

Fachhochschule Köln
Cologne University of Applied Sciences

Case Study Report:
Building and analyzing SVM ensembles with Bagging and AdaBoost on big data sets

Ricardo Ramos Guerra · **Jörg Stork**

Master in Automation and IT
Faculty of Computer Science and Engineering Sciences,
Cologne University of Applied Sciences,
Steinmüllerallee 1, 51643 Gummersbach, Germany

Submission date: 23th of April, 2013

Ricardo Ramos Guerra
E-mail: ricardo.ramos@smail.fh-koeln.de

Jörg Stork
E-mail: joergstork85@googlemail.com

Abstract This report covers an estimation of the quality of classification ensembles for large data tasks based upon Support Vector Machines (SVMs)[4]. SVMs have an cubic scaling for most kernels with the amount of training data[23]. This generates an enormous computational effort if it comes to large data sets with more than 100.000 records. It will be shown that bagging[1] and AdaBoost are suitable ensembles methods to reduce this computational effort. These methods make it possible to create one strong classifiers consisting of an ensemble of SVMs where each SVM was trained with only a fraction of the complete training data. Also ensembles using different kernels(radial, polynomial, linear), which are capable to deliver results superior to an single SVM, will be introduced.

Keywords Support Vector Machines (SVM) · SVM Ensembles · Ensemble Constructing Methods · AdaBoost · Bagging · Big Data

Contents

1 Introduction . . . 7
2 Motivation, Goals and Current Research . . . 8
3 Basic Methods . . . 10
3.1 Support Vector Machines . . . 10
3.1.1 Separable case . . . 10
3.1.2 Non separable case . . . 11
3.1.3 Kernels and Support Vector Machines . . . 12
3.2 Ensemble Methods . . . 13
3.2.1 SVM Bagging . . . 13
3.2.2 Boosting . . . 14
4 Implementation . . . 15
4.1 SVM AdaBoost . . . 15
4.1.1 Gamma (γ) Estimation . . . 16
4.2 SVM Bagging . . . 17
5 Experiments . . . 19
5.1 Data Sets . . . 19
5.1.1 SPAM . . . 20
5.1.2 Adult . . . 20
5.1.3 Satellite . . . 20
5.1.4 Optical Recognition of Handwritten Digits . . . 20
5.1.5 Acoustic . . . 20
5.2 Experimental Setup . . . 20
5.2.1 Results for Bagging . . . 20
5.2.2 AdaBoost . . . 22
6 Results . . . 24
6.1 Bagging . . . 24
6.1.1 Spam . . . 24
6.1.2 Satlog . . . 25
6.1.3 Optdig . . . 27
6.1.4 Adult . . . 28
6.1.5 Acoustic . . . 28
6.1.6 Acoustic Binary . . . 29
6.1.7 Connect4 . . . 30
6.1.8 Majority vs Probability Voting . . . 31
6.2 Results for AdaBoost . . . 32
6.2.1 Results using full train size . . . 33
6.2.2 Results using factor $bo.size$ . . . 35
6.2.3 General comparison between Full Train against $bo.size$ experiments inside SVM-AdaBoost . . . 41
7 Discussion . . . 44
7.1 SVM Bagging . . . 44
7.1.1 Early Investigations . . . 44
7.1.2 Result Summary . . . 44
7.1.3 Influence of the Sample Size . . . 44
7.1.4 Influence of Different Kernels . . . 45
7.1.5 Influence of the Ensemble Size . . . 45
7.1.6 Majority vs Probability Voting . . . 45
7.1.7 Optimization and Tuning . . . 45
7.2 AdaBoost . . . 46
7.2.1 AdaBoost Result Summary . . . 46
7.2.2 Conclusions AdaBoost . . . 47
8 Conclusion . . . 47
9 Future Work . . . 48

A AdaBoost Important Files 51
B SVM Bagging Important Files 51

List of Figures

2.1 Training times of single SVMs with the different kernels(radial, linear, 3rd degree polynomial) vs sampling size on the Adult data set wit a step size of 500 . . . 8
3.1 Example Support Vector Machines. . . . 12
3.2 Schematic showing the SVM bagging method . . . 14
4.1 Example estimated γ. . . . 16
6.1 Spam data set, boxplot with different kernels and their combinations, gain vs sample size . . . 24
6.2 Acoustic Binary data set boxplot result plot of the sample size test, sample size vs gain . . . 30
6.3 Connect4 Result Boxplot . . . 31
6.4 Accuracy on task Optical Digit Recognition 100% Train. . . . 33
6.5 Accuracy on task Spam 100% Train. . . . 34
6.6 Performance degradation on tasks Spam and Satellite against $bo.size$. . . . 35
6.7 Accuracy on task Optical Digit Recognition, $bo.size = 0.078$ . . . 36
6.8 Accuracy on task Satellite, $bo.size = 0.067$ . . . 37
6.9 Accuracy on task Spam, $bo.size = 0.1$ . . . 38
6.10 Accuracy on task Adult, $bo.size = 0.01$ . . . 39
6.11 Accuracy on task Acoustic, $bo.size = 0.003806$ . . . 40
6.12 Support Vectors per weak classifier in SVM-AdaBoost against $bo.size$. . . . 41
6.13 Selection Frequency of train elements inside SVM-AdaBoost. . . . 42
6.14 Selection frequency of train elements in SVM-AdaBoost, pt.2. . . . 43

List of Tables

3.1 Aggregation Types . . . 14
4.1 Random vs Stratified Sampling . . . 18
5.1 Data sets for this case study. . . . 19
6.1 Spam Single SVM . . . 24
6.2 Spam SST Results . . . 25
6.3 Spam EST Results . . . 25
6.4 Satlog Single SVM . . . 25
6.5 Satlog SST Results . . . 26
6.6 Satlog EST Results . . . 26
6.7 Optdig Single SVM . . . 27
6.8 Optdig SST Results . . . 27
6.9 Optdig EST Results . . . 27
6.10 Adult Single SVM . . . 28
6.11 Adult SST Results . . . 28
6.12 Adult EST Results . . . 28
6.13 Acoustic Single SVM . . . 28
6.14 Acoustic Data Set SST Results . . . 29
6.15 Acoustic Data Set EST Results . . . 29
6.16 Acoustic Binary Single SVM Results . . . 29
6.17 Acoustic Binary Set SST Results . . . 29
6.18 Acoustic Binary EST Results . . . 30
6.19 Majority vs Probability Voting . . . 31
6.20 Parameters used in AdaBoost for each task. . . . 32

6.21 Train times on task **Optical Digit Recognition** 100% Train 33
6.22 Train times on task **Spam** 100% Train 34
6.23 *bo.size* parameters used for each task. 36
6.24 Train times on task **Optical Digit Recognition** $bo.size = 0.078$ 36
6.25 Train times on task **Satellite** $bo.size = 0.067$ 37
6.26 Train times on task **Spam** $bo.size = 0.1$ 38
6.27 Train times on task **Adult** $bo.size = 0.01$ 39
6.28 Train times on task **Acoustic** $bo.size = 0.003806$ 40
7.1 Bagging Summary Result Table 44
7.2 Prediction accuracies on all tasks. 46
7.3 Training times on all tasks. 46

1 Introduction

Big data describes data sets which are becoming so large and complex that they are difficult to process. Big data introduces a whole range of new challenges, including the capture, transfer, storage, analysis and visualization of these sets. The amount of data grows every year, driven by new sensors, social media sites, digital pictures and videos, cell phones and the increasing number of computer aided processes in industry, finance, and science. The worlds technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s and in 2012 every day 2.5 quintillion (2.5×10^{18}) bytes of data were created [12]. These data sets carry a huge potential to extract different kinds of information for e.g. market research, finance fraud-detection, energy optimization, or medical treatment. But the pure size of them can make them not feasible to process in a reasonable amount of time. Therefore they introduce the need of adapting the current data analysis methods to the new needs of big data applications. The computational cost and memory consumption slip in the focus of the optimization. State-of-the-art methods like Random Forests (RF)[2], Support Vector Machines (SVMs) [4] or Neural Networks [11], which have proven to work well with small data sets, have to be adapted to solve big data problems in decent time. SVMs can be used for different kinds of classification problems and have proven to be strong classifiers which can be tuned to fit to the most different data sets. They are also robust and quite fast for small data sets but internal SVM optimization problem is equivalent to a quadratic program, that optimizes a quadratic cost function subject to linear constraints [16]. The computational and memory cost of SVMs is therefore cubic to the size of the data set [23]. Thus for large data sets the training time and the memory consumption will become an obstacle for the complete classification process. This training of SVMs is difficult to parallelize for a single SVM. Yu et al. [28] present different approaches to overcome the large computational time with methods like cluster-based data selection and parallelization without using ensemble based methods. Wang et al. [25] investigate different ensemble based methods like bagging and boosting [1], but without the focus on the big data task. Meyer et al. [19] uses bagging and cascade ensemble SVMs for large data sets. This report covers bagging and AdaBoost ensemble algorithms, which allow a significant reduction of the sample size per SVM and also an easy parallelization of the training process. This is achieved by using only a fraction of the data per single SVM in the Ensemble and then combining these SVMs to one strong classifier by suitable aggregation methods. Further, the construction of ensembles using different kernel types (linear, polynomial, radial) is investigated. In Section 2, the motivation for this paper and the current state of the research is described. This is done based on a selection of papers discussing big data, bagging, AdaBoost and parallelization of classification algorithms. In Section 3, the basic methods used in this report are further illustrated, namely SVMs, bagging and AdaBoost. In Section 4, the implementation of these methods is discussed. Next, in Section 5, the experimental setup is explained, introducing the data sets, the experimental loops and the parameters chosen for the experiments. Section 6 covers all the results for the different experiments and finally in Section 7 these results are discussed and in Section 8 a conclusion is drawn.

2 Motivation, Goals and Current Research

The motivation for this paper was introduced by the rising interest for big data tasks. Today, lots and lots of data is generated by the most different applications in industry or everyday life. For example, the social network Facebook generates huge amounts of data, which might be of interest to market research companies, advertisers, politicians and so on. The task is to analyze these data to extract some actual information which is useful to the interested parties. Classification is one method of extracting or sorting these data and one of todays most common method for classification is the Support Vector Machine. But applying SVMs to big data tasks introduces the problem of long computation times. Figure 2.1 displays the behavior of an SVM model training on the Adult data set (explained in Section 5) with a step size of 500. The time needed for the training with the different kernels versus the size of the training data set used for the modeling was measured and is shown. It is visible that the training time has a quadratic to cubic trend. The initial idea behind the investigation in this report was to reduce

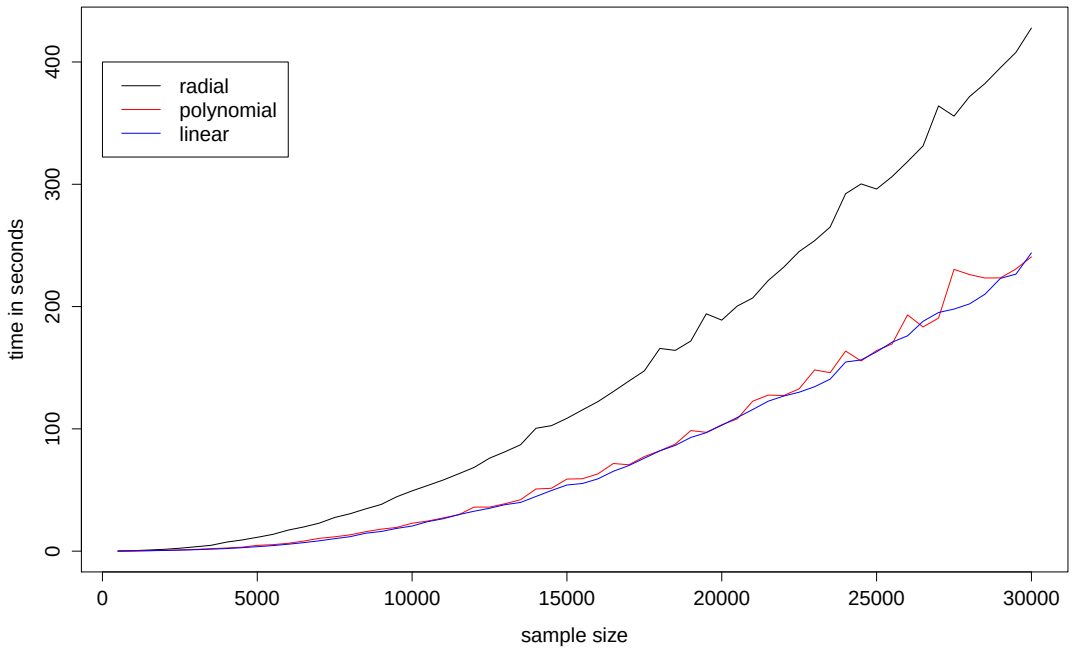


Fig. 2.1: Training times of single SVMs with the different kernels(radial, linear, 3rd degree polynomial) vs sampling size on the Adult data set wit a step size of 500

the amount of data used for the training of the SVM, but try to keep the quality of the classification as high as possible. Therefore a search for algorithms which are capable of obtaining the results was conducted and bagging and AdaBoost ensembles were identified as suitable methods. Both are capable of creating an ensemble of SVMs, where each SVM is trained with only a fraction of the data and then combining these to a single strong classifier. The goals of this report can be summarized to:

