Bachelor thesis

Support Vector Machines via Multilevel Label Propagation

Matthias Schmitt

Date: 25th June 2018

Supervisors: Prof. Dr. Peter Sanders
M. Sc. Sebastian Schlag
Dr. Christian Schulz

Institute of Theoretical Informatics, Algorithmics
Department of Informatics
Karlsruhe Institute of Technology

Abstract

The time complexity of support vector machines (SVM) prohibits training on huge data sets with millions of samples. Multilevel SVMs were developed to allow for time efficient training on huge data sets. In this thesis we present a new faster multilevel support vector machine framework which utilizes the near-linear time label propagation algorithm for the construction of the problem hierarchy. While regular SVM perform the entire training in one - time consuming - optimization step, multilevel SVMs first build a hierarchy of problems decreasing in size that resemble the original problem and then train an SVM model for each hierarchy level benefiting from the solved models of previous levels. Our experiments on a large number of benchmark data sets show that our framework (KaMLSVM), when compared to the previous best multilevel SVM, achieves a speed-up of more that two on almost all data stet and up to 20 on large data sets with many features.

Abstract (de)

Die Zeitkompletxität von Support Vector Machinen (SVM) verbietet das Training auf rießigen Datensätzen mit Millionen von Datenpunkten. Multilevel SMVs wurden entwickelt um das Trainieren auf diesen rießigen Datensätzen zeitlich möglich zu machen. In dieser Arbeit präsentieren wir eine neuartige Multilevel Support Vector Machine, welche für den Aufbau der Problemhierarchie den fast-linearzeit label propagation Algorithmus verwendet. Während einfache Support Vector Machinen das gesamte Training in einem - zeitaufwendigen - Optimierungsschritt lösen, baut eine Multilevel Support Vector Machine zuerst eine Hierarchie kleiner werdender Problemen auf, die das Ursprungsproblem abstrahieren. Dannach wird das Training auf jedem Level der Hierarchie durchgeführt und durch die erlernten Eigenschaften vorheriger Level beschleunigt. Unsere Ergebnisse auf einer Vielzahl von Benchmark-Datensätzen zeigen, dass unser Framework, im Vergleich zur bisher schnellsten Multilevel SVM, einen Speed-Up von mehr als zweit und auf sehr großen Datensätzen mit vielen Features sogar bis zu 20 erreicht.

Acknowledgments

First and foremost I want to express my gratitude towards my supervisors Sebastian Schlag and Christian Schulz for introducing me to such an interesting topic and giving me every possible guidance while encouraging me to explore and develop my own approach towards the topic. Thank you for providing such a great working atmosphere.

I want to thank all my friends and family who supported my during my thesis be it by giving advice and motivation or proof reading.

Finally I thank Ehsan Sadrfaridpour for making the source code of the mlsvm-AMG public and the delightful email conversation about the framework which led me to contributing back to the project. This is the joy of free software and a non negligible reason why I enjoyed working on the thesis and programming in general.

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Ort, den Datum

Contents

1 Introduction

It lies in the human nature to collect all possible kinds of data with the intention to better understand the world we live in. We collect far more data than we can ever process knowing that in the future we might be able to use that data with advanced methods and new technologies. There are many reasons for collecting and evaluating data in the digital age. Scientists, e.g. meteorologists, biologists or astrophysicists collect huge amounts of data to build better models abstracting their fields and companies use their customers data to increase their profit. In all of those cases immense amounts of data are available and the need for technology that can help evaluating the collected data is tremendous. With the technical advances in electric manufacturing allowing for faster computers by the year, mankind's dream of building a machine that can help understanding our environment and even find previously unthought-of connections and correlations is already taking shape.

Machine learning is an interesting and promising sub-field of computer science, that builds and studies software which is able to learn from and understand the vast amounts of data that are available in the information age. The objective of machine learning is to develop algorithms that when given input data, can learn from the data and make predictions. A concrete example of a machine learning task is the classification problem. In a classification problem we have unlabeled data points, e.g information about the financial situation of a person, and want to put them into the right class out of a finite number of classes, e.g creditworthy and not-creditworthy. Such tasks were historically done by experts in the specific field and required lots of time and man power. We can easily see why a bank would like to automate the process of checking for credit-worthiness because a machine could do this job faster, more cost efficient and hopefully less error prone. With the advances in machine learning we now have algorithms that are able to do the work previously thought of being a exclusive competence of humans. The training is done by presenting the algorithms with labeled example data points, from which the algorithm has to learn the underlying structure and later is able to rightly classify unlabeled data. Classification is an important part of machine learning and large margin classifiers can be used for that purpose.

Large margin classifiers separate the classes of the classification problem in such a way that they are able to give the distance to a decision boundary. Support Vector Machines (SVMs) are the most well known large margin classifiers. They solve a convex optimization problem and use a maximum-margin hyperplane to separate classes. SVM are known to show good performance when trained to solve a classification problem, but to achieve high quality prediction results parameter fitting is required. The process of finding the right parameters for a specific problem is called *model selection* and it is the work intensive part

of machine learning with SVM. Since the time complexity for solving the optimization problem underlying SVMs is between $O(n^2)$ and $O(n^3)$ and every set of parameters requires the training of a new SVM model, SVM training becomes a problem on data sets that have hundreds of thousand or even millions of training points. The model selection as depicted above is highly parallelizable, but the time complexity is still huge and unfeasible for very large data sets. Other model selection approaches for SVM make some training results reusable so that the training time complexity shrinks but the parallelization is more difficult. Even highly optimized SVM algorithms still can not cope well with data sets of hundreds of thousands of data points.

Huge data sets that are imbalanced, meaning that the classes have unequal size, are of great interest for the machine learning field. An example would be a medical data set where a data point consists of different medical reading of a person and the labels tell whether the person is ill and not ill. A data set like this might have many features, can be of enormous size and very imbalanced due to the fact that illnesses occur less often when the general population is considered. Different machine learning algorithms apply varying techniques to make training on those large data sets possible.

One technique to cope with large data sets is random sampling. As the name suggests a random subset of the input data is selected and used for the training. Problems can arise with infrequently occurring important data which are under represented in the sample. This technique does not take the underlying structure of the data into account. Because random sampling only reflects the distribution of training data we could miss significant regions of the testing data [29]. To train an SVM on large problems a more sophisticated approach is needed.

Another conclusion that we can deduce from the fact that huge data sets are often highly imbalanced is that it is important for us to not wrongly classify the points in the smaller class. Those data points, in the context of medical the signs of an illness, are the important characteristic we want our algorithm to learn. So we take great care that the classification algorithm is not only right on the larger class but has well-balanced prediction quality.

1.1 Motivation

A new paradigm which was recently introduced to SVMs is the multilevel paradigm. This paradigm was proposed for SVM machine learning to tackle the scalability problem of SVMs [22]. Already wildly used in, e.g. graph partitioning, a multilevel framework first builds a hierarchy of the problem. Each level of the hierarchy is a problem that decreases in size and is it similar to the original problem. Then a regular SVM is trained on the coarsest problem which is much more feasible than training on the original problem because the problem is smaller an the scalability problem of the SVM algorithm does not come into account. The performance of this model is most of the time already quite good, but to achieve even better results the complete hierarchy is used. Insights from the initial training

are used when projecting the problem upwards in the hierarchy and refining the model to better fit the original problem while only gradually increasing the size of the data the SVM is trained on.

The multilevel paradigm is a promising new research path for SVM because it cuts down on computation time while being comparable and often better in terms of prediction quality on large data sets. Because the multilevel SVM is a novel technique there is a lot of room for improvement which is what we are doing with this work.

1.2 Contribution

We improve the multilevel SVM concept introduced by Talayeh Razzaghi and Ilya Safro [22] by using a different strategy for building the hierarchy. We apply the new near-linear clustering algorithm label propagation proposed by Raghavan et al. [21] instead of the coarsening strategies used by the original authors. While the previously researched strategies worked well on large data sets with few features, number of properties or characteristics representing some object, our framework is less affected by the number of features of the data set. Our experiments indicate that our approach is much faster on large data sets with a great number of features, with a speed-up of up to 20, and achieves a speed up of two and more on data sets with a manageable amount of features. The prediction quality is on par with the quality of previous multilevel SVMs which were its self an improvement over regular SVMs.

1.3 Structure of Thesis

We first introduce necessary concepts and notations and summarize related work in Chapter 2 and Chapter 3. The main part of this thesis is Chapter 4, in which we explain our framework in detail. In Chapter 5 we provide experimental results on a variety of different data sets and compare our framework with other SVM algorithms. Finally, we conlude with Chapter 6 and present some ideas for future work.

2 Fundamentals

In this chapter, we give a very general introduction to machine learning, supervised learning and the problem of classification. We explain the concepts behind support vector machines (SVM) and their approach to the classification problem. We further introduce some basic graph notation and describe the general idea of the multilevel paradigm which is applied by our algorithm.

2.1 Machine Learning

> Machine learning is the task of developing algorithms that can learn from and make predictions on data [16].

Because machine learning is a very broad sub-field of computer science, there are many different problems that belong to the field of machine learning. Algorithms that are considered machine learning algorithms work on input data which is observed data that describes the problem either by example or as a discrete flow of information. The input data can be nearly anything from pictures or audio data to vegetation information (the covtype/forest data set in our experiments). A single data point (the representation of a tree in the forest data set) has multiple features and is also called feature vector. A single feature is a individual measurable property or characteristic of a phenomenon (the tree) being observed [4]. All data points combined make up the specific data set. They have all the same features, with the exception that some features might be unknown for some data points, which is why we also talk of the feature columns of a data set. In Section 4.2 we describe techniques we used to convert features which are not in a computer readable format into real values most algorithms need to operate. The objectives of machine learning applications are vastly different. In a picture or video we might want to recognize faces or all kinds of different objects at the same time. Listening to a wake word in an audio stream is a machine learning objective which is interesting for smart home assistants. In the previously mentioned forest data set we want to predict the tree species by vegetation information like height, soil type etc. Since we are working with support vector machines we only describe the machine learning concepts necessary to the classification problem SVMs are used to solve. A comprehensive introduction into machine learning would go beyond the scope of this thesis and can be found in the introductory literature [7].

