machines based classifiers for credit risk domain. *Expert Systems with Applications*, 42(6):3194–3204, 2015.

[DGZC17] Fang Deng, Su Guo, Rui Zhou, and Jie Chen. Sensor multifault diagnosis with improved support vector machines. *IEEE Transactions on Automation Science and Engineering*, 14(2):1053–1063, 2017.

[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.

[Dor92] Marco Dorigo. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*, 1992.

[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.

[EHG05] Emad Elbeltagi, Tarek Hegazy, and Donald Grierson. Comparison among five evolutionary-based optimization algorithms. *Advanced engineering informatics*, 19(1):43–53, 2005.

[EL03] Muzaffar M Eusuff and Kevin E Lansey. Optimization of water distribution network design using the shuffled frog leaping algorithm. *Journal of Water Resources planning and management*, 129(3):210–225, 2003.

[ELP06] Muzaffar Eusuff, Kevin Lansey, and Fayzul Pasha. Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. *Engineering optimization*, 38(2):129–154, 2006.

[Eng08] Andries P Engelbrecht. Computational Intelligence: An Introduction. 2008.

[FCH$^+$08] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[FZYL14] Yixiang Fang, Haijun Zhang, Yunming Ye, and Xutao Li. Detecting hot topics from Twitter: A multiview approach. *Journal of Information Science*, 40(5):578–593, 2014.

[Gar10] Poonam Garg. A Comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm. *arXiv preprint arXiv:1004.0574*, 2010.

[GB15] Li Guo and Samia Boukir. Fast data selection for SVM training using ensemble margin. *Pattern Recognition Letters*, 51:112–119, 2015.

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

[GCB$^+$05] Hans P Graf, Eric Cosatto, Leon Bottou, Igor Dourdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. In *Advances in neural information processing systems*, pages 521–528, 2005.

[GH13] Quanquan Gu and Jiawei Han. Clustered support vector machines. In *Artificial Intelligence and Statistics*, pages 307–315, 2013.

[GLL$^+$17] Jie Gan, Ang Li, Qian-Lin Lei, Hao Ren, and Yun Yang. K-means based on active learning for support vector machine. In *Computer and Information*

Science (ICIS), 2017 IEEE/ACIS 16th International Conference on, pages 727–731. IEEE, 2017.

[GY14] Geetika Gautam and Divakar Yadav. Sentiment analysis of twitter data using machine learning approaches and semantic analysis. In *Contemporary computing (IC3), 2014 seventh international conference on*, pages 437–442. IEEE, 2014.

[HCAV18] Thi Thom Hoang, Ming-Yuan Cho, Mahamad Nabab Alam, and Quoc Tuan Vu. A novel differential particle swarm optimization for parameter selection of support vector machines for monitoring metal-oxide surge arrester conditions. *Swarm and Evolutionary Computation*, 38:120–126, 2018.

[HG89] JH Holland and D Goldberg. Genetic algorithms in search, optimization and machine learning. *Massachusetts: Addison-Wesley*, 1989.

[HHH98] Randy L Haupt, Sue Ellen Haupt, and Sue Ellen Haupt. *Practical genetic algorithms*, volume 2. Wiley New York, 1998.

[HL04] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[Hol75] John H Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, 1975.

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

[HSZC09] Xia Hu, Nan Sun, Chao Zhang, and Tat-Seng Chua. Exploiting internal and external semantics for the clustering of short texts using world knowledge. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 919–928. ACM, 2009.

[Hun07] John D Hunter. Matplotlib: A 2D graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[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.

[KDG17] Konstantinas Korovkinas, Paulius Danėnas, and Gintautas Garšva. SVM and Naïve Bayes Classification Ensemble Method for Sentiment Analysis. *Baltic Journal of Modern Computing*, 5(4):398–409, 2017.