1. Reduction of the training data size for each SVM modeling
2. Keep the gain on the level of an single SVM trained with all data
3. Investigate the influence of introducing different kernel types to an ensemble

Actual research paper have also investigated methods to handle big data:

Kim et al. [15] covers SVM ensemble with bagging (bootstrap aggregating) or boosting using the different aggregation methods majority voting, least-squares estimation-based weighting and the double-layer hierarchical combining. They conclude that an SVM ensembles outperform a single SVM for all applications in terms of clas-

sification accuracy.

Li et al. [17] features a study of Adaboost SVMs using weak learners. They are adapting the kernel parameters for each SVM to get weak learners. They conclude that the AdaBoost performs better with SVMs than with neural networks and delivers promising results. They also mention the reduction in computational cost due to an less accurate model selection.

Meyer et al. [19] discuss bagging, cascade SVMs and a combination of both covering different data sets, gain and time comparisons. They have been able to significantly reduce the computation time by the use of a parallelized bagging approach, but the achieved gains are below the one of a single SVM. Their combined approach shows promising results, but still the gain is not optimal over all data sets.

Valentini [24] discusses random aggregated and bagged ensembles of SVMs with an analysis of the bias-variance. He concludes that the bias-variance is consistently reduced using bagged ensembles in comparison to single SVMs.

Wang et al. [25] make an empirical analysis of support vector ensemble classifiers covering different types of AdaBoost and bagging SVMs. They conclude that although SVM ensembles are not always better than single SVM for every data set, the SVM ensemble methods on average resulted in a better classification accuracy than a single SVM. Moreover, among SVM ensembles, bagging is considered the most appropriate ensemble technique for most problems for its relatively better performance and higher generality.

Yu et al. [28] introduces hierarchical cluster indexing as a method for Clustering-Based SVM (CB-SVM) for real world data mining applications with large sets. Their experiments show that CB-SVM are very scalable for very large data sets while generating high classification accuracy, but that they also suffer in classifying high dimensional data, because the scaling is here not optimal.

3 Basic Methods

3.1 Support Vector Machines

Support Vector Machines (SVM) [4] are a kernel-based or *modified inner product* technique, explained later in section 3.1.3 and represent a major development in machine learning algorithms. SVMs are a group of supervised learning methods that can be applied to classification or regression. SVMs represent an extension to nonlinear models of the generalized portrait algorithm developed by Corinna Cortes and Vladimir Vapnik. The SVM algorithm is based on the statistical learning theory and the Vapnik-Chervonenkis (VC) dimension introduced by Vladimir Vapnik and Alexey Chervonenkis.

3.1.1 Separable case

Support vector machines are meant to deal with binary and multiple class problems, where classes may not be separable by linear boundaries. Originally, these problems were developed to perfectly separate two classes by maximizing the space between the closest points of each class [4]. This provides two advantages, a unique solution is found to the separating hyperplane problem and by maximizing this margin on the training data, a better classification performance can be acquired on the test data [10]. Consider the case where a train set consists of N number of pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ with $x_i \in \Re^p$ and $y_i \in \{-1, 1\}$. The general maximization problem of the separable case is

$$\max_{\beta, \beta_0, \|\beta\|=1} M,$$
$$\text{subject to } y_i \left(x_i^T \beta + \beta_0 \right) \geq M, i = 1, \ldots, N, \tag{3.1}$$

where the condition ensures that the points are located at a signed distance from margin M, and which can be also described as a minimization problem by eliminating the parameter ($\| \beta \| = 1$) and setting $\| \beta \| = \frac{1}{M}$ as follows:

$$\min_{\beta, \beta_0} \left(\frac{1}{2} \| \beta \|^2 \right),$$
$$\text{subject to } y_i \left(x_i^T \beta + \beta_0 \right) \geq 1, i = 1, \ldots, N, \tag{3.2}$$

where M is the margin or space between the hyperplane and the closest points of the two classes. Thus the maximization of the thickness of this margin will be defined by β and β_0. This convex problem can be solved by minimizing the Lagrange function:

$$\mathcal{L}(\beta, \beta_0, \alpha_i) = \frac{1}{2} \| \beta \|^2 - \sum_{i=1}^{N} \alpha_i [y_i (x_i^T \beta + \beta_0) - 1]. \tag{3.3}$$

which derivatives are:

$$\frac{\partial \beta}{\partial \mathcal{L}} = \beta - \sum_{i=1}^{N} \alpha_i y_i x_i^T = 0, \tag{3.4}$$

$$\frac{\partial \beta_0}{\partial \mathcal{L}} = \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.5}$$

where if Equations 3.4 and 3.5 are substituted in 3.3, the dual Lagrange convex problem

$$\mathcal{L}_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k. \tag{3.6}$$

is obtained subject to $\alpha_i \geq 0$. And the solution can be solved by maximizing $\mathcal{L}_D$ with the Karush-Kuhn-Tucker conditions:

$$\alpha_i [y_i (x_i^T \beta + \beta_0) - 1] = 0, \forall i \tag{3.7}$$

Notice that to satisfy this, the following options must be considered:

– if $\alpha_i > 0$, then $(x_i^T\beta + \beta_0) = 1$, meaning that x_i lies on the boundary of the margin;
– if $(x_i^T\beta + \beta_0) > 1$, x_i will not lie on the boundary and thus $\alpha = 0$.

From these conditions, it is shown that for x_i to lie on the boundary as a *support point* of the classification, β is obtained by a linear combination from Equation 3.4 using $\alpha_i > 0$. β_0 can be obtained solving Equation 3.7 by substituting any of the support points x_i. Now the hyperplane function to classify new elements is:

$$\hat{f}(x) = x^T\hat{\beta} + \hat{\beta}_0, \tag{3.8}$$

with

$$\hat{G}(x) = \text{sign}\hat{f}(x). \tag{3.9}$$

This solution might work for the case when classes are perfectly separable, where just a linear hyperplane can give the optimum solution. For the non separable case, where a nonlinear solution is needed because the classes overlap and the optimum linear boundary is not enough, the support vector classifier considers the slack variables $\xi = (\xi_1, \xi_2, \ldots, \xi_N)$ for the points on the wrong side of the margin M, allowing the optimization problem to consider this overlapping [10].

3.1.2 Non separable case

Consider again the case where a train set consists of N number of pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ with $x_i \in \Re^p$ and $y_i \in \{-1, 1\}$. The hyperplane is defined in Equation 3.8 and its classification rule by Equation 3.9. This problem can be obtained by maximizing also the margin M but considering the slack variables and changing the conditions of Equation 3.1 to

$$y_i\left(x_i^T\beta + \beta_0\right) \geq M(1 - \xi_i), i = 1, \ldots, N, \tag{3.10}$$

$\forall i$, $\xi_i > 0$, $\sum_{i=1}^N \xi_i <$ constant, where Equation 3.10 defines the amount by which prediction 3.8 is on the wrong side of the margin. Hence by adding the constraint $\sum_{i=1}^N \xi_i < K$ bounds the optimization problem to a total proportional amount by which points fall beyond their margin, where misclassifications occur if $\xi_i > 1$ and the $\sum_{i=1}^N \xi_i$ can be bounded to a limited K.
Now the maximization problem can be defined as the minimization problem, like shown in Equation 3.2, considering the slack variables as:

$$\min_{\beta,\beta_0}\left(\frac{1}{2} \parallel \beta \parallel^2\right) \text{ subject to } \begin{cases} y_i\left(x_i^T\beta + \beta_0\right) \geq (1 - \xi_i), \forall i \\ \xi_i \geq 0, \\ \sum_{i=1}^N \xi_i < K \end{cases} \tag{3.11}$$

which can be rewritten as:

$$\min_{\beta,\beta_0}\left(\frac{1}{2} \parallel \beta \parallel^2 + C\sum_{i=1}^N \xi_i\right) \text{ subject to } \begin{cases} y_i\left(x_i^T\beta + \beta_0\right) \geq (1 - \xi_i), \forall i \\ \xi_i \geq 0 \end{cases} \tag{3.12}$$

where the constant K is now replaced by the *cost* parameter C to balance the model fit and the constraints. The case where a full separation is achieved is determined by $C = \infty$ [10]. This problem, again, is a convex optimization problem considering the slack variables, and can be solved by the Lagrange multipliers:

$$\mathcal{L}(\beta, \beta_0, \xi_i, \alpha_i, \mu_i) = \frac{1}{2} \parallel \beta \parallel^2 + C\sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^N \mu_i\xi_i, \tag{3.13}$$

which derivatives are:

$$\frac{\partial\beta}{\partial\mathcal{L}} = \beta - \sum_{i=1}^N \alpha_i y_i x_i^T = 0, \tag{3.14}$$

$$\frac{\partial\beta_0}{\partial\mathcal{L}} = \sum_{i=1}^N \alpha_i y_i = 0. \tag{3.15}$$

$$\frac{\partial\xi_i}{\partial\mathcal{L}} = C - \alpha_i - \mu_i = 0, \forall i. \tag{3.16}$$

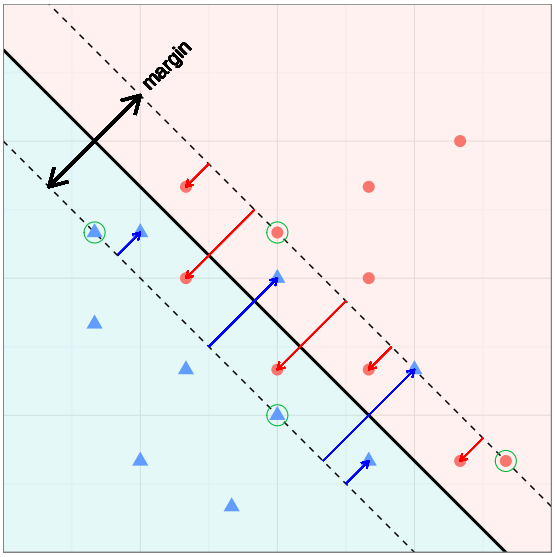


Fig. 3.1: Support vector classifiers for the non separable case where the cost C was tuned to consider some observations i besides the support points surrounded with the green circle. The arrows show the points that lie on the wrong side of the margin.

where if Equations 3.14 to 3.16 are substituted in 3.13, the Lagrange dual problem can be obtained as:

$$\mathcal{L}_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k, \tag{3.17}$$

and maximized subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$ to obtain the objective function for any feasible point. The Karush-Kuhn-Tucker conditions for this problem are:

$$\alpha_i [y_i (x_i^T \beta + \beta_0) - (1 - \xi_i)] = 0, \tag{3.18}$$

$$\mu_i \xi_i = 0, \tag{3.19}$$

$$y_i (x_i^T \beta + \beta_0) - (1 - \xi_i) \geq 1, \tag{3.20}$$

for $i = 1, 2, \ldots, N$. β can be obtained from Equation 3.14 for all the nonzero α_i using those observations i that satisfy the constraint 3.20. This observations are then called the *support vectors*, where some of them will lie on the edge of the margin ($\xi_i = 0$) having $0 < \alpha_i < C$ and some will not ($\xi_i > 0$) having $\alpha_i = C$. β_0 can be solved using the margin points ($\xi_i = 0$). Maximizing 3.17, knowing β and β_0, the optimum decision function can be defined as:

$$\hat{G}(x) = \text{sign} \hat{f}(x). \tag{3.21}$$

The cost parameter C can be tuned respectively to obtain a soft margin including an specific amount of observations i. Notice that if this parameter is too high, the solution can lead to over fitting. Figure 3.1 shows an example of the support vector classifier for the non separable case just discussed.