Supervised Learning. In supervised learning the training data is a finite number of example inputs and their desired output, e.g. the vegetation information of a tree (training) and the type of that tree (output). Labeling the data often requires a skilled human agent. The input data is presented to an algorithm, which learns a function that maps from the input space to the desired output space. The learned mapping is then used to determine the output for unseen instances. The algorithm's quality is defined by the correctness of the determined class labels for unseen instances. This is often evaluated by splitting the available labeled data into two sets, a training set on which the training in conducted and a test set on which the algorithms performance is evaluated. For further information about our evaluation procedure see Section 4.6 and Section 4.5.

On the contrary unsupervised learning works by only feeding an algorithm data points without any attached desired output and letting the algorithm find and describe hidden structure in the data.

Classification. A common supervised learning task in the machine learning field is the classification problem. The premise is that we want to associate unlabeled data points with a specific class out of a finite number of classes (cf. in the forest data set the possible vegetation types). We do so by training an algorithm on a set of training examples $X_1 \dots X_n$ with associated labels $y_1 \dots y_n$. Once the algorithm is trained with the training set, we wish to predict the class/label y_{n+1} for a new data point X_{n+1}. Classification is always a supervised learning problem, because the possible classes are set by the training data. A special case of classification is binary classification, where there are only two possible classes which are isomorphic to $\{+1, -1\}$. We later show how to do multiclass classification with SVM by training SVM on distinct binary classification problems.

Clustering. Cluster analysis or clustering is a unsupervised learning task similar to the classification problem. In both tasks one is interested in the class/cluster of a data point. In the classification task we have training examples from which the classes are derived. The data points in a classes share some kind of similarity and differ from the points in other classes. In a clustering problem the training data is unlabeled and no classes are given. It is the task of the algorithm to find clusters, i.e. groups that divide the data such that similar points end up in the same cluster. In our framework we use a clustering algorithm called label propagation [21], which we explain in Section 3.2, to represent a cluster by a single data point and later reduce the size of the input data for the SVM training.

2.2 Support Vector Machines

Support vector machines (SVMs) are supervised learning models that can be used for classification. SVMs are studied very thoroughly and are one of the most well-known machine learning algorithms [23]. As depicted in the introduction, SVM are large margin classifiers

and work by finding a hyperplane separating the classes and then use the hyperplane to decide the class for a new data point. In this section we explain the mathematical concepts behind support vector machines in detail and show some tricks applied to SVM to make them more adaptable.

Formal Description. We will give a formal description of the optimization problem which is solved in binary linear classification SVMs. It is important to mention that our proposed framework does not depend on how the optimization algorithm achieves the result and alternative formulations can be used equally well in our architecture.

Given a set $\mathcal{I}$ of n data points x_i with corresponding labels $y_i \in \{+1, -1\}$, we get the minority class $\mathbf{C}^+$ consisting of all data points with positive $(+1)$ label $|\mathbf{C}^+| = n^+$ and the other data points accordingly create the majority class $\mathbf{C}^-$ with $|\mathbf{C}^-| = n^-$ and $n = n^+ + n^-$. Every training data point x_i is interpreted as a d dimensional vector in $\mathbb{R}^d$ and the SVM finds $d-1$ dimensional hyperplanes separating the two classes. The best separating plane is the one furthest away from both classes, i.e. the one with the largest margin between the two classes, hence the name large margin classifier.

The standard SVM formulation with a linear kernel is given by the following constrained optimization problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(w \cdot x_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

The hyperplane $f(x) = w \cdot x + b$ with maximum margin is calculated by solving for the parameters w and b. This problem is called the primal problem and is often indirectly solved by solving the dual problem. The variables $\xi_i = \max(0, 1 - y_i(w \cdot x_i - b))$, also called slack variables, introduce the hinge loss function. It is used to extend SVMs to cases, where the data is not linearly separable, meaning there is no hyperplane separating $\mathbf{C}^+$ and $\mathbf{C}^-$. It allows for misclassification but ensures that every x_i lies on the correct side of the margin. Since it is part of the term that is to minimize, it also penalizes misclassification. This method is known as soft margin extension to the SVM. The parameter $C > 0$ controls the magnitude of the penalization.

New data points are classified by the sign of the hyperplane equation $h(x) = \text{sign}(w \cdot x + b)$. In simpler terms the hyperplane splits the input space into two and puts data points above the plane in $\mathbf{C}^+$ and below in $\mathbf{C}^-$.

Figure 2.1 shows a non linear separable example where the slack variables are important and allow the SVM to find a separating hyperplane in a not linearly separable problem. We see that $\frac{2}{\|w\|}$ is the distance between the margins and that in order to find the maximum separating hyperplane we need to minimize $\|w\|$. An easy-to-see but important consequence of the way the hyperplane is calculated is that the maximum margin hyperplane is only

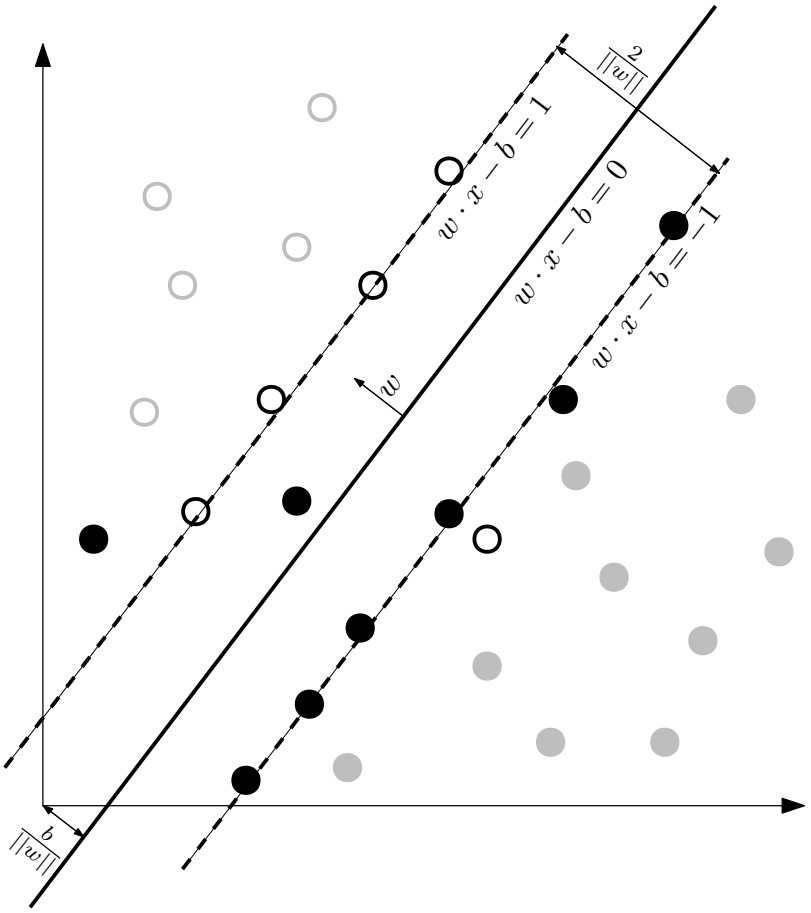

Figure 2.1: A non separable binary classification problem solved with SVM and slack variables.

determined by those x_i which lie nearest to it. Those are the data points whose removal would result in a change of the hyperplane. These vectors are called support vectors (SVs) and in Figure 2.1 the support vectors are colored black while the non support vectors, the vectors whose removal would not change the hyperplane, are colored in gray. We need this property in our frameworks uncoarsening step explained in Section 4.5.

Kernel Trick. The *kernel trick*, originally proposed by Aizerman et al. [1] and applied to SVMs by Boser et al. [5], is used to calculate a reasonable separating hyperplane for problems that are not linearly separable. Often times the data is not linear separable in the euclidean space and even the soft margin approach can not find a good hyperplane. A key observation is that when the original problem is transformed to other higher level spaces, better separating hyperplanes exist. A simple implementation would first transform the problem, and therefore every data point, into the new space and then calculate the hyperplane by solving the optimization problem. This has the disadvantage that the trans-

formation is time and space consuming and because the right target space is not known beforehand. The kernel trick allows for a better technique. To avoid costly transformation and calculations in the target space we use a kernel function to only compute the dot product in the feature space The dot product is the only information the SVM optimization problem needs in the feature space. We do not even have to know the space into which we are projecting, we only need the inner product.

With this trick linear learning algorithms such as SVMs can learn nonlinear functions and nonlinear decision boundaries. The algorithm fits a maximum-margin hyperplane in a transformed feature space which is nonlinear in the original problem space. Figure 2.2[1] is a visualization of the idea of the kernel trick. The training points are mapped to a 3-dimensional space where a separating hyperplane, separating red and blue points, can be easily found.

The Gaussian kernel function (radial basis function, RBF) defined as

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \quad \gamma \geq 0$$

is one of the most commonly used kernel functions and is known to be generally reliable when no additional assumptions about the data are known [26].