[KDG18] Konstantinas Korovkinas, Paulius Danėnas, and Gintautas Garšva. SVM Accuracy and Training Speed Trade-Off in Sentiment Analysis Tasks. In *International Conference on Information and Software Technologies*, pages 227–239. Springer, 2018.

[KG15] Monisha Kanakaraj and Ram Mohana Reddy Guddeti. NLP based sentiment analysis on Twitter data using ensemble classifiers. In *Signal processing, communication and networking (ICSCN), 2015 3rd international conference on*, pages 1–5. IEEE, 2015.

[KG18] Konstantinas Korovkinas and Gintautas Garsva. Selection of Intelligent Algorithms for Sentiment Classification Method Creation. In *Proceedings of the International Conference on Information Technologies, Vol-2145*, pages 152–157. CEUR, 2018.

[KK13] Jayashri Khairnar and Mayura Kinikar. Machine learning algorithms for opinion mining and sentiment classification. *International Journal of Scientific and Research Publications*, 3(6):1–6, 2013.

[KKK$^+$13] Jurgita Kapočiut, Algis Krupavičius, Tomas Krilavičius, et al. A Comparison of Approaches for Sentiment Classification on Lithuanian Internet Comments. In *Proceedings of the 4th Biennial International Workshop on Balto-Slavic Natural Language Processing*, pages 2–11, 2013.

[KMM$^+$14] Olga Kurasova, Virginijus Marcinkevicius, Viktor Medvedev, Aurimas Rapecka, and Pavel Stefanovic. Strategies for big data clustering. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, pages 740–747. IEEE, 2014.

[KS$^+$16] Vishal Kharde, Prof Sonawane, et al. Sentiment analysis of twitter data: a survey of techniques. *arXiv preprint arXiv:1601.06971*, 2016.

[KSTA15] Olga Kolchyna, Tharsis TP Souza, Philip Treleaven, and Tomaso Aste. Twitter sentiment analysis: Lexicon method, machine learning method and their combination. *arXiv preprint arXiv:1507.00955*, 2015.

[LG05] Hansheng Lei and Venu Govindaraju. Speeding up multi-class SVM evaluation by PCA and feature selection. *Feature Selection for Data Mining*, 72, 2005.

[lid01] Natural language processing, author=Liddy, Elizabeth D. 2001.

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

[LL18] Sisi Liu and Ickjai Lee. Email Sentiment Analysis Through k-Means Labeling and Support Vector Machine Classification. *Cybernetics and Systems*, 49(3):181–199, 2018.

[LM01] Yuh-Jye Lee and Olvi L Mangasarian. RSVM: Reduced support vector machines. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, pages 1–17. SIAM, 2001.

[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.

[LSD12] Chenliang Li, Aixin Sun, and Anwitaman Datta. Twevent: segment-based event detection from tweets. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 155–164. ACM, 2012.

[LYC$^+$10] Zitao Liu, Wenchao Yu, Wei Chen, Shuran Wang, and Fengyi Wu. Short text feature selection for micro-blog mining. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1–4. IEEE, 2010.

[LZ17] Jie Liu and Enrico Zio. SVM hyperparameters tuning for recursive multi-step-ahead prediction. *Neural Computing and Applications*, 28(12):3749–3763, 2017.

[M$^+$67] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[MBW14] Oliver Meyer, Bernd Bischl, and Claus Weihs. Support vector machines on large data sets: Simple parallel approaches. In *Data Analysis, Machine Learning and Knowledge Discovery*, pages 87–95. Springer, 2014.

[MDJB12] I Martisius, R Damasevicius, V Jusas, and D Birvinskas. Using higher order nonlinear operators for SVM classification of EEG data. *Elektronika ir Elektrotechnika*, 119(3):99–102, 2012.

[MF97] Peter Merz and Bernd Freisleben. A genetic local search approach to the quadratic assignment problem. In *Proceedings of the 7th international conference on genetic algorithms*, pages 1–1, 1997.