3.1.3 Kernels and Support Vector Machines

So far, it has been described how to find the linear boundary of the input space. The procedure to find the boundary the problem can be extended by using polynomial or spline functions. This extension, referred as *support vector machines* allows the separation to be more accurate by using this functions.
First, the linear combinations of input features $r_m(x_i)$, representing basis functions, can be introduced to the optimization problem of Equation 3.13 by transforming the vector feature and obtain the inner products without

too much cost. Hence, from the Lagrange dual problem,

$$\mathcal{L}_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k \langle r(x_i), r(x_k) \rangle, \tag{3.22}$$

where $\langle r(x_i), r(x_k) \rangle$ is the inner product of the transformed input features, the solution function is

$$\begin{aligned} f(x) &= r^T(x)\beta + \beta_0 \\ &= \sum_{i=1}^{N} \alpha_i y_i \langle r(x), r(x_i) \rangle + \beta_0 \end{aligned} \tag{3.23}$$

using only the inner product of $r(x)$. By knowing the kernel function,

$$K(u, v) = \langle r(u), r(v) \rangle \tag{3.24}$$

this inner product must not be specified. The kernel functions used in this case study research are:

$$\begin{aligned} \text{Linear: } K(u, v) &= \langle u, v \rangle, \\ n\text{th-Degree polynomial: } K(u, v) &= (1 + \langle u, v \rangle)^n, \\ \text{Radial basis: } K(u, v) &= exp^{(-\gamma \|u - v'\|^2)}. \end{aligned}$$

3.2 Ensemble Methods

3.2.1 SVM Bagging

Bagging, which is an abbreviation of bootstrap aggregating, was first introduced by Breiman [1] to be used with decision trees [2], but can also be applied to other methods. It was constructed to improve the accuracy and stability of machine learning algorithms for classification and regression problems. The algorithm is as follows: The training set given by T with size n is sampled uniformly with replacement to create m new training sets T_i. Each training set has the size $n^* < n$. By sampling with replacement, some observations are repeated in each T_i, leading to an expected fraction of 63.3% of unique samples in the set T_i for large n and $n^* = n$. Each training set predictor is then aggregated by majority voting, creating an single predictor. Due to Breimans paper [1], bagging has shown that it can give substantial gains in accuracy. He pointed out that the stability of the prediction method is the key factor for performance of bagging. If the constructed predictor has significant changes for the different samples of the learning set, thus is unstable, it can improve the overall accuracy. If the predictor is a stable learner, it can degrade the performance. Example for unstable learners would be neural nets or classification or regression trees, while methods like K-nearest neighbors are seen as stable. SVMs are stable learners [22] so the bagging method is adjusted to introduce significant changes in the different learning sets. This is done by significantly reducing the amount of samples per SVM which also reduces greatly the computation time and memory usage per SVM training. The aggregation method for the classification is also not the often used voting, where each predictor in the bagging ensemble has one vote per class. Instead the, by the here used SVM implementation, provided probability models are used to have a more distinguished aggregation, where also the strength of the class prediction has influence for the final prediction. This prediction strength is not to mistake with the unstable or stable learners, which are in the literature also referred to as *strong(stable)* or *weak(unstable)* learners. It here defines the quality of the prediction per case. Strong predictions are, where the algorithm was capable of choosing a class with an high probability. This is seen as very beneficial to the whole process. Table 3.1 shows an example and also a comparison to the often used majority voting for a two-class prediction. As shown in the Table, the strong prediction classifier has a high probability of choosing the second class, while the two weak classifier have a near equal probability for both classes. In an ensemble using majority voting these would still dominate the overall prediction, while with the here used probability voting clearly prefers the class with the aggregated higher probability. If the probability voting really has the indented positive effect on the accuracy will be tested in the experiment Section 6 and later discussed in Section 7.

Table 3.1: Probability aggregation vs majority voting showing the different influence of weak classifiers, the strong prediction classifier has a high probability of choosing the second class, while the two weak classifier have a near equal probability for both classes

classifier strength	class 1 probability	class 2 probability	class 1 vote	class 2 vote
weak	51	49	1	0
weak	53	47	1	0
strong	20	80	0	1
aggregated	**123**	**177**	**2**	**1**

Another difference from Breimans bagging algorithm is the sampling method for the learning sets. As described, the original bagging uses sampling with replacement. This introduces duplicate data, while in this implementation sampling without replacement is used to have as much unique data per predictor as possible. This is done because of two reasons: First reason is that for a high computation speed the amount of training data per SVM is to be reduced. Second reason is that it is a key factor for the accuracy of bagging to have unstable classifiers and thus a difference in the predictors as high as possible. To achieve this high difference, the SVM bagging algorithm also introduces the option to use different kernel types(radial, linear, polynomial) in one bagging ensemble. Figure 3.2 shows a schematic diagram of the complete bagging process. The here implemented SVM bagging process is easily parallelized by attaching each predictor to one thread or kernel, which makes it a good choice a multi-core CPU or computer cluster.

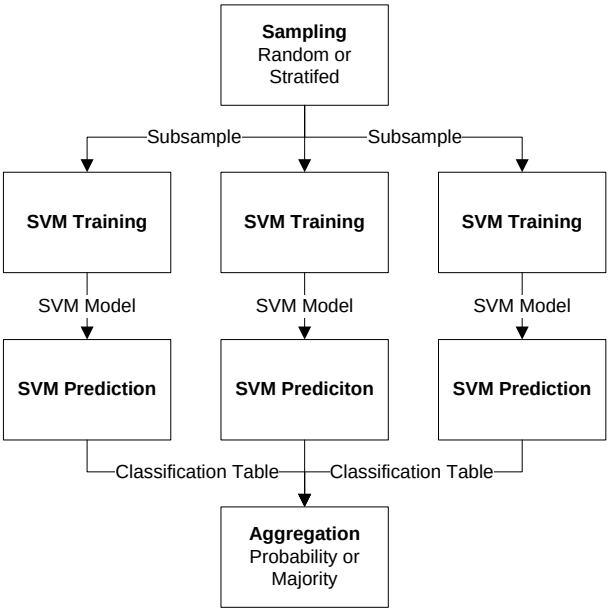


Fig. 3.2: Schematic showing the SVM bagging method

3.2.2 Boosting

Boosting has been one of the most important developments in classification problems in the last 10 years. The basic motivation is to combine many *weak* classifiers as ensemble to produce a powerful classification committee [7]. The boosting algorithms discussed in this paper is the AdaBoost for two class problems from Freund and Schapire [7] and for multi class problems explained in [29].

Two-class problems
Consider a set with an output labeled $Y \in [-1, 1]$ where given a vector of predictor variables X, a classifier $H(X)$

produces a prediction taking one of the two class values. Hastie et al. define a *weak classifier* as one whose error rate is slightly better than random guessing, where the error rate is defined by:

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq H(x_i)). \tag{3.25}$$

Boosting applies a weak classification algorithm repeatedly to resample the data, producing many weak classifiers $h_m(X), m = 1, 2, \ldots, S$. The predictions are then combined to obtain a final prediction of the data:

$$H(x) = \text{sign}\left[\sum_{m=1}^{S} \alpha_m h_m(x)\right]. \tag{3.26}$$

α_m is called the *goodness of classification* and is computed by the algorithm based on the error of classification ϵ_m to weight the contribution of each respective $h_m(x)$, and its purpose is to give more weight to the more accurate classifiers of the sequence. After every iteration, the data is modified by changing the weight w_m of each observation $(x_i, y_i), i = 1, 2, \ldots, N$, where initially they were set equally to $1/N$, in such a way that the first time the data is sampled normally. At every step, the weights of those miss-classified observations are increased, whereas the weight for the good classified observations are decreased to be less selected for the next modification of the data, which is going to be used for the prediction $h_m(x)$. Algorithm 3.1 presents the AdaBoost method for a two class problem used in this research.

Algorithm 3.1: AdaBoost algorithm for two-class problems.

input : Train set Γ with pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, n samples and labels $y_n \in Y = \{-1, 1\}$
Initialize the observation weights: $\mathbf{w}_i = 1/N, i = 1, 2, \ldots, N$.
for $(m \leftarrow 1$ **to** $S)$ **do**
  Fit a Classifier $h_m(x)$ to the training data using weights $\mathbf{w}_i$.
  Compute $\epsilon_m = \sum_{i=1}^{N} \mathbf{w}_i I(y_i \neq h_m(x_i))$
  Compute $\alpha_m = ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$.
  Set $\mathbf{w}_i \leftarrow \frac{\mathbf{w}_i \cdot exp[\alpha_m \cdot I(y_i \neq h_m(x_i))]}{Z_m}, i = 1, 2, \ldots, N,$
  where Z_m is the normalization factor to make $\sum_{i=1}^{N} \mathbf{w}_i = 1$.
end
output: $H(x) = \text{sign}\left[\sum_{m=1}^{S} \alpha_m h_m(x)\right]$.

Multi-class problems
Consider a set with an output labeled $Y \in \{1, \ldots, C\}$, where given a vector of predictor variables X, a classifier $H(X)$ produces a prediction taking one of the C class values. The weak classifiers are $h_m(X), m = 1, 2, \ldots, S$ and are then combined to obtain a final prediction of the data:

$$H(x) = \arg\max_{m}\left[\sum_{m=1}^{S} \alpha_m [h_m(x) == Y]\right], \tag{3.27}$$

The multi-class method, proposed by Zhu et al., used for this research is presented in Algorithm 3.2.

4 Implementation

4.1 SVM AdaBoost

The AdaBoost implementation in this case study research is an extension and combination of the two available options described in section 3.2.2. The same algorithm 4.1 was used for all two type of classification problems. A modification of the ME algorithm presented by Zhu et al. in [29] and [30] is introduced as well as the 0.5ME version. The addition of the parameter Cl_{type}, as shorthand for *Classification type*, to the Algorithm 4.1, helps it

Algorithm 3.2: AdaBoost algorithm for multi-class problems.

input : Train set Γ with pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, n samples and labels $y_n \in Y = \{1, \ldots, C_n\}$
Initialize the observation weights: $\mathbf{w}_i = 1/N, i = 1, 2, \ldots, N$.
for $(m \leftarrow 1$ **to** $S)$ **do**
  Fit a Classifier $h_m(x)$ to the training data using weights $\mathbf{w}_i$.
  Compute $\epsilon_m = \sum_{i=1}^{N} \mathbf{w}_i I(y_i \neq h_m(x_i))$
  Compute $\alpha_m = ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right) + ln(C_n - 1)$.
  Set $\mathbf{w}_i \leftarrow \frac{\mathbf{w}_i \cdot exp[\alpha_m \cdot I(y_i \neq h_m(x_i))]}{Z_m}, i = 1, 2, \ldots, N,$
  where Z_m is the normalization factor to make $\sum_{i=1}^{N} \mathbf{w}_i = 1$.
end
output: $H(x) = \arg\max_m \left[\sum_{m=1}^{S} \alpha_m [h_m(x) == Y]\right]$.