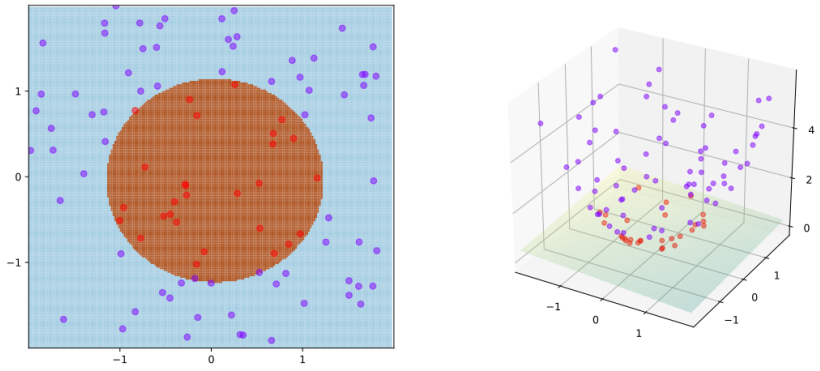

Figure 2.2: The training points are mapped into a higher dimensional space by $\phi((a, b)) = (a, b, a^2 + b^2)$ and the kernel function is $k(x, y) = x \cdot y + x^2 y^2$. (created by Shiyu Ji)

Parameter Tuning. Prior to training a SVM model we have to choose parameters of the algorithm. The parameters for SVM learning are C and various kernel parameters. In the case of the Gaussian Kernel γ is the only additional parameter. The optimization problem with fixed parameters is convex which means, that we will find the maximum margin

[1] https://commons.wikimedia.org/wiki/File:Kernel_trick_idea.svg

hyperplane without getting stuck in local maxima. The parameters are highly dependent on the specific problem and parameter fitting is required to set optimal or near optimal parameters for a concrete problem. This is necessary to achieve good classification results with SVMs. The parameter search is also called model selection and is a general problem not only found in the context of SVMs. We evaluate several different approaches towards this task and described our results in Section 4.4.

Multi-class Classification. In this work we only study binary classification problems, but the concepts can be extended to support multi-class classification where the y_i are not restricted to $\{+1, -1\}$. There are different concepts of SVM that support multiple classes. One idea is to rearrange the optimization problem to allow for multiple labels directly [8]. Another concept is to train independent one-versus-rest binary SVMs and choose the class which classifies the test datum with the greatest margin to get a multi-class SVM [17]. To allow for multi-class classification in our framework we could either use a capable SVM version as our underlying SVM solver or build a multi-class classifier ourselves by combined several multilevel binary classifiers.

2.3 Graphs

A directed weighted Graph $G = (V, E, c, w)$ is a vertex set $V = [1...n]$ and an edge set $E \subseteq V \times V$ with a vertex weight function $c : V \to \mathbb{R}$ and an edge weight function $w : E \to \mathbb{R}$. If $e = (v, w)$ with $v, w \in V$ is an edge of G, then v is *adjacent* to w and v is *incident* to e. The neighborhood $N(v)$ of a vertex v is the set of all vertices that are adjacent to v. The *degree* of a vertex v is $deg(v) = |N(v)|$, i.e. the size of its neighborhood. The number of vertices and edges are denoted as $n = |V|$ and $m = |E|$, respectively. The *order* of G, denoted by $|G|$, is the number of vertices, $|G| = |V|$. The *size* of G, denoted by $\|G\|$, is the number of edges, $\|G\| = |E|$. A graph $H = (V', E')$ is a *subgraph* of G, denoted by $H \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$.

A (directed) path p is a finite sequence of distinct incident edges and all edges have to be directed in the same direction. Two vertices are connected if there exists a path containing both of them. A graph G is called *connected* if any two vertices are connected.

Matching. A matching in a graph is a set of edges without common vertices. It is also called independent edge set. A matching M is *maximal* in a graph $G = (V, E)$ if no edge can be added to M without destroying the matching properties, i.e. every edge in $E \setminus M$ contains a vertex that is already part of an edge of M. A *maximum matching* is a matching that contains the largest possible number of edges of all matchings. Every maximum matching is per definition maximal but not the other way around. There can be multiple maximum matchings. A matching M is a *perfect matching* of G, when every vertex of G is part of an edge in M.

Independent set An independent set I in a graph G is a set of vertices such that no two vertices are adjacent. An equivalent definition is that every edge in G has at most one endpoint in I. The same maximal and maximum terminology as used for matchings is also used for independent sets. A maximal independent set can not be expanded with a new vertex and a maximum independent set has maximum size of all possible independent sets.

Contraction. Let $G = (V, E)$ be a directed graph. A contraction of an edge $e = (v, w)$ results in a new graph $G \circ e = (V \setminus \{v, w\} \dot{\cup} z, E', c', w')$ where in E' every edge between another vertex x and v or w is replaced by the corresponding edge between x and z. The edge weight function is updated accordingly and the weight of the new node is $c'(z) = c(v) + c(w)$. We can contract a connected subset S of V by contracting edges of the component until it is a single vertex. To give an example for a contraction consider a circle $G = (V, E)$ on $V = [1..6]$ with $E = \{(x, y)|x = y + 1\} \cup (6, 1)$. $G \circ (5, 6)$ with the new vertex named 7 is a circle $G' = (V', E')$ on the vertices on the vertices $V' = [1..4] \cup \{7\}$ and $E' = \{(x, y)|x = y + 1\} \cup (4, 7), (7, 1)$.

2.4 Multilevel Framework

The multilevel paradigm originated from multigrid solvers for solving systems of linear equations [27]. It is already wildly used in the field of graph theory, e.g. graph partitioning and successful in various practical machine learning and data mining tasks such as clustering and dimensionality reduction [22]. This section describes the general idea of mulitilevel frameworks. The fairly recent application to SVM by Razzaghi et. al. is described in the Section 3.1. The multilevel term as we use it, is not to be confused with multilevel modeling and Deep Learning term used in the machine learning context of artificial neural networks.

A multilevel algorithm usually consists of tree phases, the *coarsening phase*, a phase where the *initial algorithm* is applied and the uncoarsening and *refinement phase*.

In the coarsening phase starting from a large problem, on which the initial algorithm is impractical to use, a hierarchy of problems decreasing in size is created. Each problem should resemble the original problem but be of lower-dimensionality or smaller-size or both. There are multiple ways to build this hierarchy and different algorithms for different fields but the most common way is to contract matchings or clusterings. These contractions obey the idea of the coarsening phase because they quickly reduce the size of the input problem while keeping the global structure of the input data. The contraction process is stopped when the graph is sufficiently small so that the initial algorithm can be used. In the field of SVMs two separate hierarchies for $\mathbf{C}^+$ and $\mathbf{C}^-$ are created.

In the next phase the initial algorithm is applied on the coarsest iteration. In the case of multilevel support vector machines a SVM library like LibSVM [6] is trained on the coarsest problem and we call this phase *initial training*. In other fields the specific initial

algorithm is run, e.g. a graph partitioner or another machine learning algorithm. It was not feasible to run the algorithm on the original problem but on the coarse problem it is. The quality of the initial result depends on the quality of the approximation achieved in the coarsening phase.

Afterwards, in the refinement phase, the hierarchy of smaller problems is iteratively uncontracted and at each level a local improvement algorithm is used to improve the quality of the result. In current multilevel SVMs the support vectors and optionally their neighbors are uncontracted and are used to train the SVM algorithm again using insights about the parameter search that happened in the initial training. In other graph partitioning a lot of research went into finding new refinement strategies and optimizing the existing ones [12, 3]. This is yet to be done for multilevel support vector machines.

3 Related Work

3.1 Multilevel Support Vector Machines

Independent Set Multilevel SVM. The first multilevel SVM (mlsvm) was developed by Talayeh Razzaghi and Ilya Safro [22]. Their goal was to make SVM training feasible on large data sets, by generalizing the problem recursively and building a hierarchy of problems. While several other approaches such as Yu et al. [29] already explored hierarchical SVM techniques, Razzaghi and Safro extended their work by building a complete multilevel framework. They combined the hierarchical idea of Yu et al. with a multilevel refinement phase. Instead of solving the whole training set in one optimization process, the multilevel approach is used to train on a hierarchy of the input data from coarsest to finest and use the results of the previous coarser level of the hierarchy to train on the current level, as described in Section 2.4. The hierarchy is constructed by calculating an approximate k-nearest neighbor graph which is gradually coarsened by contracting independent sets (IS). An approximate k-nearest neighbor algorithm was chosen because of the faster running time and exact k-nearest neighbor algorithms failed to show quality improvements in the classification results. In the refinement phase the support vectors are refined at multiple levels of coarseness of the data. Because of the utilization of independent sets the framework is called iterative independent set mulitilevel SVM (mlsvm-IIS).

In addition to being able to use support vector machines on huge training sets they showed that the multilevel approach has the advantage of being less sensitive to imbalanced data. On the coarsest level the data classes are of roughly equal size because the coarsening is stopped at the same size constrain for both classes individually. This helps the initial SVM algorithm since the coarsest level is not imbalanced anymore but still resembles the imbalanced original problem. To further improve on those instances they implemented the cost-sensitive SVM extension, also known as weighted SVM (WSVM). Experiments for the mlsvm-IIS results on benchmark data sets demonstrated that the approach substantially faster with no loss of quality in the performance measures, when compared the the underlying (W)SVM solver LibSVM [22]. The reduced computational time achieved by the multilevel framework is significant for training on large-scale data sets [22].