[MF11] Diana Maynard and Adam Funk. Automatic detection of political opinions in tweets. In *Extended Semantic Web Conference*, pages 88–99. Springer, 2011.

[MFWH16] Xue Mao, Zhouyu Fu, Ou Wu, and Weiming Hu. Fast kernel SVM training via support vector identification. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 1554–1559. IEEE, 2016.

[MHK14] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, 2014.

[MMG$^+$18] Rita Morisi, David Neil Manners, Giorgio Gnecco, Nico Lanconelli, Claudia Testa, Stefania Evangelisti, Lia Talozzi, Laura Ludovica Gramegna, Claudio Bianchini, Giovanna Calandra-Buonaura, et al. Multi-class parkinsonian disorders classification with quantitative MR markers and graph-based features using support vector machines. *Parkinsonism & related disorders*, 47:64–70, 2018.

[MMS02] Daniel Merkle, Martin Middendorf, and Hartmut Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE transactions on evolutionary computation*, 6(4):333–346, 2002.

[MS18] R Manikandan and R Sivakumar. Machine learning algorithms for text-documents classification: A review. *Machine learning*, 3(2), 2018.

[MSZ$^+$03] Holger R Maier, Angus R Simpson, Aaron C Zecchin, Wai Kuan Foong, Kuang Yeow Phang, Hsin Yeow Seah, and Chan Lim Tan. Ant colony optimization for design of water distribution systems. *Journal of water resources planning and management*, 129(3):200–209, 2003.

[MTV17] Sara Mourad, Ahmed Tewfik, and Haris Vikalo. Data subset selection for efficient SVM training. In *Signal Processing Conference (EUSIPCO), 2017 25th European*, pages 833–837. IEEE, 2017.

[NKT14] Manu Nandan, Pramod P Khargonekar, and Sachin S Talathi. Fast SVM training using approximate extreme points. *The Journal of Machine Learning Research*, 15(1):59–98, 2014.

[NNVP14] Dai Quoc Nguyen, Dat Quoc Nguyen, Thanh Vu, and Son Bao Pham. Sentiment classification on polarity reviews: an empirical study using rating-based features. In *Proceedings of the 5th workshop on computational approaches to subjectivity, sentiment and social media analysis*, pages 128–135, 2014.

[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.

[OGN17] Haidar Osman, Mohammad Ghafari, and Oscar Nierstrasz. Hyperparameter optimization to improve bug prediction accuracy. In *Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), IEEE Workshop on*, pages 33–38. IEEE, 2017.

[OMC14] Alvaro Ortigosa, José M Martín, and Rosa M Carro. Sentiment analysis in Facebook and its application to e-learning. *Computers in human behavior*, 31:527–541, 2014.

[OSO12] Ozer Ozdikis, Pinar Senkul, and Halit Oguztuzun. Semantic expansion of tweet contents for enhanced event detection in twitter. In *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*, pages 20–24. IEEE, 2012.

[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.

[PM17] Tomas Pranckevičius and Virginijus Marcinkevičius. Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification. *Baltic Journal of Modern Computing*, 5(2):221, 2017.

[PP10] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326, 2010.

[PVG$^+$11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[RAD18] Abhilasha Singh Rathor, Amit Agarwal, and Preeti Dimri. Comparative Study of Machine Learning Approaches for Amazon Reviews. *Procedia Computer Science*, 132:1552–1561, 2018.

[RR15] Kumar Ravi and Vadlamani Ravi. A survey on opinion mining and sentiment analysis: tasks, approaches and applications. *Knowledge-Based Systems*, 89:14–46, 2015.

[RWL18] Rui Ren, Desheng Dash Wu, and Tianxiang Liu. Forecasting Stock Market Movement Direction Using Sentiment Analysis and Support Vector Machine. *IEEE Systems Journal*, 2018.

[S$^+$16] Zubin A Sunkad et al. Feature Selection and Hyperparameter Optimization of SVM for Human Activity Recognition. In *Soft Computing & Machine Intelligence (ISCMI), 2016 3rd International Conference on*, pages 104–109. IEEE, 2016.