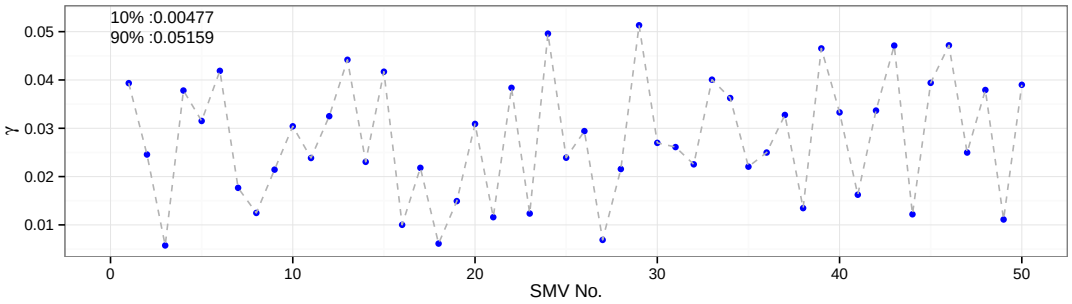


Fig. 4.1: Estimated γ for Spam task on a 50 SVM-AdaBoost ensemble uniformly distributed from 10% and 90% quantiles of $|u - v'|^2$.

produce the expected task, either if it is a two or multi classification problem. The different independent selection of desired task will produce the goodness of classification ($alpha$). The implemented prediction for a two class problem is shown in Equation 3.26 and for the multiclass problems in Equation 3.27.

From Algorithm 4.1, notice that in the switch clause for case $multi$, a variation of the algorithm presented in [29] and [30], is introduced as the $0.5ME$ for multi-class problems. Also notice that if the number of classes in C_n is 2, and the Cl_type option selected is $multi$, the problem reduces to a two class problem as presented in Algorithm 3.1, this switch case is shown only for presentation purposes of the variation explained before.

4.1.1 Gamma (γ) Estimation

For the experiments where the Radial Basis kernel was used, the γ parameter was calculated by building a vector of uniformly distributed γ values from 10% to 90% quantile range of $|u - v'|^2$ as suggested in [3]. The vector size depends on the ensemble size to train. Figure 4.1 shows an example of a 50 SVM-AdaBoost ensemble estimated γ parameters.

Algorithm 4.1: SVM-AdaBoost algorithm implemented in this paper.

input : Train set Γ_r with (x_n, y_n) features, n samples and labels $y_n \in Y = \{1, \cdots, C_n\}$
input : Number of SVMs to build the ensemble m_{svm}
input : Factor size to resample train inside AdaBoost $bo.size$
input : The classification problem or algorithm to use $Cl_{type} = ("two", "multi")$
input : The kernel type to use on the next ensemble: $pars\$kernel$
input : The mixed kernel ensemble selection: $pars\$mixed \leftarrow ("TRUE", "FALSE")$
input : The kernels to use on the mixed ensemble: $kernel.list \leftarrow ("radial", "polynomial", "linear")$
input : The Cost parameter for each kernel: $(pars\$rad\$C)(pars\$poly\$C)(pars\$linear\$C)$
input : The gamma parameter for the radial kernel SVMs: $(pars\$rad\$gamma)$
input : The breaking tolerance to terminate AdaBoost algorithm: $(pars\$brTol)$
input : The maximum number of allowed resets inside AdaBoost: $(pars\$cntbr)$
initialize:
The weight vector according to the number of samples: $\mathbf{w}(i)_1 = 1/n$
for $(m \leftarrow 1$ **to** $m_{svm})$ **do**
  Sample Γ_r with replacement based on the weight vector $\mathbf{w}_m$ and build a new train set Γ_m used to train next model SVM_m.
  if $pars\$mixed$ **then**
    Randomly select the next kernel type from $kernel.list$: $pars\$kernel \leftarrow kernel.list$
  Train model SVM_m using Γ_m: $h_m \leftarrow svm(\Gamma_m, pars)$.
  Re-sample a new training set Γ_m using $bo.size$ by stratified sampling: $\Gamma_m \leftarrow \Gamma_m \times bo.size$.
  Predict using the last trained model h_m.
  Calculate the error $\epsilon_m = \sum_{i=1}^{N} \mathbf{w}_i I(y_i \neq h_m(x_i))$
  Calculate goodness of classification α depending on the Cl_{type}:
  switch Cl_{type} **do**
    case two
      $\alpha_m = 0.5 \times \ln(\frac{\epsilon_m - 1}{\epsilon_m})$
    case $multi$
      $\alpha_m = 0.5 \times \left(\ln(\frac{\epsilon_m - 1}{\epsilon_m}) + \ln(n_C - 1)\right)$
  endsw
  Obtain $\mathbf{w}_{m+1} = \mathbf{w}_m \times \exp(\alpha_m)|\{i|h_m \neq y_i\}|$
  Normalize vector $\mathbf{w}_{m+1} = \frac{\mathbf{w}_{m+1}}{\sum_{i=1}^{n} \mathbf{w}(i)_{m+1}}$,
end
output: The models formed inside the ensemble: $results\$kernel\$svms$
output: The alphas for each model inside the ensemble: $results\$kernel\$alphas$

4.2 SVM Bagging

The implementation of the SVM bagging algorithm was done in R. It uses the SVM implementation of the {e1071} package. The complete bagging algorithm was split into modular steps. All algorithms are implemented as parallel processes so that they can utilize the performance of multi-core CPUs or clusters. The sampling of the data is the

first step. This can be done by either random or stratified sampling. Stratified sampling is hereby seen as very beneficial to multi class problems.

Algorithm 4.2: Random Sampling

input : Training dataset Trn with n samples
input : desired sample size n^* for each subset
input : desired ensemble size m, number of training subsets
for $k \leftarrow in\ m$ **do**
  draw n^* random values out of Trn without replacement
end
output: Set Trn_m of m Training subsets with n^* samples each

Algorithm 4.3: Stratified Sampling

input : Training dataset Trn with with n samples
input : desired sample size n^* for each subset
input : desired ensemble size m, number of training subsets
input : name of the class prediction feature column
for $k \leftarrow in\ m$ **do**
  sort data by prediction feature(class)
  estimate fractions fr for each class
  draw n^* $\times$ the respective fr random values out of every class in Trn without replacement
  combine class samples to get stratified sample
end
output: Set Trn_m of m stratified training subsets with n^* samples each

Stratified sampling creates a stratified sample for each data set, this is important for low sample sizes in combination with multi class problems. Table 4.1 shows an comparison between random and stratified sampling. The original class distribution is shown with two different random samples in comparison to the stratified sample for a sampling fraction of 10%. It is visible that for the random samples the class distribution is different from the original data. In the second example the third class gets no cases, which can lead to crashes of the algorithm. The stratified sample has the same class distribution as the original data, which is seen as beneficial to the algorithm and also avoids crashes.

Table 4.1: comparison of random vs stratified sampling for a three class problem with 10% data per subset

Data set	number class 1 cases	number class 2 cases	number class 3 cases	total
orginal	2000 / 67%	800 / 26%	200 / 7%	3000
random sampling 1	150 / 50%	30/ 10%	120/ 40%	300
random sampling 2	280 / 93%	20 / 7%	0 / 0%	300
stratified sampling	200 / 67%	80 / 26%	20 / 7%	300

The set of training subsets is then used as an direct input for the modeling of the SVM. The algorithm features a dynamic pass-through for all parameters used by the {e1071} SVM function, so all parameters defined in this function can be used.

Algorithm 4.4: SVM modeling

input : Training subsets Trn_m
input : name of the class prediction feature column
input : SVM kernel parameters KP
for $k \leftarrow in\ m$ **do**
  train SVM model with class prediction probability for each training subset in Trn_m with the defined KP
end
output: set of SVM models SVM_m

In the next step, the training probability models are used to predict the classes on the given test data. There is also an option to convert the probability model to an basic voting model here. This is done by setting the class with the highest probability for each data point to 1 and the other classes to 0.

Algorithm 4.5: SVM prediction

input : SVM models SVM_m
input : test data set Tst
input : SVM parameters
for $k \leftarrow in\ m$ **do**
  create class prediction for every SVM model for Tst
  optional: convert probability to basic voting model
end
output: class predictions P_m

In the end the aggregation is done summing up the probabilities/votes for each data point in the class predictions and choosing the class with the highest probability sum or most votes to be the specific prediction. Here is also the option to use cutoffs to have a weighting of the different classes.

Algorithm 4.6: result aggregation

input : class predictions P_m
input : optional: cutoffs
for $k \leftarrow in\ m$ **do**
  sum up probabilities or votes for each data point
  optional: apply cutoffs
  estimate max for each data point to get result class
end
output: class prediction table for each data point in the test set Tst

5 Experiments

5.1 Data Sets

The benchmark Data Sets selected for these experiments were obtained from the UCI Repository [6] to analyze the behavior of SVM Ensembles with different classification problems. The selection of data sets was made to compare the work of this case study with different results proposed in [25] and to analyze the performance of SVM ensembles with bagging using large data sets with many features. The selection of data sets, which are freely available and often used for benchmarking, enables an easy comparison to other algorithms and also ensures a certain amount of generalization of the upcoming results. Table 5.1 shows the properties for each data set used in this research.

*Table 5.1: Data sets used in this research. Those rows with a * are data sets that were randomly sampled by 2/3 of the full set to form the train set. The rest were already separated in test and train sets.*

Name	Records	Train Size	Features	Classes	Labels
*Spam	4601	3067	57	2	is spam (yes, no)
Satellite	6435	4435	36	6	soil type (1,2,3,4,5,7)
OptDig	5620	3823	64	10	digits (0 to 9)
Adult	45222	30162	14	2	yearly income ($<$\$50K, $\geq$\$50K)
Acoustic	98528	78823	50	3	vehicle class 1 to 3
Acoustic Binary	98528	78823	50	2	binarized (class 3 against others(1 & 2))

5.1.1 SPAM

The SPAM Data Set was originally donated by Hewlett-Packard Labs in 1999 to the UCI Repository. It is a two class problem to classify emails as spam or not spam. It consists of 57 features plus the class column. The total number of instances is 4601 where 2788 (60.6%) samples are nonspam and only 1813 (39.4%) are spam. From these samples, 3067 were used to train and 1534 for testing. To avoid scaling issues with SVMs the data was scaled first before its use.

5.1.2 Adult

Donated in 1996 to the UCI Repository, the main purpose of the data is to classify if the income of a citizen in the USA exceeds $50K/year or not. It consists on 14 features plus the class column. The total number of instances without missing values is 45222 where 34014 samples are for income less than $50K and 11208 for income more than $50K. For the experiments 30162 samples were used to train and 15060 for testing. The data was scaled before its use and columns "fnlwgt", "race" and "country" were eliminated for their low importance on the data set.

5.1.3 Satellite

The Landsat Satellite data set contains multi spectral values of pixels in 3x3 neighborhoods in a satellite image and the classification associated with the central pixel [6]. It consists in 36 features plus the class column where the available types are 1 for "red soil", 2 for "cotton crop", 3 for "grey soil", 4 "damp grey soil", 5 for "soil with vegetation stubble", 6 "mixture class" and 7 for "very damp grey soil". The has 6435 samples in total where 1994 are for class 1, 1029 for class 2, 1949 for class 3, 884 for class 4, 964 for class 5, 0 for class 6 and 2050 for class 7. For training 4435 samples were used and for testing 2000.

5.1.4 Optical Recognition of Handwritten Digits

This data set is a pre-processed set of handwritten digits, where the aim is to classify those digits. Populated with 5620 samples where there are 10 classes from 0 to 9, distributed as follows, 0 with 554, 1 with 571, 2 with 557, 3 with 572, 4 with 568, 5 with 558, 6 with 558, 7 with 566, 8 with 554 and 9 with 562. The data set is composed by 64 features plus the class column. 3823 samples were used to train and 1797 to test.

5.1.5 Acoustic

The Acoustic data set [5] is created for Vehicle type classification by acoustic sensor data. This is a widespread military and civilian application and used for e.g. intelligent transportation systems. There are three different classes which represent different military vehicles which where used in the experiments. The data set has a total of 98528 entries, form which are 78823 used for training. It covers 50 different features. For an easier classification, also the binary case in which class 1 and 2 were combined to one class is investigated. This leads to an nearly perfect class distribution of 50/50.