Algebraic Multigrid Multilevel SVM. Ehsan Sadrfaridpour et al. [24] improved the original mlsvm by utilizing an algebraic multigrid (AMG) multilevel scheme. Instead of independent set contraction a AMG algorithm is used in the coarsening phase. In this coars-

ening algorithm equivalent groups of nodes are grouped in subsets with the possibility of intersections. Each of the subsets correspond to a coarse point at the next coarser level. The number of coarse points a finer point can contribute to is controlled by a tuning parameter. The authors claim that "AMG coarsening generalizes the independent set and clustering based approaches leveraging a high quality coarsening and flexibility of AMG" [23]. The rest of the framework is very similar to the mlsvm-IIS framework. The separate coarsening of the two classification classes is still performed on a approximate k-nearest neighbor graph. LibSVM [6] is utilized as the underlying SVM solver and handles the initial training and the training steps in the refinement. The SVM parameters are chosen by the Uniform Design technique [15]. For the refinement local learning of the support vectors and model selection parameters is applied. We also apply most of these multilevel SVM techniques and explain them in Chapter 4.

When compared to DC-SVM [13], one of the fastest SVM, the mlsvm-AMG, demonstrates significantly better computation time than DC-SVM on almost all data sets [23]. In Chapter 5 we compare our results to the latest mlsvm-AMG results and note that comparisons of the mlsvm-AMG framework to other SVM can be transferred to our framework.

3.2 Label Propagation

The label propagation algorithm (LPA) was proposed by Raghavan et al. [21]. It is a near-linear time graph clustering algorithm. We describe the concepts of the algorithm which is used in our coarsening phase.

The algorithm works in rounds. Initially every node is in its own unique cluster and in each round it its moved to the cluster that is most dominant in its neighborhood, i.e. the cluster V_i that maximizes $|N(v) \cup V_i|$. The nodes are visited in random order every round and ties are broken randomly. The algorithm terminates when in every node is in the cluster that appears most frequently in its neighborhood or after a fix amount of rounds are performed.

In our framework we use the implementation of Meyerhenke et al. which adds a size-constraint to the clusters and is still of linear time complexity [18]. The maximum cluster size is a tuning parameter which in our use case can influence the size of the uncontractions in the refinement phase.

At the end of the algorithm we get a clustering of the graph and use it to contract the graph. Each cluster is represented by a node in the coarse graph. The node weights of the nodes in the cluster are summed up and make the node weight of the coarse node. The weight of an edge between two coarse nodes A and be B is set to the sum of the weight of edges that run between the cluster of A and cluster of B in the finer graph.

4 Multilevel Support Vector Machine via Label Propagation

In this chapter we describe our implementation of the three multilevel phases and the inner workings of our complete framework which additionally includes how we pre-process the data and what evaluation strategies we use. We begin with a simplified overview of the complete framework to provide context for the following sections, which each describe one step of our framework in detail.

4.1 Overview

Algorithm 1: Overview of our framework

```
1  preprocess data
2  build k-fold instances
3  foreach k-fold instance do
4      use a fraction of the training data as validation data
5      build a k-nearest neighbor graph for C+ and C−
6      contract the graphs recursively and build the hierarchies
7      initial training on the coarsest problem
8      while levels in the hierarchies do
9          train a SVM model on the SV of the previous level
10         evaluate on the validation data
11     use the best trained model of all levels as final model
12     evaluate the final model with the test data
13 average the results of the k-folds
```

The data on which the learning is conducted is given to our framework in one CSV (comma separated values) file. The more sophisticated format developed for LibSVM [6] is also supported. We then preprocess the data and afterwards split it into the two classes $\mathbf{C}^+$ and $\mathbf{C}^-$. In Section 4.2 we describe what preprocessing techniques we use and why they are necessary. We split our data into k parts which allows us to train our multilevel process k times, each time with a different part as test data and the other parts combined as training data. This process is called k-fold and is a common cross-validation technique used in

statistics and machine learning. It is used to more accurately predict the performance of the framework in practice. From the training data we build k-nearest neighbor graphs on which we preform the reduction of the original problem. With this technique we turn the task of generalizing the problem by finding good representatives into a graph coarsening problem. We build two k-nearest neighbor graphs, one for the minority class and one for the majority class. The next step is to coarsen the graphs recursively. This is done separately for the two graphs and each step of the recursive coarsening is saved into a hierarchy. After we have coarsened the graphs, we train an SVM model on the coarsest (the last) level of the hierarchies. How the initial training works is described in detail in Section 4.4. Once the initial model is trained we recursively uncontract the next finer hierarchy level and train a new SVM model on the support vectors of the previous SVM model. We also use insights from the parameter selection of the previous levels training process as discussed in Section 4.5. After the finest level is processed and we have trained models for every level of the hierarchy and we choose the overall best model as the final model of the complete multilevel process.

Our multilevel SVM framework is similar to the original mlsvm framework [22] we explained in Section 3.1. Key differences lie in the coarsening step Section 4.3 in which we use the label propagation algorithm instead of independent sets or algebraic multigrids. Figure 4.1 shows an overview of the mlsvm frameworks and is also a rough outline of our framework.

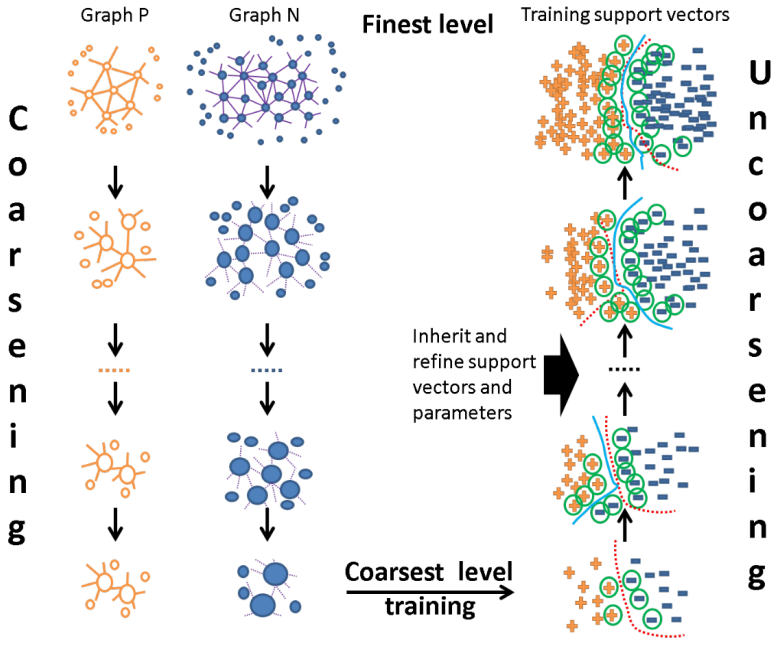

Figure 4.1: Overview of the multilevel SVM technique [23]

4.2 Data Preprocessing

Data preprocessing is an essential step of machine learning, because machine learning algorithms expect data to be in a certain format and the available data often needs to be transformed to that format. Additionally to transforming the data a scaling is performed. With the right scaling of the feature columns a much better prediction accuracy can be achieved.

Categorical Features. As described in Section 2.2, SVMs require that each data instance is represented as a vector of real numbers. Often times features such as the country someone lives in or the martial status are given as categorical variables. Categorical means that the feature can take one of a limited and fixed number of possible values. For categorical attributes an extra preprocessing step is needed to convert them into numerical data. We do so by using one hot encoding [11]. A l-category attribute is represented by l numbers where one of the l numbers is one and the others are zero. To give an example, the marital status attributes with possible values in $\{\text{married}, \text{single}, \text{divorced}, \text{widowed}\}$ are encoded as $(0, 0, 0, 1)$, $(0, 0, 1, 0)$, $(0, 1, 0, 0)$ and $(1, 0, 0, 0)$. Experience indicates that if the number of values in an attribute is not too large, this coding might be more stable than using a single number [14].

Scaling. The main advantage of scaling is to avoid that attributes in greater numeric ranges dominate those in smaller numeric ranges [14]. To avoid these problems it is necessary to scale the feature columns independently. Two ways of scaling used in machine learning are rescaling and standardization [28]. Linearly rescaling to $[0, 1]$ or $[-1, 1]$ is a simple scaling technique where the feature column is scaled to the chosen range. When scaled to $[0, 1]$ we keep zero valued entries which can be beneficial to some algorithms as zero values can allow for formulas with less computational overhead or earlier branch cut offs and result in a faster running time [23]. An example for such an algorithm is the LibSVM [6] implementation of the SVM formula. The other way of scaling is to do standardization of the feature column by subtracting the mean and then dividing by the standard deviation, s.t. the feature columns have zero-mean and unit-variance. We found that standardization worked better with our framework, as the linear scaling approach did not result in reduced computational effort even on data sets with lots of categorical attributes and therefore lots of zero valued features.

4.3 Coarsening

In the coarsening phase we use the training set, which is just a set of vectors in the feature space, and build a hierarchy of problems. In Section 2.4 we explained that the goal is to reduce the size of the problem in each level while still keeping the overall structure

of the original problem. We achieve this by recursively merging nearby vectors which represent very similar information about the problem. The following procedure is done independently once for the $\mathbf{C}^+$ and once for the $\mathbf{C}^-$ class.

By applying a k-nearest neighbor algorithm on the classes data we obtain a graph with information about the proximity of every vertex. The k-nearest neighbor graph is the graph $G = (V, E, c, w)$ calculated by the k-nearest neighbor algorithm, where V is the classes data set and every vertex is adjacent to its k nearest neighbors. The weight of an edge (v, w) is the euclidean distance between the two vertices. Previous experiments with an exact nearest neighbor algorithm showed that an approximate k-nearest neighbor (AkNN) algorithm did not suffer any loss in quality while improving the running time [23]. In our framework we use the AkNN library FLANN [19]. By transforming the abstract data set into a graph we can now use graph theory algorithms to solve the coarsening.