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

[SLC17] Shiliang Sun, Chen Luo, and Junyu Chen. A review of natural language processing techniques for opinion mining systems. *Information Fusion*, 36:10–25, 2017.

[SW11] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[TAR16] Abinash Tripathy, Ankit Agrawal, and Santanu Kumar Rath. Classification of sentiment reviews using n-gram machine learning approach. *Expert Systems with Applications*, 57:117–126, 2016.

[TG18] Antonela Tommasel and Daniela Godoy. Short-text feature construction and selection in social media data: a survey. *Artificial Intelligence Review*, 49(3):301–338, 2018.

[THGL13] Jiliang Tang, Xia Hu, Huiji Gao, and Huan Liu. Unsupervised feature selection for multi-view data in social media. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 270–278. SIAM, 2013.

[TKF15] Jevgenij Tichonov, Olga Kurasova, and Ernestas Filatovas. Vaizdų klasifikavimas pagal suspaudimo algoritmo poveikį jų kokybei. *Informacijos Mokslai/Information Sciences*, 73, 2015.

[TL12] Jiliang Tang and Huan Liu. Feature selection with linked data in social media. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 118–128. SIAM, 2012.

[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, 2001.

[TRJ09] Hastie Trevor, Tibshirani Robert, and Friedman JH. The elements of statistical learning: data mining, inference, and prediction, 2009.

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

[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.

[TWY$^+$14] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1555–1565, 2014.

[TXW$^+$14] Guoyu Tang, Yunqing Xia, Weizhi Wang, Raymond Lau, and Fang Zheng. Clustering tweets usingWikipedia concepts. In *LREC*, pages 2262–2267. Citeseer, 2014.

[VVC$^+$11] Sudha Verma, Sarah Vieweg, William J Corvey, Leysia Palen, James H Martin, Martha Palmer, Aaron Schram, and Kenneth Mark Anderson. Natural Language Processing to the Rescue? Extracting "Situational Awareness" Tweets During Mass Emergency. In *ICWSM*, pages 385–392. Barcelona, 2011.

[Ž08] Krista Rizman Žalik. An efficient k′-means clustering algorithm. *Pattern Recognition Letters*, 29(9):1385–1391, 2008.

[WG15] Yun Wan and Qigang Gao. An ensemble sentiment classification system of twitter data for airline services analysis. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, pages 1318–1325. IEEE, 2015.

[WHYL12] Bing-kun Wang, Yong-feng Huang, Wan-xia Yang, and Xing Li. Short text classification based on strong feature thesaurus. *Journal of Zhejiang University SCIENCE C*, 13(9):649–659, 2012.

[WLL$^+$14] Senzhang Wang, Zhoujun Li, Chunyang Liu, Xiaoming Zhang, and Haijun Zhang. Training data reduction to speed up SVM training. *Applied intelligence*, 41(2):405–420, 2014.

[WZC$^+$18] Shengnan Wang, Xiaoyong Zhang, Yijun Cheng, Fu Jiang, Wentao Yu, and Jun Peng. A Fast Content-Based Spam Filtering Algorithm with Fuzzy-SVM and K-means. In *Big Data and Smart Computing (BigComp), 2018 IEEE International Conference on*, pages 301–307. IEEE, 2018.

[YIH17] Alireza Yousefpour, Roliana Ibrahim, and Haza Nuzly Abdel Hamed. Ordinal-based and frequency-based integration of feature selection methods for sentiment analysis. *Expert Systems with Applications*, 75:80–93, 2017.

[YLY$^+$13] Yukai Yao, Yang Liu, Yongqing Yu, Hong Xu, Weiming Lv, Zhao Li, and Xiaoyun Chen. K-SVM: An Effective SVM Algorithm Based on K-means Clustering. *JCP*, 8(10):2632–2639, 2013.

[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.