5.2 Experimental Setup

Different experiments were conducted for the two proposed ensemble methods, namely Bagging and AdaBoost, on 5 data sets available on the UCI repository [6]. The general experiments to compare results against each kernel ensemble by using the average performance of ten runs.

5.2.1 Results for Bagging

To analyze the performance of Bagging, the behavior of the method is tested in different cases, which estimate the influence of the sample size, the ensemble size and also different aggregation methods. To see the goodness of the gain, first single SVM runs with each kernel type and the complete training data were conducted. For this tests also the model training time was measured. For all runs, an experiment script is set up, which allows to change the parameters. All runs were conducted with three different kernel types linear, polynomial and radial and their receptive combinations. The naming schema is as follows:

- **LinRad** linear and radial kernel combined
- **RadPol** radial and polynomial kernel combined
- **LinPol** linear and polynomial kernel combined
- **LinRadPol** linear, radial and polynomial kernel combined
- **Radialx3** radial kernel for each training set and then combined

The ensemble size for the combinations of kernel is added up, resulting in a higher total number of SVMs for each. So the RadPol, LinRad and LinPol have twice the number of SVMs and LinRadPol and Radialx3 have three times the number. Radialx3 is is added to see if the combination of different kernels or the higher number of SVMs has a greater influence on the results. All tests were conducted on an Intel Core i5 2500k (4cores/4 threads) with 8GB of RAM with R version 2.15.2.
The general setup:

test parameter	spam, optdig, satellite	adult, acoustic, acoustic binary
ensemble size	10,20,30,40,50 with 300 sample size	10,20,30,40,50 with 500 sample size
sample size	300 to 2700 step 300 with 10 ensemble size	500,1000,2000,4000 with 10 ensemble size

Also the Connect4 data set was tested, but as the results were difficult to interpret, it is discussed separately. Before executing the runs, a tuning for the cost, degree and cutoff parameter was conducted. This was done for each data set and with each kernel and a single SVM. It was tried to use the hereby gained information for the SVM bagging, but early experiments have indicated that the tuning parameters were not giving the best accuracy for the SVM bagging algorithm. The degree and Coeff0 for the of the tuning were used in the experiments, but for the cost, a simple rule-of-thumb approach was used. The radial gamma parameter was for most data sets calculated by the internal gamma estimation of the SVM algorithm. For the OptDig set, these procedure failed and gave poor accuracies, therefore here the sigest estimation method was used. The kernel parameters for each run were as follows:

Data Set	Sample Method	Radial Gamma and Cost	Poly Cost, Coeff0 and Degree	Linear Cost
Spam	Random	auto, 10	10, 0.67, 3	10
Satellite	Stratified	auto, 10	10, 0.67, 3	0.1
OptDig	Stratified	sigest, 10	10, 0.67, 3	10
Adult	Random	auto, 10	10, 0.67, 3	0.1
Acoustic	Random	auto, 10	10, 0.67, 3	10
Acoustic Binary	Random	auto, 10	10, 0.67, 3	10

The procedure shown below is the experiment loop used for the different experiments.

Algorithm 5.1: Experimental loop for Bagging

input : Train set Trn with (X_i, Y_i) with i samples
input : Test set Tst with (X_i, Y_i) with i samples
input : Prediction Feature of the data for the SVMs
input : Ensemble Size ES
input : Sample Size SS
input : fixed random seed
input : The gain matrix for the data set, if available gm
input : A parameter list $params$ including kernel parameters, cutoffs, samling and aggregation method
for $k \leftarrow in\ ES$ **do**
  for $j \leftarrow in\ SS$ **do**
    for $m \leftarrow in\ seed$ **do**
      set radnom seed
      For each kernel type, sample $ES\times$ train sets with stratified or random sampling
      Trn_1, Trn_2, Trn_3

      $radial \leftarrow$ create SVM models using Trn_1 for radial kernel using $Bagging$ algorithm .
      $polynomial \leftarrow$ create SVM models using Trn_2 for polynomial kernel using $Bagging$ algorithm .
      $linear \leftarrow$ create SVM models using Trn_3 for linear kernel using $Bagging$ algorithm .
      $radialx3 \leftarrow$ create SVM models using Trn_1, Trn_2, Trn_3 for radial kernel using $Bagging$ algorithm .
      $RadPol \leftarrow$ combine radial and polynomial models.
      $LinPol \leftarrow$ combine polynomial and linear models.
      $LinRad \leftarrow$ combine radial and linear models.
      $RadLinPol \leftarrow$ combine radial, linear and polynomial models.
      Calculate predictions for all normal and combined SVM models
      Aggregate results using majority voting or probability aggregation
      Calculate accuracy of classification and save results
    end
  end
end
output: data frame with results

5.2.2 AdaBoost

The independent experiments for AdaBoost are intended to show the accuracy and internal functionality of the algorithm with SVMs. Considering 10 runs for each ensemble size, the experiments were conducted with 1, 3, 5, 7, 10, 20, 30 and 50 SVMs plus the three kernels per ensemble, leading to (10+30+50+70+100+200+500)*3, giving a total of 2880 runs for each experiment, where a run consists of one iteration of the loop presented in Procedure *Experimental loop for SVM-AdaBoost.*.

1. Besides the three kernel types selected to build the ensembles, an extra ensemble was built using a random mixture of each kernel type, adding then another 960 experimental runs to the 2880. These experiments will be referred as "Mixed-kernel Ensemble".
2. Related to the all-combined ensemble, for AdaBoost a second combination is considered where only the radial and polynomial ensembles are combined, which will be known as "RadPol Ensemble". The ensemble size will be given by the sum of the ensemble sizes of Radial and Polynomial.
3. Wickramaratna et al. [26] state that boosting a strong learner generally leads to performance degradation. To prove this fact, the next experiment is intended to show that if a boosting factor ($bosize$) of the original train set is introduced after AdaBoost resample to create a weak classifier, a performance improvement can be achieved from the algorithm. And that if the full train is used, no improvement shall be shown. These experiments are referred as "$bosize$ boosting factor" and "Full Train", where $bosize$ is the boosting factor to reduce the train size inside AdaBoost after resampling the train set.

As general purpose experiments, the following methods or procedures are considered on the runs only on specific data sets:

1. An automatic estimation of the gamma parameter for the radial kernel types is considered for all the experiments and data sets.
2. Internally AdaBoost learns from weak classified samples to rebuild a weight vector and use it to resample the next train set that will be used to train the next model. It will be analyzed how many times every single sample of the train set is selected by AdaBoost resample process by changing its weight and observe the behavior of the most and less selected sample by building a 10 SVM ensemble.
3. Linked to the performance of SVMs inside the ensemble, it will be analyzed the increments or decrements of Support Vectors of the internal models while the iterations increase to show the connection between the performance of the ensembles and the adaptive algorithm by building a 50 SVM ensemble.

The Experimental Loop
The experimental loop used to collect information from the AdaBoost Algorithm 4.1 is presented in the Procedure called Experimental loop for SVM-AdaBoost..

Procedure Experimental loop for SVM-AdaBoost.

input : Train set Γ with (X_i, Y_i) pairs and i samples
input : Test set Γ_s with (X_i, Y_i) pairs and i samples
input : Number of SVMs, in a vector $\mathbf{m_{svm}} = [1, 3, 5, 7, 10, 20, 30, 50]$
input : The training size factor Γ_{size}
input : Number of maximum runs $r_{max} = 10$
input : The cost matrix for the data set, if available cm
input : Factor size of resampling inside AdaBoost $bo.size$
input : The classification problem or algorithm to use $Cl_{type} = (\text{"}two\text{"}, \text{"}multi\text{"})$
input : The parameters for the different kernel types used inside AdaBoost $pars$
for $k \leftarrow in$ $\mathbf{m_{svm}}$ **do**
  for $j \leftarrow in$ 1 **to** r_{max} **do**
    Sample an alternate train set Γ_r from train Γ and Γ_{size}: $\Gamma_r \leftarrow sample(1 : i, \Gamma_{size} \times \Gamma)$

    $rad.ens \leftarrow$ Form the ensemble for radial kernel using $AdaBoost$ algorithm 4.1.
    $poly.ens \leftarrow$ Form the ensemble for polynomial kernel using $AdaBoost$ algorithm 4.1.
    $linear.ens \leftarrow$ Form the ensemble for linear kernel using $AdaBoost$ algorithm 4.1.
    $mixed.ens \leftarrow$ Form the ensemble for mixed kernels using $AdaBoost$ algorithm 4.1.
    $radpol.ens \leftarrow$ Combine $rad.ens$ and $poly.ens$ for the radial-polynomial ensemble.
    $allcomb.ens \leftarrow$ Combine $rad.ens$, $poly.ens$ and $linear.ens$ for the all-combined ensemble.

    Using Cl_{type}, predict all ensembles independently using the Γ_s set with Equations 3.26 and 3.27.
    Calculate accuracy of classification and save results in $exp.res$.
  end
end
output: List of experiment results $exp.res$

6 Results

6.1 Bagging

6.1.1 Spam

Table 6.1: Spam data set, gain for single svm trained on complete training data with training time in seconds

SVM Type	Gain	Model Training Time
radial	93.68	2.67
linear	92.76	11.92
polynomial	93.28	1.74

Table 6.1 shows the behavior of the different kernel types for a modeling with the complete data, the time is given in seconds and the gain in %. The experiments were conducted once. The same kernel parameters as for the bagging tests were used. As the result shows all kernel types reach a goodness of about 93% and the difference in the gain between the kernel types is low. The linear kernel has a significant higher training time than the other kernels.

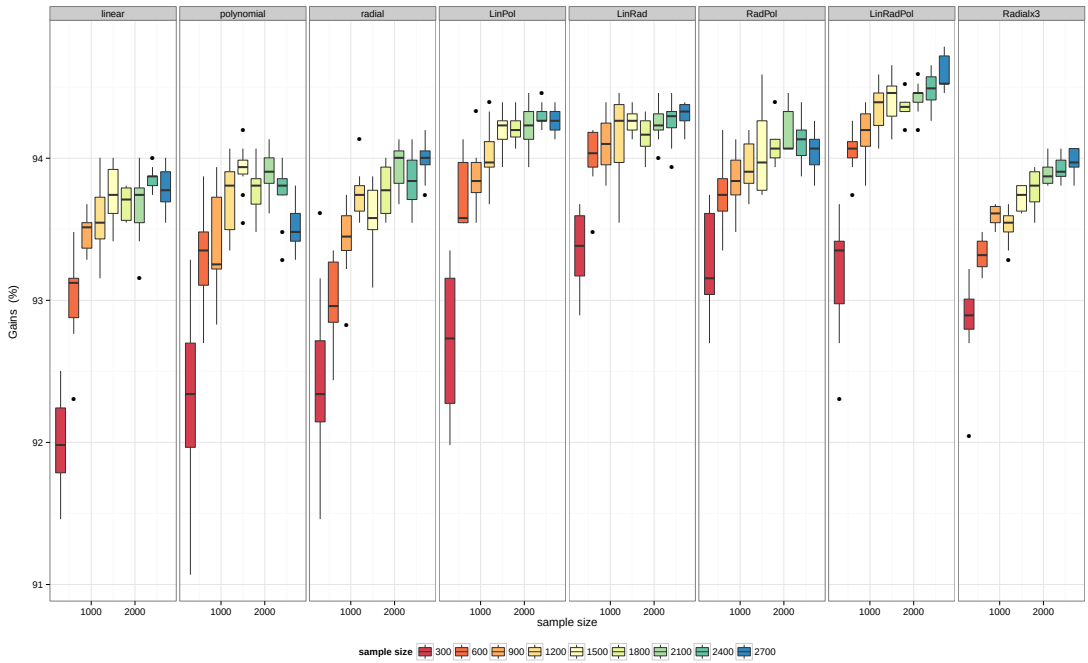


Fig. 6.1: Spam data set, boxplot with different kernels and their combinations, gain vs sample size