Coarsening a graph $G = (V, E)$ means finding a graph $G' = (V', E')$ that is related to G where $|V'| < |V|$. We first run the label propagation algorithm [21] to find clusters of G in near-linear time. We then contract the clusters, i.e. replace each cluster with a single node. We update node and edge weights as described in Section 3.2. This procedure is repeated recursively until a stop condition is met. Our stop condition is a simple fixed size rule. When the graph order is less than ω the recursive process is stopped and the coarsening is done. The value of ω is an tuning parameter of our framework.

4.4 Initial Training

After creating the problem hierarchies in the coarsening phase, we train a SVM on the coarsest problem. In our implementation we use the highly optimized LibSVM [6] library as the underlying SVM solver. We train the SVM model that we described in Section 2.2 which LibSVM calls C-SVM and use the Gaussian kernel to allow for nonlinear classification. This means we have two SVM parameters to adjust: C and γ. The problem of choosing a good parameter setting for a better generalization performance in a learning task is called *model selection*.

Grid Search. A common way to find the right parameter in SVM learning is to use grid search [20] over the parameter space. In grid search a regular grid is put on the parameter space and the parameter combinations corresponding to the grid vertices are evaluated. Since the parameter range of C and γ is unbounded the grid only covers a plausible space in which the optimum parameters are presumed.

A further optimization is to do a two sweep grid search. We begin searching with a coarse grid over the plausible parameter space and follow with a second sweep using a finer grid in the local area of the best parameter set of the first sweep. This improves the speed of the models selection by only searching coarse in the complete parameter space but still fine in the region where the best parameter set is assumed [20].

Uniform Design. While grid search is a robust way of finding good parameters it is still quite work intensive because many parameter combinations have to be tested. In our framework we use the technique of Uniform Design (UD) introduced to SVM training in [15] which was also used by mlsvm-AMG. UD also does a search for parameters in a specified space but other then grid search less parameter combinations are evaluated.

While grid search evaluates every grid point in the space of the grid, UD has a pre-computed set of point that have maximal distance to each other. This way the parameter space is covered well without checking every point on the grid. The two sweep technique used for grid search is also used in UD. Huang et al. applied this for large data sets [15]. The first coarser sweep consists of 9 points and the second finer sweep of 5 points in the region of the previously best parameter combination. It is also taken care of that no parameter combination is checked twice. Since UD evaluates less points than grid search the model selection is sped up but since the parameter space is still covered well the prediction quality does not suffer a loss.

Validation Set. When we train a SVM with a specific set of parameters we need to evaluate the performance of the prediction for the SVM model trained with these parameters. We use data points with known labels and compare the known correct label to the label which our trained SVM predicts for the data point. The data points we use for this task are called the validation set. The question that arises is which data point should go into the validation set. We can not use the test data for that because it is important not to use the test data in the training process, so that the evaluation on the test data after the finished multilevel training is uncorrupted and meaningful. Otherwise our final evaluation is distorted because our algorithm learned with the test data and the test data is not comparable to never seen new data.

Sadrfaridpour et al. discussed several different approaches for creating the validation set [23]. An investigated attempt was to use a subset of the current levels problem for the validation. This attempt did not show good results possibly because the SVM models were only validated on an already generalization problem and could not effectively learn the underlying structure of the original problem. Rather than the data of the same level we use the original problem, the training set, as validation set in our framework. Since we aim at classifying large problems the training set is huge and we can not validate every parameter combination on the whole training set, which is why we use a subset of the training set for validation. The size of the subset is an other optimization parameter and we found a random 10% subset of the original problem to work best. This technique is also used in the current mlsvm-AMG after it was found superior to other approaches [23].

Training. Let $\mathbf{C}_c^+$ and $\mathbf{C}_c^-$ be the coarsest problems, the vertex set of the coarsest graphs with associated labels and VD the validation set. In Algorithm 2 we present a pseudo code overview of the initial training phase. The first UD sweep is done in Line 1-6 and the

second in Line 7-12. Since mlsvm-AMG also uses the two sweep UD model selection this pseudo code also roughly represents their initial training phase.

Algorithm 2: Initial training.

Input: $\mathbf{C}_c^+$, $\mathbf{C}_c^-$, VD
Output: $\text{model}_{best}, \text{SV}_{best}, C_{best}, \gamma_{best}$

```
1  evalList := list of evaluated SVM models and parameters
2  params_1 ← UD sweep around initial position
3  foreach (C, γ) ∈ params_1 do
4      (model, SV) ← train SVM on C_c^+ ∪ C_c^- using C, γ
5      res ← evaluate model on VD
6      evalList.add((res,model,SV,C,γ))
7  (C_good, γ_good) ← evalList.getEntryWithBestResult()
8  params_2 ← UD sweep around C_good and γ_good
9  foreach (C, γ) ∈ params_2 do
10     (model, SV) ← train SVM on C_c^+ ∪ C_c^- using C, γ
11     res ← evaluate model on VD
12     evalList.add((res,model,SV,C,γ))
13 (model_best, SV_best, C_best, γ_best) ← evalList.getEntryWithBestResult()
```

4.5 Refinement

After the initial training is completed the lowest level of the hierarchy is finished and the multilevel algorithm recursively progresses to the next level by uncontracting the hierarchy, or rather the hierarchies for the majority and minority class at the same time. We have a hierarchy of problems and information about the projection of the problems, i.e. which points of the finer level, the current level, contributed to which of the coarser level, the previous level. In the refinement phase we use the result of the previous level combined with our knowledge about the projection to the current level and train a SVM on the data of the current level.

We can not use every data point of the current levels problem in the training step of this level, otherwise a multilevel algorithm would just increase the work load without benefits. We only use data points which are able to further improve the prediction quality when compared to the trained SVM of previous level.

In Section 2.2 we already saw that the support vectors are the only important data points for the construction of the maximum separating hyperplane. In the refinement phase we use this property of SVM to decrease the size of the problem when going up a level. Consider a trained SVM model on the previous level. The support vectors of the model are data points

of the previous levels problem. For the current level we uncontract those support vectors and use the resulting data points as input for the SVM training in the refinement. In the case that there are more hierarchy steps for the majority class than for the minority class, which is often the case for imbalanced data, we only uncontract the majority hierarchy until both hierarchies have equal size.

Another result we adopt from the previous level is the model selection. In the inital training we found the best parameter setting through a two sweep UD search as explained in Section 4.4. We can assume that the best parameters for the levels above the coarsest level, on which the initial training is conducted, are in the area of the best parameter setting for the coarsest level. In the refinement model selection we skip the first coarser UD sweep we did the initial training and only do the second finer sweep around the best found parameters for the previous level. When we only uncontract one hierarchy due to unbalanced hierarchies levels, we do the complete two sweep. When the problem size exceeds a certain point we only train one SVM model with the best parameters of the previous level and skip the model selection. Our experiments indicate 10.000 to be a good threshold.

We then train a SVM on the adjusted problem of the current steps and modify the model selection as explained. We do so recursively until the complete hierarchy is processed. As the last step we return the best model of the complete multilevel training, which must not be the SVM model of the last level in fact it often is not the model of the finest level because this model is prone to over-fitting. Our refinement phase is based on the previous multilevel SVMs but since our coarsening is small adjustment were made. The mlsvm-AMG authors implemented an option to not only add the uncontracted support vector but also their n-distant neighbors. They explained that on certain data stets this improves the prediction quality of the framework [23]. Such an option is not necessary for our framework because we already perform well on those data sets and the increased running time is not expected to yield any improvements.

4.6 Evaluation

In machine learning is common to split the available data into two sets the training set and the test set [2]. The learning is conducted on the training set and when the learning is finished the resulting (multilevel) model is evaluated on the test set. The test data acts as newly incoming data for which in a real world application we do not have labels and want to use the trained classifier to obtain said labels. Because we have labels for the test data, we can measure the prediction quality of our framework on completely new data point.

In our framework we use k-fold cross-validation to get a more accurate estimate of the prediction performance. By using k-fold we can simulate real world application of our algorithm because our test results are not influenced by a particular good or bad training data to test data relation. In our k-fold implementation we first shuffle the data and split it into k parts of equal size. We then perform k complete runs of our multilevel algorithm.

In every run one of the parts is used as the *testing set* while the other $k - 1$ parts make the *training set*. The algorithm is trained using only the training set. The test set is never used to train or validate any of the intermediate results of the hierarchy; only the final result of a run will be evaluated with the test set. Once all k runs have finished we average the results of all runs and present that as the overall quality of our training process.

5 Experimental Evaluation

In this chapter we show the results of our benchmarks on a number of publicly available data sets. We compare our framework to mlsvm-IIS [22] and mlsvm-AMG [24] on the same instances and use the results that are published in the respective papers. A comparison of the running times given in the papers is difficult because we used a different machine for our algorithm. To compare our framework speed wise to the mlsvm-AMG we used the code of mlsvm-AMG that is available on github[1]. It runs on the same machine as our framework and the quality and running time of the algorithms can be compared.

5.1 Experimental Setup

Environment. We name our framework KaMLSVM (Karlsruhe Multilevel Support Vector Machine). It is build on top of the KaHIP (Karlsruhe High Quality Partitioning) framework [25]. Additionally we use the latest versions of the FLANN [19] and LibSVM [6] library, which at the time of writing are FLANN 1.8.4 and LibSVM 3.22. We compile our code with the latest GNU C++ compiler 8.1.1. All of our experiment were executed on a machine with an AMD Opteron 6168 1,9GHz and 256GB RAM.