Table 6.2: Result table for the spam data set comparing different sample sizes with a fixed ensemble size of 10, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Sample Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
300	92.44 (±0.62)	92.27 (±0.65)	91.99 (±0.32)	93.25 (±0.36)	93.36 (±0.28)	92.71 (±0.52)	93.19 (±0.43)	92.85 (±0.32)
600	92.97 (±0.32)	93.31 (±0.35)	93.04 (±0.34)	93.75 (±0.24)	94.00 (±0.22)	93.74 (±0.25)	94.05 (±0.14)	93.33 (±0.12)
900	93.43 (±0.27)	93.40 (±0.35)	93.47 (±0.12)	93.83 (±0.20)	94.09 (±0.20)	93.87 (±0.21)	94.18 (±0.17)	93.59 (±0.08)
1200	93.75 (±0.18)	93.74 (±0.26)	93.59 (±0.25)	93.95 (±0.18)	94.16 (±0.30)	94.01 (±0.24)	94.35 (±0.17)	93.52 (±0.13)
1500	93.58 (±0.24)	**93.92 (±0.18)**	93.74 (±0.20)	94.05 (±0.30)	94.26 (±0.09)	94.20 (±0.13)	94.41 (±0.18)	93.72 (±0.09)
1800	93.78 (±0.18)	93.78 (±0.18)	93.68 (±0.11)	94.09 (±0.13)	94.17 (±0.12)	94.21 (±0.11)	94.36 (±0.08)	93.78 (±0.14)
2100	93.94 (±0.15)	93.90 (±0.16)	93.66 (±0.25)	**94.18 (±0.17)**	94.24 (±0.12)	94.22 (±0.16)	94.43 (±0.11)	93.89 (±0.08)
2400	93.85 (±0.20)	93.75 (±0.21)	**93.86 (±0.07)**	94.12 (±0.17)	94.25 (±0.15)	**94.30 (±0.08)**	94.49 (±0.12)	93.92 (±0.08)
2700	**93.99 (±0.14)**	93.51 (±0.15)	93.79 (±0.14)	94.05 (±0.14)	**94.30 (±0.09)**	94.27 (±0.09)	**94.60 (±0.13)**	**93.98 (±0.09)**

Figure 6.1 and Table 6.2 display the results of the test with different sample sizes with a fixed ensemble size of 10. All kernel types give strong results and the gain is the higher, the higher the sample size is. The combination of all kernels *LinRadPol* performs best and gives the best overall result (94.60) . The ensemble even outperforms the best single SVMs trainied on the complete data.

Table 6.3: Result table for the spam data set comparing different ensemble with a fixed sample size of 300, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Ensemble Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
10	92.47 (±0.40)	92.51 (±0.30)	92.05 (±0.76)	93.50 (±0.28)	93.55 (±0.17)	92.97 (±0.35)	93.55 (±0.20)	93.02 (±0.14)
20	92.72 (±0.35)	92.69 (±0.29)	92.57 (±0.27)	93.47 (±0.20)	93.45 (±0.13)	92.97 (±0.25)	93.44 (±0.21)	92.99 (±0.15)
30	92.87 (±0.30)	92.61 (±0.31)	92.65 (±0.40)	93.53 (±0.16)	**93.73 (±0.21)**	93.05 (±0.27)	**93.60 (±0.23)**	**93.12 (±0.15)**
40	92.87 (±0.32)	92.71 (±0.24)	**92.71 (±0.32)**	93.56 (±0.22)	93.55 (±0.20)	**93.14 (±0.17)**	93.52 (±0.16)	93.02 (±0.20)
50	**93.09 (±0.11)**	**92.82 (±0.27)**	92.59 (±0.29)	**93.61 (±0.13)**	93.68 (±0.12)	92.99 (±0.24)	93.56 (±0.09)	93.06 (±0.11)

Table 6.3 shows the results of the ensemble size testing with a fixed sample size of 300. The Table shows that a increasing ensemble size does not always lead to a higher gain. The *LinRad* combination has the overall best gain for an ensemble size of 30.

6.1.2 Satlog

Table 6.4: Satlog data set, gain for single svm trained on complete training data with training time in seconds

SVM Type	Gain	Model Training Time
radial	90.65	3.26
linear	85.4	2.18
polynomial	90.1	2.07

Table 6.4 displays the performance of the different kernel types for a training on the Satlog data set with the complete training data. It is visible that the radial and polynomial kernel perform best on this data set. The radial kernel is the slowest, but the difference in the training times is not high.

Table 6.5: Result table for the satlog data set comparing different sample sizes with a fixed ensemble size of 10, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Sample Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
300	87.78 (±0.31)	86.28 (±0.45)	83.70 (±0.20)	87.38 (±0.21)	86.17 (±0.35)	84.97 (±0.33)	86.23 (±0.24)	87.80 (±0.25)
600	88.93 (±0.28)	88.14 (±0.25)	84.16 (±0.30)	88.75 (±0.29)	87.14 (±0.23)	86.33 (±0.11)	87.61 (±0.21)	88.97 (±0.24)
900	89.50 (±0.26)	88.85 (±0.25)	84.41 (±0.12)	89.27 (±0.23)	87.53 (±0.35)	86.94 (±0.21)	88.19 (±0.19)	89.39 (±0.22)
1200	89.80 (±0.22)	89.41 (±0.21)	84.68 (±0.11)	89.67 (±0.19)	88.03 (±0.22)	87.61 (±0.25)	88.64 (±0.14)	89.84 (±0.16)
1500	89.97 (±0.36)	89.67 (±0.22)	84.77 (±0.17)	89.80 (±0.17)	88.11 (±0.21)	87.70 (±0.18)	88.83 (±0.19)	90.03 (±0.17)
1800	90.20 (±0.16)	89.62 (±0.25)	84.91 (±0.19)	89.94 (±0.15)	88.44 (±0.12)	87.94 (±0.17)	89.06 (±0.14)	90.08 (±0.11)
2100	90.16 (±0.24)	89.69 (±0.20)	85.06 (±0.09)	90.00 (±0.12)	88.73 (±0.20)	88.13 (±0.27)	89.25 (±0.06)	90.24 (±0.23)
2400	90.28 (±0.15)	89.80 (±0.24)	85.06 (±0.13)	90.06 (±0.07)	88.78 (±0.15)	88.28 (±0.19)	89.33 (±0.16)	90.15 (±0.09)
2700	**90.48 (±0.22)**	**90.09 (±0.19)**	**85.09 (±0.15)**	**90.26 (±0.11)**	**88.92 (±0.23)**	**88.49 (±0.19)**	**89.44 (±0.09)**	**90.35 (±0.14)**

Table 6.5 shows the results of the sample size test of the Satlog data set with a fixed ensemble size of 10. The gain rises with a higher sampling size for all kernels and combinations. The overall best result is gotten by the pure radial ensemble (90.48). No ensemble reaches the goodness of the best single SVM trained with the complete data.

Table 6.6: Result table for the satlog data set comparing different ensemble with a fixed sample size of 300, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in brackets

Ensemble Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
10	87.47 (±0.53)	86.17 (±0.52)	83.59 (±0.19)	87.14 (±0.37)	85.97 (±0.22)	84.78 (±0.34)	86.03 (±0.26)	87.58 (±0.22)
20	87.64 (±0.34)	86.30 (±0.39)	**83.69 (±0.26)**	87.30 (±0.28)	85.99 (±0.36)	84.91 (±0.16)	86.08 (±0.22)	87.68 (±0.17)
30	87.66 (±0.10)	86.45 (±0.40)	83.66 (±0.17)	87.30 (±0.19)	86.03 (±0.15)	**85.09 (±0.25)**	86.20 (±0.24)	87.70 (±0.26)
40	**87.81 (±0.27)**	86.42 (±0.24)	83.58 (±0.13)	**87.42 (±0.14)**	**86.06 (±0.17)**	85.06 (±0.23)	**86.27 (±0.16)**	**87.80 (±0.20)**
50	87.51 (±0.28)	**86.49 (±0.36)**	83.67 (±0.16)	87.21 (±0.19)	85.98 (±0.17)	**85.09 (±0.23)**	86.16 (±0.24)	87.76 (±0.14)

Table 6.6 displays the results of the ensemble size test with a fixed sample size of 300. The trend is different for each kernel, the best gain is usually gotten by an ensemble size of 40. The best overall gain is achieved by the radial ensemble.

6.1.3 Optdig

Table 6.7: Optdig data set, gain for single svm trained on complete training data with training time in seconds

SVM Type	Gain	Model Training Time
radial	97.94	3.47
linear	96.61	1.87
polynomial	97.50	2.33

Table 6.7 displays the gains for single SVMs trained with the complete data, comparing different kernels. The radial kernel performs best, but has the slowest training time. The linear kernel is the fastest, but has the worst gain.

Table 6.8: Result table for the optdig data set comparing different sample sizes with a fixed ensemble size of 10, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Sample Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
300	96.23 (±0.21)	96.07 (±0.21)	95.47 (±0.33)	96.21 (±0.15)	95.95 (±0.23)	95.90 (±0.23)	96.10 (±0.16)	96.39 (±0.13)
600	96.93 (±0.28)	96.93 (±0.17)	96.12 (±0.17)	97.07 (±0.15)	96.68 (±0.15)	96.66 (±0.15)	96.78 (±0.10)	97.11 (±0.12)
900	97.39 (±0.25)	97.33 (±0.13)	96.32 (±0.13)	97.40 (±0.12)	96.96 (±0.14)	96.94 (±0.06)	97.09 (±0.08)	97.44 (±0.08)
1200	97.65 (±0.15)	97.51 (±0.12)	96.52 (±0.11)	97.61 (±0.10)	97.24 (±0.17)	97.08 (±0.11)	97.40 (±0.14)	97.74 (±0.12)
1500	97.66 (±0.16)	97.47 (±0.16)	96.67 (±0.24)	97.62 (±0.08)	97.33 (±0.23)	97.21 (±0.11)	97.42 (±0.13)	97.82 (±0.07)
1800	97.85 (±0.11)	**97.66 (±0.24)**	96.71 (±0.18)	97.79 (±0.07)	97.41 (±0.08)	97.32 (±0.13)	97.53 (±0.09)	97.88 (±0.04)
2100	97.95 (±0.16)	97.63 (±0.05)	**96.87 (±0.16)**	97.84 (±0.12)	97.56 (±0.13)	97.31 (±0.10)	97.57 (±0.08)	97.94 (±0.11)
2400	97.95 (±0.16)	97.65 (±0.10)	96.80 (±0.19)	**97.86 (±0.10)**	97.55 (±0.07)	97.30 (±0.10)	97.57 (±0.09)	98.03 (±0.08)
2700	**98.01 (±0.06)**	97.64 (±0.13)	96.84 (±0.11)	97.84 (±0.12)	**97.62 (±0.14)**	**97.38 (±0.11)**	**97.59 (±0.07)**	**98.05 (±0.07)**

Table 6.8 shows the results for the sample size test of the optdig data set with a set ensemble size of 10. The gain is rising with the sample size. The best result is achieved by the *Radialx3* ensemble with a gain of 98.05, which even outperforms the best result of the single SVMs.

Table 6.9: Result table for the optdig data set comparing different ensemble with a fixed sample size of 500, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Ensemble Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
10	96.33 (±0.27)	96.15 (±0.19)	95.45 (±0.27)	96.27 (±0.31)	95.99 (±0.16)	95.88 (±0.24)	96.09 (±0.14)	96.32 (±0.17)
20	96.30 (±0.13)	96.10 (±0.14)	95.61 (±0.20)	96.25 (±0.10)	**96.15 (±0.09)**	**96.04 (±0.17)**	96.12 (±0.11)	96.39 (±0.11)
30	**96.43 (±0.12)**	**96.25 (±0.17)**	95.54 (±0.09)	96.31 (±0.15)	96.04 (±0.13)	95.96 (±0.13)	**96.17 (±0.18)**	**96.44 (±0.08)**
40	96.36 (±0.15)	96.22 (±0.15)	**95.61 (±0.15)**	**96.33 (±0.06)**	96.05 (±0.06)	95.98 (±0.11)	96.09 (±0.09)	96.41 (±0.09)
50	96.36 (±0.16)	96.19 (±0.10)	95.61 (±0.22)	**96.33 (±0.16)**	96.01 (±0.17)	96.03 (±0.12)	96.15 (±0.08)	96.38 (±0.10)

The results of the esemble size tests of the optdig data set are shown in Table 6.9. The *Radialx3* ensemble performs best with an gain of 96.44 for an ensemble size of 30.