Instances. We mostly use large problem instances (data sets with size >20.000) with a high imbalance factor (quotient of the majority class size and the total size) because making training feasible on those problems and improve on previous attempts is our main goal. Most of our problem instances are available on the UC Irvine Machine Learning Repository [9] and a few were given to us by Ehsan Sadrfaridpour, the author of the mlsvm-AMG framework. The details about all the benchmark data sets we used are presented in Table 5.1

Tuning Parameters. The tuning parameter of the SVM C and the kernel parameter γ are determined during SVM training in the model selection as depicted in Section 4.4 and therefore are not tuning parameter of our framework.

A tuning parameter of our framework is the size constrain of the label propagation algorithm. It determines the maximum size of a cluster and a smaller size containing results in smaller and more clusters which leads to more levels in the hierarchy. We experimented

[1] `https://github.com/esadr/mlsvm`

Name	Size	Features	C^+	C^-	imbalance
Advertisement	3.279	1.558	459	2.820	0,86
APS failure	76.000	170	1.375	74.625	0,98
Buzz	140.707	77	27.775	112.932	0,80
Census	299.285	41	18.568	280.717	0,94
Cod-rna	59.535	8	19.845	39.690	0,67
EEG Eye State	14.980	14	6.723	8.257	0,55
Forest (Class 1)	581.012	54	221.840	369.172	0,64
Forest (Class 2)	581.012	54	283.301	297.711	0,51
Forest (Class 3)	581.012	54	35.754	369.172	0,94
Forest (Class 4)	581.012	54	2.747	578.265	1,00
Forest (Class 5)	581.012	54	9.493	571.519	0,98
Forest (Class 6)	581.012	54	17.367	563.645	0,97
Forest (Class 7)	581.012	54	20.510	560.502	0,96
Hypothyroid	3.919	21	240	3.679	0,94
Isolet (Class A)	6.919	617	240	5.998	0,96
Letter (Class A)	20.000	16	786	19.266	0,96
Letter (Class B)	20.000	16	766	19.266	0,96
Letter (Class H)	20.000	16	734	19.266	0,96
Letter (Class Z)	20.000	16	734	19.266	0,96
Musk (Clean)	6.598	166	1.017	5.581	0,85
Nursery	12.960	8	4.320	8.640	0,67
Protein	145.751	74	1.296	144.455	0,99
Ringnorm	7.400	20	3.664	3.736	0,50
Skin	245.057	3	50.859	194.198	0,79
Sleep (Class 1)	105.908	13	9.052	96.856	0,91
Twonorm	7.400	20	3.703	3.697	0,50

Table 5.1: Benchmark data sets.

with a size constrain as small as 10 or 20 and tried to overcome the over-fitting of the SVM models in the later levels but experiment showed that that was not the case. We found that not bounding the cluster size worked best.

As depicted in Section 3.2 the maximum number of rounds in the label propagation algorithm is subject to tuning. Previous studies found that the value of 10 is enough to get good clusterings and that increasing the rounds is not necessary [27, 25].

Another tuning parameter that influences the coarsening is the stopping rule of the coarsening. We decided to stick to the rule used in the previous mlsvm experiments which is a hard cap once the graph size is smaller than ω. For $omega$ we used 500 because it is used by mlsvm-AMG. In the future more sophisticated rules are to be explored.

For the k-fold validation 5 and 10 were considered as good values and the authors of the mlsvm-AMG choose to use 5 as a default parameter [23]. We also experimented with this value and stuck with 5. This decision is also backed up by a general machine learning rule of thumb that suggests to use 20% of the available data as test data and the size of the test data is exactly 20% for every run of the 5-fold [10]. For our benchmark results for every data set we ran our framework with a full 5-fold run 5 times and averaged the results that makes 25 individual multilevel run. For the mlsvm-AMG we also ran 5 full experiments per data set with a 5-fold and averaged the results.

5.2 Experiments

5.2.1 Performance Measures

We use the performance measures *sensitivity* (SN), *specificity* (SP), *G-mean*, and *accuracy* (ACC) to evaluate our prediction results. The definition for those are

$$\begin{aligned} ACC &= \frac{TP + TN}{FP + TN + TP + FN} \\ SN &= \frac{TP}{TP + FN} \\ SP &= \frac{TN}{TN + FP} \\ G\text{-}mean &= \sqrt{SP * SN} \end{aligned}$$

where TP are to true positives, the correctly classified points of C^+, FN the false negatives, wrongly classified points of the minority class C^+, TN the true negatives, correctly classified point of C^- and FP the false positives, points of C^- wrongly classified as points of the minority class. These metrics are common in statistical analysis with accuracy being the most used metric in machine learning [2].

We can not use the accuracy as a primary measure for the prediction quality because we work with highly imbalance data and especially when working with medical data sets type

II errors (false negatives) are of eminent importance to us. When using the accuracy on large data sets that are imbalanced getting high accuracy is trivial by only predicting the larger class. This leads to a zero percent sensitivity but since the number of true negatives dominates the other values the accuracy is still high. Instead of using the accuracy which is flawed on large imbalanced data sets, we use the G-mean, the geometric mean of the sensitivity and the specificity, because it is more sensitive to wrong classification in general and yields more informative results on imbalanced data sets.

When we compare trained SVM models, e.g. in the model selection step of the training or later when searching for the best overall SVM model, we use the G-mean as primary decision criterion and for models with roughly the same G-mean we choose the model with less support vectors. From this we get a good compromise between the quality of the prediction on the validation set and the generalization that is needed to allow for the best performance on the test data later.

5.2.2 Comparison with mlsvm-IIS and mlvsm-AMG

To put our frameworks prediction quality into context we compare our results to the previous multilevel SVM framework. In the respective papers results of the mlsvm-IIS and the mlsvm-AMG on the same benchmark data sets where presented [23]. We run our framework on the exact same instances that were reported in the papers. Because we used a different machine with a worse single-core performance the results are only comparable prediction quality wise. In the upper part of Table 5.2 we compare the mlsvm-IIS and mlsvm-AMG results to the benchmark of our own framework. The results of the mlsvm-IIS can be found in Table 3 and 4 of the mlsvm-AMG paper and the results for the mlsvm-AMG itself on the same data sets can be found in Table 9 and 10 of the paper [23].

It is important to note that we initially were unable to reproduce the results of mlsvm-AMG with the code published on github. In an email conversation the author we found out that the configuration for each data set is individually tuned and that the default parameters were only a starting point for the per instance optimization. We did not adjust our parameter for every data set because we think a general framework as ours should report the results unaltered and use the default parameters on in the experiments. Therefore our results are a few percent worse on some data sets. We believe that with the same amount of individual parameter tuning this gap could be closed but leave the individual tuning work to researchers working on a specific data set.

To present a fair comparison between the open source mlsvm-AMG framework and our framework we compiled the code of the mlsvm-AMG which can be found on github at `https://github.com/esadr/mlsvm`. We run the mlsvm-AMG on the same machine as our framework, the one described in Section 5.1, to provide true comparability. We also payed attention to use the frameworks under the same conditions, e.g. with the same optimization compile flags and comparable release configurations for both code bases. When we ran the mlsvm-AMG framework we used the default parameters that are

Dataset	mlsvm-IIS		mlsvm-AMG		KaMLSVM	
	ACC	G-mean	ACC	G-mean	ACC	G-mean
Advertisement	0,94	**0,87**	0,90	0,86	0,94	0,80
Buzz	0,94	0,90	0,94	**0,95**	0,94	0,94
Cod-rna	0,95	**0,95**	0,93	0,94	0,94	0,94
Forest (Class 1)			0,73	0,74	0,80	**0,80**
Forest (Class 2)			0,70	0,70	0,80	**0,80**
Forest (Class 3)			0,90	0,94	0,93	**0,95**
Forest (Class 4)			0,92	0,96	0,97	**0,98**
Forest (Class 5)	0,93	**0,91**	0,78	0,85	0,90	0,89
Forest (Class 6)			0,86	0,90	0,90	**0,94**
Forest (Class 7)			0,91	0,89	0,89	**0,93**
Letter (Class Z)	0,98	0,97	0,98	**0,99**	0,96	0,96
Musk (Clean)	1,00	**0,99**	0,99	**0,99**	0,97	0,97
Nursery	1,00	0,99	1,00	**1,00**	1,00	**1,00**
Ringnorm	0,98	**0,98**	0,98	0,98	0,97	0,97
Twonorm	0,97	0,97	0,98	**0,98**	0,97	0,97
Advertisement			0,91	0,78	0,94	**0,80**
APS failure			0,94	**0,94**	0,94	0,93
Buzz			0,94	**0,95**	0,94	0,94
Census			0,74	0,80	0,84	**0,83**
Cod-rna			0,94	**0,95**	0,94	0,94
EEG Eye State			0,78	**0,78**	0,78	0,77
Forest (Class 1)			0,73	0,75	0,80	**0,80**
Forest (Class 2)			0,73	0,73	0,80	**0,80**
Forest (Class 3)			0,90	0,94	0,93	**0,95**
Forest (Class 4)			0,94	0,97	0,97	**0,98**
Forest (Class 5)			0,73	0,79	0,90	**0,89**
Forest (Class 6)			0,82	0,89	0,90	**0,94**
Forest (Class 7)			0,83	0,87	0,89	**0,93**
Hypothyroid			0,93	**0,94**	0,97	0,85
Isolet (Class A)			0,97	**0,99**	0,99	**0,99**
Letter (Class A)			0,99	**0,98**	0,97	0,96
Letter (Class B)			0,96	**0,95**	0,94	0,94
Letter (Class H)			0,96	0,88	0,90	**0,91**
Letter (Class Z)			0,96	**0,97**	0,96	0,96
Musk (Clean)			0,92	0,92	0,97	**0,96**
Nursery			1,00	**1,00**	1,00	**1,00**
Protein			0,93	0,92	0,96	**0,93**
Ringnorm			0,98	**0,98**	0,97	0,97
Skin			0,99	0,99	1,00	**1,00**
Sleep (Class 1)			0,65	0,64	0,71	**0,66**
Twonorm			0,97	**0,97**	0,97	**0,97**