6.1.4 Adult

Table 6.10: Adult data set, gain for single svm trained on complete training data with training time in seconds

SVM Type	Gain	Model Training Time
radial	85.10	320.87
linear	84.46	639.84
polynomial	84.61	149.97

Table 6.10 shows the performance of single SVMs with different kernels trained on the complete training data of the Adult data set. The radial kernel performs best, while the polynomial is double as fast as the radial and four times faster than the linear kernel.

Table 6.11: Result table for the Ault data set comparing different sample sizes with a fixed ensemble size of 10, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Sample Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
500	84.79 (±0.18)	82.86 (±0.46)	84.25 (±0.18)	84.09 (±0.15)	84.71 (±0.07)	83.95 (±0.29)	84.47 (±0.12)	84.86 (±0.11)
1000	84.95 (±0.12)	82.90 (±0.23)	84.57 (±0.15)	84.05 (±0.14)	84.81 (±0.08)	84.04 (±0.10)	84.44 (±0.09)	84.93 (±0.07)
2000	84.93 (±0.14)	83.42 (±0.13)	84.63 (±0.08)	84.37 (±0.07)	**84.87 (±0.11)**	84.31 (±0.05)	84.60 (±0.06)	**84.95 (±0.08)**
4000	**84.99 (±0.15)**	**83.84 (±0.07)**	**84.63 (±0.11)**	**84.63 (±0.07)**	84.83 (±0.09)	**84.42 (±0.07)**	**84.68 (±0.04)**	84.94 (±0.06)

Table 6.11 displays the results of the sample size test with the adult data set and a fixed ensemble size of 10. For most kernels and combinations the gain is the higher the higher the sample size gets. The best overall result is obtained by the *Radial* ensemble with a gain of 84.99, which is close to the best single SVM.

Table 6.12: Result table for the Adult data set comparing different ensemble with a fixed sample size of 500, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Ensemble Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
10	84.79 (±0.14)	84.53 (±0.14)	84.25 (±0.22)	84.69 (±0.09)	84.56 (±0.09)	84.50 (±0.12)	84.61 (±0.08)	84.80 (±0.10)
20	84.76 (±0.11)	84.70 (±0.09)	84.25 (±0.17)	84.78 (±0.07)	84.58 (±0.07)	84.57 (±0.10)	84.67 (±0.07)	84.83 (±0.07)
30	**84.87 (±0.12)**	**84.72 (±0.12)**	84.25 (±0.06)	**84.81 (±0.06)**	84.62 (±0.08)	**84.57 (±0.06)**	**84.68 (±0.04)**	**84.86 (±0.03)**
40	84.79 (±0.08)	82.99 (±0.19)	84.25 (±0.08)	84.07 (±0.11)	84.71 (±0.06)	83.96 (±0.18)	84.50 (±0.09)	84.84 (±0.07)
50	84.84 (±0.07)	82.94 (±0.13)	**84.29 (±0.10)**	84.03 (±0.10)	**84.72 (±0.04)**	83.93 (±0.08)	84.50 (±0.05)	84.86 (±0.05)

The results of the ensemble size test with a fixed sample size of 500 and the adult data set are displayed in Table 6.12.The *Radial* ensemble with an ensemble size of 30 performs best.

6.1.5 Acoustic

Table 6.13: Acoustic data set, gain for single svm trained on complete training data with training time in seconds

SVM Type	Gain	Model Training Time
radial	79.59	14768.85
linear	–	training failed
polynomial	–	training failed

The results achieved by single SVMs trained on the complete training data of the acoustic set are shown in Table 6.13. The linear and polynomial kernel failed to complete in 12 hours of computing, so the test was aborted. The training of the radial SVM took more than 4h.

Table 6.14: Result table for the Acoustic data set comparing different sample sizes with a fixed ensemble size of 10, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Sample Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
500	75.15 (±0.46)	70.80 (±1.24)	56.91 (±0.95)	74.60 (±0.51)	72.75 (±0.80)	65.83 (±1.50)	72.69 (±0.66)	75.54 (±0.32)
1000	76.59 (±0.22)	71.97 (±0.82)	66.26 (±0.56)	76.09 (±0.24)	75.21 (±0.33)	70.65 (±0.57)	75.08 (±0.27)	76.91 (±0.17)
2000	77.66 (±0.09)	72.82 (±0.74)	69.59 (±0.29)	77.26 (±0.18)	76.39 (±0.14)	73.16 (±0.23)	76.58 (±0.17)	78.01 (±0.05)
4000	**78.58 (±0.10)**	**73.88 (±0.40)**	**70.06 (±0.19)**	**78.16 (±0.16)**	**77.14 (±0.10)**	**74.24 (±0.16)**	**77.57 (±0.11)**	**78.81 (±0.11)**

The results of the sample size test for the acoustic data set with an fixed ensemble size of 10 are shown in 6.14. With rising sample size the gain also rises. The best overall result is achieved by the *Radialx3* ensemble with a gain of 78.81.

Table 6.15: Result table for the Acoustic data set comparing different ensemble with a fixed sample size of 500, the best gain for each kernel is in bold letters, the best overall gain is underlined, the standard deviation in in brackets

Ensemble.Size	radial	polynomial	linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
10	75.10 (±0.43)	70.70 (±0.71)	56.93 (±1.06)	74.77 (±0.34)	73.02 (±0.59)	65.82 (±1.12)	72.90 (±0.49)	75.56 (±0.21)
20	75.60 (±0.11)	72.19 (±0.95)	**57.39 (±0.84)**	75.20 (±0.35)	73.22 (±0.51)	66.96 (±1.03)	73.28 (±0.39)	75.58 (±0.08)
30	75.45 (±0.30)	72.10 (±0.51)	57.02 (±0.58)	75.13 (±0.19)	73.08 (±0.32)	66.55 (±0.45)	73.16 (±0.17)	75.64 (±0.16)
40	75.54 (±0.17)	**72.43 (±0.38)**	56.80 (±0.79)	75.22 (±0.10)	73.18 (±0.29)	**66.59 (±0.74)**	73.22 (±0.22)	**75.65 (±0.11)**
50	**75.61 (±0.07)**	72.37 (±0.52)	56.97 (±0.56)	**75.32 (±0.13)**	**73.26 (±0.23)**	66.48 (±0.49)	**73.31 (±0.18)**	75.59 (±0.13)

Table 6.15 displays the results of the ensemble size test with a fixed sample size of 500. The best ensemble size is different for each kernel. The best overall gain is achieved by the *Radialx3* ensemble with a gain of 76.65.

6.1.6 Acoustic Binary

Table 6.16: Acoustic Binary data set, gain for single svm trained on complete training data with training time in seconds

SVM Type	Gain	Model Training Time
radial	90.70	12062.73
linear	–	training failed
polynomial	–	training failed

Table 6.16 displays the performance of the single SVMs trained with the complete training data of the Acoustic Binary data set. The linear and polynomial SVM training was aborted after a time of 12h with no result. The training for the single radial SVM took more than 3h.

Table 6.17: Result table for the Acoustic Binary data set comparing different sample sizes with a fixed ensemble size of 10 per Kernel, the best gain for each kernel is in bold letters, the best overall gain is underlined

Sample Size	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
500	88.55 (±0.11)	88.52 (±0.16)	87.57 (±0.36)	88.97 (±0.12)	88.72 (±0.12)	88.99 (±0.17)	89.07 (±0.08)	88.69 (±0.09)
1000	89.13 (±0.15)	89.04 (±0.17)	**87.78 (±0.18)**	89.53 (±0.12)	89.06 (±0.09)	89.33 (±0.08)	89.48 (±0.08)	89.29 (±0.07)
2000	89.71 (±0.13)	89.45 (±0.16)	87.74 (±0.10)	90.00 (±0.14)	89.41 (±0.07)	89.46 (±0.10)	89.83 (±0.09)	89.84 (±0.08)
4000	**90.10 (±0.13)**	**89.85 (±0.10)**	87.65 (±0.08)	**90.28 (±0.11)**	**89.67 (±0.12)**	**89.54 (±0.06)**	**90.05 (±0.05)**	**90.26 (±0.08)**

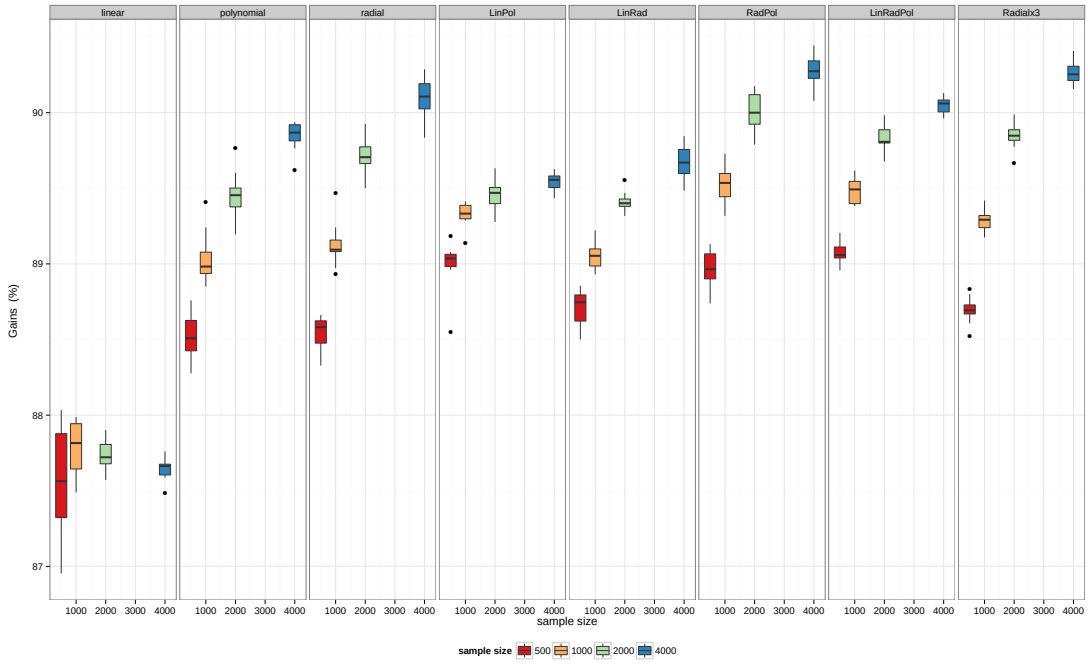


Fig. 6.2: Acoustic Binary data set boxplot result plot of the sample size test, sample size vs gain

Table 6.17 and Figure 6.2 shows the results for the Acoustic Binary data set with different sample sizes and a set ensemble size of 10 per kernel. It is visible that the gain improves with greater sample sizes for all kernel and their respective combinations except the linear kernel. The linear kernel also performs worst on this data set. The best overall result is reached by the combination of the radial with the polynomial kernel.

Table 6.18: Result table for the Acoustic Binary data set comparing different ensemble with a fixed sample size of 500, the best gain for each kernel is in bold letters, the best overall gain is underlined

Ensemble.Size	radial	polynomial	linear	RadPol	LinRad	LinPol	LinRadPol	Radialx3
10	88.56 (±0.14)	88.29 (±0.25)	87.58 (±0.42)	88.93 (±0.16)	88.67 (±0.09)	88.82 (±0.22)	89.02 (±0.14)	88.63 (±0.15)
20	88.60 (±0.12)	88.83 (±0.15)	87.91 (±0.13)	89.10 (±0.09)	88.72 (±0.13)	89.14 (±0.07)	89.12 (±0.08)	88.73 (±0.06)
30	88.62 (±0.10)	89.05 (±0.10)	88.01 (±0.21)	**89.17 (±0.11)**	88.79 (±0.12)	**89.27 (±0.10)**	**89.18 (±0.11)**	88.72 (±0.06)
40	88.70 (±0.04)	89.08 (±0.14)	88.03 (±0.16)	89.16 (±0.06)	88.78 (±0.07)	89.23 (±0.07)	**89.18 (±0.06)**	88.74 (±0.07)
50	**88.74 (±0.14)**	**89.12 (±0.09)**	**88.14 (±0.16)**	89.16 (±0.05)	**88.84 (±0.10)**	89.20 (±0.09)	**89.18 (±0.07)**	**88.76 (±0.05)**

Table 6.6 displays the results of the ensemble size test with a fixed sample size of 500. The best ensemble size is different for each kernel. The best overall gain is achieved by the *LinPol* ensemble with a gain of 89.27 and an ensemble size of 30.