Table 5.2: Comparison of prediction quality to mlsvm-IIS and mlsvm-AMG. In the upper table are the results of the mlsvm papers with individually tuned parameters and in the lower table the results of the mlsvm-AMG with the default configuration.

published in the github repository and used the default parameters for our framework too. The second part of Table 5.2 compares the results of the complied mlsvm-AMG to our framework. We highlighted the better G-mean for each instance. On almost all instances our framework provides prediction results that are comparable to the results of the mlsvm-AMG. On about half the benchmark data sets we are one to two percent better and on the other one to two percent worse. Only on the `Hypothyroid` data set the G-mean of our framework is considerable worse than the G-mean of the mlsvm-AMG framework. `Hypothyroid` is a very small data set in fact with a size of less than 5.000 a standard, no-mulitilevel, SVM such as LibSVM is perfectly capable of training on the data set directly and the generalization that happens in our framework is unnecessary and leads to a poor prediction quality. On the `Advertisement` data set the discrepancy between our results of our compiled mlsvm-AMG and the results of the paper is quite large. With the default parameters the mlsvm-AMG fails to perform good on some of the experiment runs so that the average classification results are worse.

It is also interesting that we got slightly better results for the mlsvm-AMG on some of the `Forest` classes and the `Cod-rn` data set when compared to the reported results of mlsvm-AMG in [23]. This could be due to the fact that we used the latest github version of the framework which is more consolidated than the version which was used for the paper and therefore results in slightly better prediction quality.

5.2.3 Running Time Comparison

With compiling the mlsvm-AMG code we can now compare the running times of the two frameworks. In Table 5.3 we compare the running time of the mlsvm-AMG to the running time of our algorithm. The running time for each data set is given in seconds and includes a single multilevel run. We can see that our algorithm is faster than the mlsvm-AMG approach in almost all benchmarks and that on data sets with a large amount of features the difference in computational time is even bigger.

The exceptions are the `EEG Eye State` and the `Hypothyroid` data sets on which our algorithm needs more time than the mlsvm-AMG. The most time in our framework is spend in the refinement phase and when the coarsening is very unfavorable such that the trained SVM models have a lot of support vectors the refinement can take a long time since the problem gets almost to the original size. This problem could probably be solved by developing a better coarsening stop criterion. For all other data sets we see a considerable speed up in our framework when we compare it to the mlsvm-AMG.

When we consider the speed-up dependent on the number of features, e.g. the data sets `APS failure` and `Protein` which are lager data sets with a large amount of features, we see that our framework has a speed-up of almost 20 on those data sets. We assume that the mlsvm-AMG does not scale as well with increasing feature size because the algebraic multigrid coarsening has to do a lot more work for each feature. The label propagation algorithm on the other hand is independent of the number of features as it only uses the k-

Dataset	size * features	mlsvm-AMG	KaSMLVM	Speed-up
Advertisement	5.108.682	444,60	**262,00**	1,70
APS failure	12.920.000	1.523,92	**78,60**	19,39
Buzz	10.834.439	290,65	**106,00**	2,74
Census	12.270.685	1.551,31	**347,00**	4,47
Cod-rna	476.280	53,60	**15,40**	3,48
EEG Eye State	209.720	**83,87**	471,00	0,18
Forest (Class 1)	31.374.648	12.658,44	**4.510,00**	2,81
Forest (Class 2)	31.374.648	13.546,60	**7.220,00**	1,88
Forest (Class 3)	31.374.648	12.775,10	**373,00**	34,25
Forest (Class 4)	31.374.648	2.358,41	**494,00**	4,77
Forest (Class 5)	31.374.648	6.985,84	**1.420,00**	4,92
Forest (Class 6)	31.374.648	2.327,19	**948,00**	2,45
Forest (Class 7)	31.374.648	3.075,57	**377,00**	8,16
Hypothyroid	82.299	**1,64**	3,23	0,51
Isolet (Class A)	3.848.846	101,53	**22,20**	4,57
Letter (Class A)	320.000	9,50	**3,65**	2,60
Letter (Class B)	320.000	27,32	**5,53**	4,94
Letter (Class H)	320.000	68,25	**10,50**	6,50
Letter (Class Z)	320.000	12,29	**4,54**	2,71
Musk (Clean)	1.095.268	35,09	**13,50**	2,60
Nursery	103.680	8,33	**2,52**	3,31
Protein	10.785.574	1.350,35	**69,60**	19,40
Ringnorm	148.000	**3,77**	4,14	0,91
Skin	735.171	122,61	**20,40**	6,01
Sleep (Class 1)	1.376.804	921,17	**654,00**	1,41
Twonorm	148.000	9,86	**0,71**	13,93

Table 5.3: Computational time of mlsvm-AMG and our framework in seconds.

nearest neighbor graph and data points can not be in multiple clusters. Of course the k-NN graph is harder to compute for large data sets and data sets with a high number of features but this affects both frameworks and the creation of the k-NN graph is not of significant importance to the overall computational time.

We note that the mlsvm-AMG was tested against the fastest standard SVM and outperformed those algorithms running time wise by several orders of magnitude and was comparable in terms of the prediction quality. In conclusion this means that our framework is also on par with the standard SVMs and that the running time difference is even larger in favor for our framework.

5.2.4 Refinement Assessment

In this section we judge the refinement step of multilevel SVM in general and of our framework in particular. To get an idea of what the refinement step actually achieves we compare the results of our framework after the initial training to the final results of the complete multilevel procedure of our framework. In Table 5.4 we present our classification results after the initial training evaluated on the test data and the final classification results also on the test data. Additionally the number of levels in the hierarchy for each benchmark data set are depicted and compared to the number of levels in the mlsvm-AMG framework.

Dataset	initial		final		KaMLSVM	mlsvm-AMG
	ACC	G-mean	ACC	G-mean	Levels	Levels
Advertisement	0,84	0,87	0,94	0,80	2	3
APS failure	0,96	0,93	0,94	0,93	4	4
Buzz	0,94	0,93	0,94	0,94	4	5
Census	0,76	0,81	0,84	0,83	5	5
Cod-rna	0,93	0,94	0,94	0,94	4	4
EEG Eye State	0,65	0,64	0,78	0,77	3	3
Forest (Class 1)	0,76	0,75	0,80	0,80	5	5
Forest (Class 2)	0,75	0,75	0,80	0,80	5	5
Forest (Class 3)	0,94	0,94	0,93	0,95	5	5
Forest (Class 4)	0,94	0,97	0,97	0,98	5	5
Forest (Class 5)	0,83	0,88	0,90	0,89	5	5
Forest (Class 6)	0,89	0,94	0,90	0,94	5	5
Forest (Class 7)	0,95	0,92	0,90	0,93	5	5
Hypothyroid	0,97	0,87	0,97	0,85	2	3
Isolet (Class A)	0,80	0,89	0,99	0,99	2	3
Letter (Class A)	0,95	0,95	0,97	0,96	3	4
Letter (Class B)	0,93	0,93	0,94	0,94	3	4
Letter (Class H)	0,93	0,89	0,90	0,91	3	4
Letter (Class Z)	0,95	0,95	0,96	0,96	3	4
Musk (Clean)	0,93	0,86	0,97	0,96	2	3
Nursery	1,00	1,00	1,00	1,00	2	3
Protein	0,90	0,92	0,96	0,93	3	5
Ringnorm	0,80	0,80	0,97	0,97	2	3
Skin	1,00	1,00	1,00	1,00	5	5
Sleep (Class 1)	0,89	0,36	0,71	0,66	3	4
Twonorm	0,97	0,97	0,97	0,97	2	3

Table 5.4: Comparison inital of the training accuracy and G-mean to the final prediction results.

It is odd that the initial training results for the `Advertisement` and the `Hypothyroid` data set are better than the final results. This can happen because during refinement the

SVM models are evaluated on the validation set and not the test data. The SVM models get better on the validation set but in a few cases worse on the train data. This is a sign of over-fitting because the SVM models gets better on the problem data, in this case the validation set, but the improvements of the refinement show an opposing effect on the test data. In the multilevel process we do not have results of the initial training on the test data, we only have the results on the validation set, which is why it is not possible for the framework to know that the initial model performs better on the test data than the actual chosen final model.

When we compare the hierarchy level our framework builds to the hierarchy levels of the mlsvm-AMG [23] we see that the label propagation coarsening build less levels than the algebraic multigrid approach. Building fewer levels is of advantage because the coarsening time goes down and since in the refinement phase one SVM training is done for each level the refinement time also goes down. But few hierarchy levels have the disadvantage that even a few support vectors on the coarse levels can uncoarse to a large number of data points in the next upper level. This make SVM training more time consuming as the data set to train on gets large quite fast.

In Figure 5.1 we present a stacked bar plot with the running time percentages of the different phases of our framework namely the coarsening, initial training and refinement. We see that on the `Hypothyroid` data set the refinement take almost all of the time which is an indication that our framework has a problem on this data set. Since the refinement step is the most time consuming we need to make sure that it is of benefit for all data sets. In the refinement phase training a SVM on larger as necessary problems increases the running time without benefits. Improving the refinement phase further can be an interesting task as it can lead to better running time and improved classification quality.

For other data sets like `EEG Eye State`, `Forest Class 1 & 2`, `Musk`, `Ringnorm` and `Sleep` the refinement is very important and the improvement is between 5 and 30 percent.

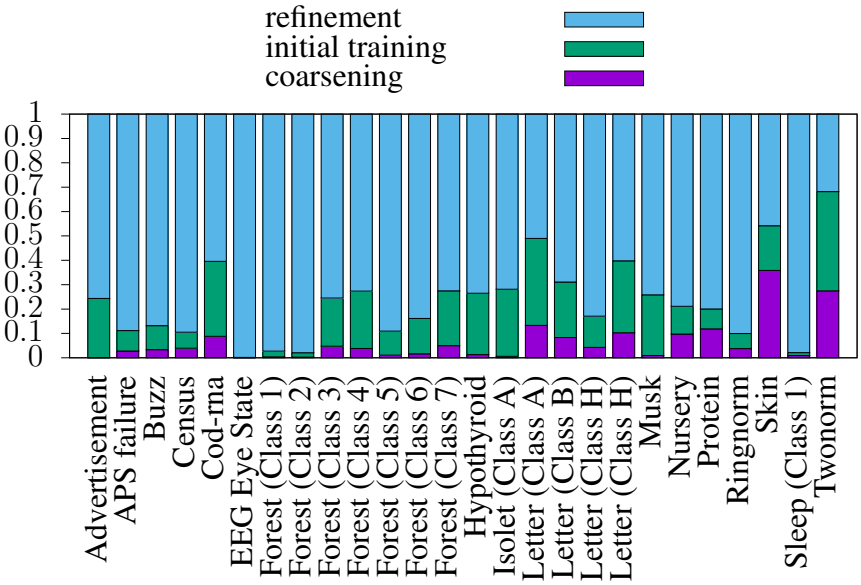

Figure 5.1: Plot showing the running time percentage for the different multilevel phases of our framework.

6 Discussion

6.1 Conclusion

In this theses we developed a novel multilevel framework (KaMLSVM) for nonlinear support vector machines. Our framework utilizes the label propagation algorithm in the coarsening step and is build on top of the KaHIP framework [25]. We ran a variety of experiments to compare the state-of-the-art multilevel SVM frameworks and our framework on classification quality and computation performance. Our experiments show that our framework has a speed-up of at least two, showing significant improved in the running time. The classification quality is comparable or improved when compared to the previous multilevel SVM frameworks. For large data sets with a lot of features our framework achieves a speed-up of up to 20 presumably due to the fact that the AMG-framework does not scale well with the number of features. With improving the running time of multilevel SVM we can now train SVM even more efficiently on large data sets than the previous multilevel SVM allowed.

6.2 Future Work

There is still a lot of research potential on the topic of multilevel SVMs. In our experiments we saw that in exceptional cases the fixed stopping rule of our coarsening step can lead to a significant running time increase on some data sets and further research to optimize the stopping criterion is possible.

Another interesting task is the parallelization of the framework which was out of the scope of this thesis but should be straightforward as many steps of the framework are parallelizable.

We explained why we choose not to optimize the tuning parameters for each data set individually but it would be interesting to see what results can be achieved with our framework when we optimize tuning parameters for the specific data sets.

Bibliography

[1] M. A. Aizerman, E. M. Braverman, and L. I. Rozoner. “Theoretical foundations of the potential function method in pattern recognition learning”. In: *Automation and Remote Control* 25 (1964), pp. 821–837.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. 2nd. The MIT Press, 2010. ISBN: 026201243X, 9780262012430.

[3] Robin Andre, Sebastian Schlag, and Christian Schulz. “Memetic Multilevel Hypergraph Partitioning”. In: *CoRR* abs/1710.01968 (2017). arXiv: 1710.01968. URL: http://arxiv.org/abs/1710.01968.

[4] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN: 9780387310732. URL: http://www.worldcat.org/oclc/71008143.

[5] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. “A Training Algorithm for Optimal Margin Classifiers”. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT ’92. Pittsburgh, Pennsylvania, USA: ACM, 1992, pp. 144–152. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130401. URL: http://doi.acm.org/10.1145/130385.130401.

[6] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM: A library for support vector machines”. In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

[7] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd ed. The MIT Press, 2001. ISBN: 0-262-03293-7.

[8] Koby Crammer and Yoram Singer. “On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines”. In: *Journal of Machine Learning Research* 2 (2001), pp. 265–292. URL: http://www.jmlr.org/papers/v2/crammer01a.html.

[9] Dua Dheeru and Efi Karra Taniskidou. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[10] Isabelle Guyon. “A Scaling Law for the Validation-Set Training-Set Size Ratio”. In: 1 (Aug. 1996).

[11] David Money Harris and Sarah Harris. “Introductory digital design & computer architecture curriculum”. In: *2013 IEEE International Conference on Microelectronic Systems Education, MSE 2013, Austin, TX, USA, June 2-3, 2013*. 2013, pp. 14–16. DOI: 10.1109/MSE.2013.6566693. URL: https://doi.org/10.1109/MSE.2013.6566693.

[12] Tobias Heuer, Peter Sanders, and Sebastian Schlag. “Network Flow-Based Refinement for Multilevel Hypergraph Partitioning”. In: *17th International Symposium on Experimental Algorithms (SEA 2018)*. Ed. by Gianlorenzo D’Angelo. Vol. 103. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 1:1–1:19. ISBN: 978-3-95977-070-5. DOI: 10.4230/LIPIcs.SEA.2018.1. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8936.

[13] Cho-Jui Hsieh, Si Si, and Inderjit S. Dhillon. “A Divide-and-Conquer Solver for Kernel Support Vector Machines”. In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. 2014, pp. 566–574. URL: http://jmlr.org/proceedings/papers/v32/hsieha14.html.

[14] Chih-wei Hsu, Chih-chung Chang, and Chih-jen Lin. *A practical guide to support vector classification*. 2010.

[15] Chien-Ming Huang et al. “Model selection for support vector machine via uniform design”. In: 52 (Sept. 2007), pp. 335–346.

[16] Ron Kohavi and Foster Provost. “Glossary of Terms”. In: *Machine Learning* 30.2 (Feb. 1998), pp. 271–274. ISSN: 1573-0565. DOI: 10.1023/A:1017181826899. URL: https://doi.org/10.1023/A:1017181826899.

[17] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008. ISBN: 978-0-521-86571-5.

[18] Henning Meyerhenke, Peter Sanders, and Christian Schulz. “Partitioning (hierarchically clustered) complex networks via size-constrained graph clustering”. In: *J. Heuristics* 22.5 (2016), pp. 759–782. DOI: 10.1007/s10732-016-9315-8. URL: https://doi.org/10.1007/s10732-016-9315-8.

[19] Marius Muja and David G. Lowe. “Scalable Nearest Neighbor Algorithms for High Dimensional Data”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 36.11 (2014), pp. 2227–2240. DOI: 10.1109/TPAMI.2014.2321376. URL: https://doi.org/10.1109/TPAMI.2014.2321376.

[20] Alexander Popov and Alexander Sautin. “Selection of support vector machines parameters for regression using nested grids”. In: *2008 Third International Forum on Strategic Technologies*. June 2008, pp. 329–331. ISBN: 978-1-4244-2319-4. DOI: 10.1109/IFOST.2008.4602974.

[21] Nandini Raghavan, Réka Albert, and Soundar Kumara. “Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks”. In: 76 (Oct. 2007), p. 036106.

[22] Talayeh Razzaghi and Ilya Safro. “Scalable Multilevel Support Vector Machines”. In: *Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavı k, Iceland, 1-3 June, 2015, 2014*. 2015, pp. 2683–2687. DOI: 10.1016/j.procs.2015.05.381. URL: https://doi.org/10.1016/j.procs.2015.05.381.

[23] Ehsan Sadrfaridpour, Talayeh Razzaghi, and Ilya Safro. “Engineering multilevel support vector machines”. In: *CoRR* abs/1707.07657 (2017). arXiv: 1707.07657. URL: http://arxiv.org/abs/1707.07657.

[24] Ehsan Sadrfaridpour et al. “Algebraic multigrid support vector machines”. In: *CoRR* abs/1611.05487 (2016). arXiv: 1611.05487. URL: http://arxiv.org/abs/1611.05487.

[25] Peter Sanders and Christian Schulz. “Think Locally, Act Globally: Highly Balanced Graph Partitioning”. In: *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*. 2013, pp. 164–175. DOI: 10.1007/978-3-642-38527-8_16. URL: https://doi.org/10.1007/978-3-642-38527-8_16.

[26] Bernhard Schölkopf and Alexander Johannes Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning series. MIT Press, 2002. ISBN: 9780262194754. URL: http://www.worldcat.org/oclc/48970254.

[27] Christian Schulz. “High Quality Graph Partitioning”. PhD dissertation. Karlsruhe Institute of Technology, 2013. URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000035713.

[28] Andreas Stolcke, Sachin S. Kajarekar, and Luciana Ferrer. “Nonparametric feature normalization for SVM-based speaker verification”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2008, March 30 - April 4, 2008, Caesars Palace, Las Vegas, Nevada, USA*. 2008, pp. 1577–1580. DOI: 10.1109/ICASSP.2008.4517925. URL: https://doi.org/10.1109/ICASSP.2008.4517925.

[29] Hwanjo Yu, Jiong Yang, and Jiawei Han. “Classifying large data sets using SVMs with hierarchical clusters”. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*. 2003, pp. 306–315. DOI: 10.1145/956750.956786. URL: http://doi.acm.org/10.1145/956750.956786.