6.1.7 Connect4

Figure 6.3 shows the result of the sample size test for the Connect4 data set. The results are hard to interpret, since the gain is in the range from 0 to 100 and for the linear kernel it is always 100 regardless of the sample size. The Connect4 data set is an artificial data set build up on all the moves of the game Connect4. A possible explanation of these results could be that there are some easy learning strategies for these set which lead to the given results. This issue has to be further investigated, but the focus of these tests was to see the performance of bagging and these data set is not appropriate for this matter.

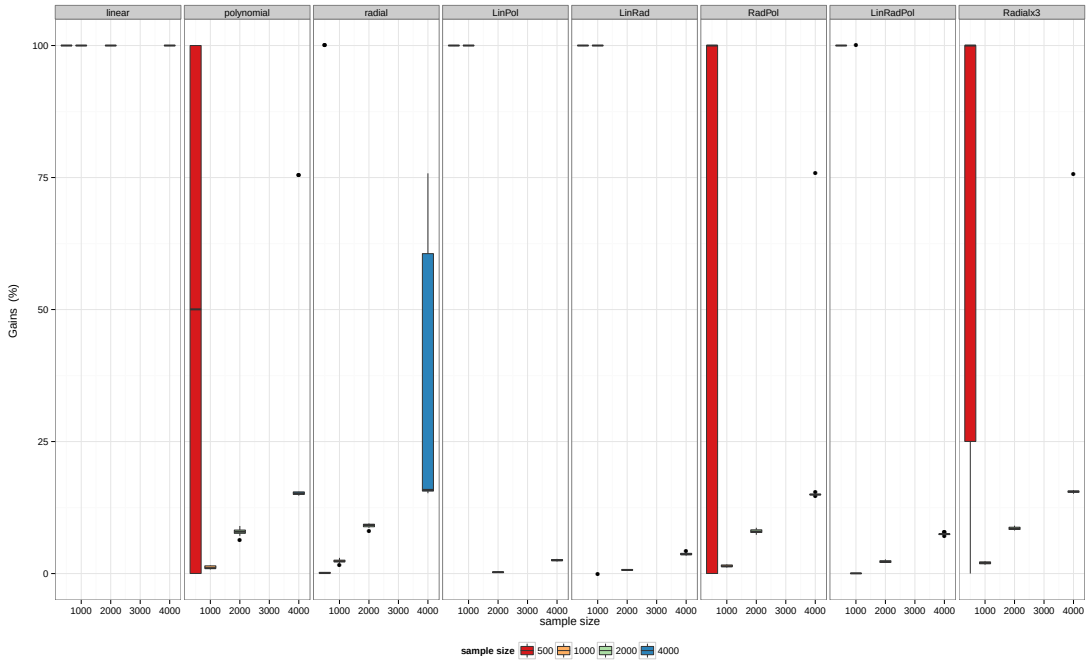


Fig. 6.3: Connect 4 data set boxplot result plot of the sample size test, sample size vs gain

6.1.8 Majority vs Probability Voting

Table 6.19: Majority vs Probability Result Table covering the small data sets and the Ensemble Size Test(EST) with an ensemble size of 50 and the Sample Size Test(SST) with an sample size of 2700

Test Type	Radial	Polynomial	Linear	RadPol	LinRad	LinPol	LinRadPol
Spam Probability SST	**93.99 (±0.14)**	**93.51 (±0.15)**	**93.79 (±0.14)**	**94.05 (±0.14)**	**94.30 (±0.09)**	**94.27 (±0.09)**	**94.60 (±0.13)**
Spam Majority SST	93.55 (±0.15)	93.14 (±0.12)	92.39 (±0.15)	93.38 (±0.23)	93.23 (±0.15)	93.06 (±0.19)	93.89 (±0.13)
Spam Probability EST	**93.09 (±0.11)**	**92.82 (±0.27)**	**92.59 (±0.29)**	**93.61 (±0.13)**	**93.68 (±0.12)**	**92.99 (±0.24)**	**93.56 (±0.09)**
Spam Majority EST	91.85 (±0.22)	92.52 (±0.29)	91.06 (±0.48)	93.00 (±0.13)	93.12 (±0.24)	92.29 (±0.20)	93.10 (±0.20)
Satlog Probability SST	**90.48 (±0.22)**	**90.09 (±0.19)**	85.09 (±0.15)	**90.26 (±0.11)**	**88.92 (±0.23)**	**88.49 (±0.19)**	89.44 (±0.09)
Satlog Majority SST	90.39 (±0.19)	90.03 (±0.20)	**85.14 (±0.14)**	90.14 (±0.16)	88.34 (±0.10)	87.94 (±0.17)	**89.61 (±0.17)**
Satlog Probability EST	87.51 (±0.28)	86.49 (±0.36)	**83.67 (±0.16)**	87.21 (±0.19)	**85.98 (±0.17)**	**85.09 (±0.23)**	**86.16 (±0.24)**
Satlog Majority EST	**87.67 (±0.21)**	**86.53 (±0.30)**	83.59 (±0.31)	**87.38 (±0.22)**	85.59 (±0.19)	84.84 (±0.18)	85.94 (±0.08)
Optdig Probability SST	**98.01 (±0.06)**	97.64 (±0.13)	**96.84 (±0.11)**	97.84 (±0.12)	97.62 (±0.14)	**97.38 (±0.11)**	97.59 (±0.07)
Optdig Majority SST	**98.01 (±0.15)**	**97.74 (±0.11)**	96.79 (±0.16)	**97.90 (±0.13)**	**97.64 (±0.11)**	97.35 (±0.12)	**97.66 (±0.09)**
Optdig Probability EST	**96.36 (±0.16)**	**96.19 (±0.10)**	**95.61 (±0.22)**	**96.33 (±0.16)**	**96.01 (±0.17)**	**96.03 (±0.12)**	**96.15 (±0.08)**
Optdig Majoriy EST	96.34 (±0.19)	96.17 (±0.15)	95.50 (±0.16)	96.29 (±0.12)	**96.01 (±0.12)**	95.97 (±0.18)	96.10 (±0.11)

Table 6.19 shows the difference between the two aggregation methods introduced in 3.2.1. The tests covers the small data sets and the results of the Sample Size Test with 2700 cases and the Results of the Ensemble Size Tests with 50 SVMs per type Ensemble. These tests have been repeated using majority aggregation. For the Spam data set, which is a binary case, the probability voting performs better in every case and has a significant positive effect on the gain. For the Satlog multiclass problem the trend is not clear, sometimes the probability voting obtains a better gain and in some cases the majority voting performs better. For the Optdig data set, both aggregation methods have only minimal difference.

6.2 Results for AdaBoost

The following section exposes a series of results which will help understand and observe how the implemented SVM-AdaBoost algorithm performs under several circumstances. The first experiments were made considering a 100% train set to build the ensemble. Afterwards a performance comparison using different boosting factors $bo.size$ will show how the ensembles behave. At last, two general experiments will show how the SVM-AdaBoost selects the training set for each next classifier in the algorithm and how many support vectors are used on each classifier. The proposed experiments were conducted using the parameters of Table 6.20 for each task.

Table 6.20: *Parameters used for the svm functions inside AdaBoost. The gamma (γ) parameters were randomly selected before every iteration between min and max using the 10% and 90% quantiles of the estimated best gamma as normal distribution. Column 1SVM γ stands for the parameter used for one SVM.*

Name	Rad. Cost (C)	1SVM γ	Min γ	Max γ	Poly. Cost (C)	Poly. Deg.	Lin. Cost (C)
Spam	7.56	$0.545e^{-2}$	$0.486e^{-2}$	$5.33e^{-2}$	3.39	3	4.93
Satellite	8.17	$4.73e^{-2}$	$0.61e^{-2}$	$11.32e^{-2}$	3.86	3	6.14
OptDig	8	$0.04e^{-2}$	$0.029e^{-2}$	$0.04e^{-2}$	5	3	6
Adult	15.7	$13.74e^{-2}$	$2.82e^{-2}$	$13.78e^{-2}$	11.9	3	18.7
Acoustic	25	$2.98e^{-2}$	$0.475e^{-2}$	$6.13e^{-2}$	10	3	8

6.2.1 Results using full train size

The following results show the accuracies obtained from the experiments conducted using the full train size for every data set. For general demonstration, only tasks Optical Digit Recognition and Spam were selected to show how the prediction accuracy behaves while the ensemble size increases. Figure 6.4 shows the behavior for task Optical Digit Recognition and the respective mean times are shown in Table 6.21. For Spam task, Figure 6.5 and Table 6.22 show the respective results.

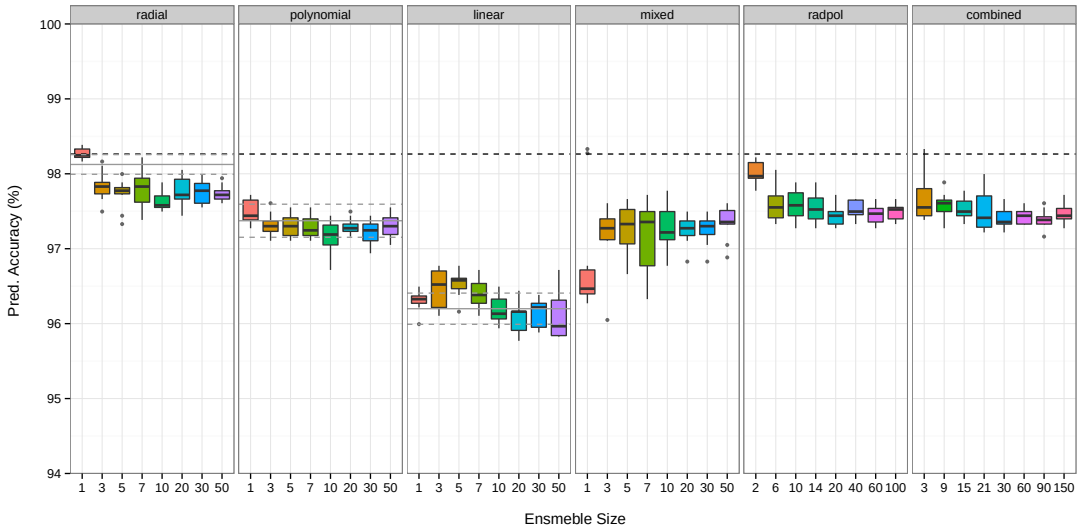


Fig. 6.4: Accuracy on task **Optical Digit Recognition** *with SVM-AdaBoost ensemble. Each sub-classifier uses all training sample. On the x axis the ensemble size and on the y axis the prediction accuracy. The group of grey lines show the best mean (± sd) of the single SVM as comparison for each kernel type used. The black dashed line indicates the best mean accuracy obtained from the ensemble formation.*

Table 6.21: Time taken to train (mean sec ±sd) task **Optical Digit Recognition** *from 10 experimental runs on every ensemble type and 50 svm versus one single SVM (first row). Marked in bold, the best times for each experiment type.*

Name	Radial	Polynomial	Linear	Mixed	RadPol	Combined
SVM	3.42 (±0.05)	1.97 (±0.03)	**1.62 (±0.04)**	2.55 (±0.83)	5.40 (±0.06)	7.01 (±0.07)
Ensemble	92.31 (±15.19)	62.51 (±6.26)	**56.32 (±7.25)**	86.37 (±16.20)	154.82 (±17.49)	211.14 (±19.14)

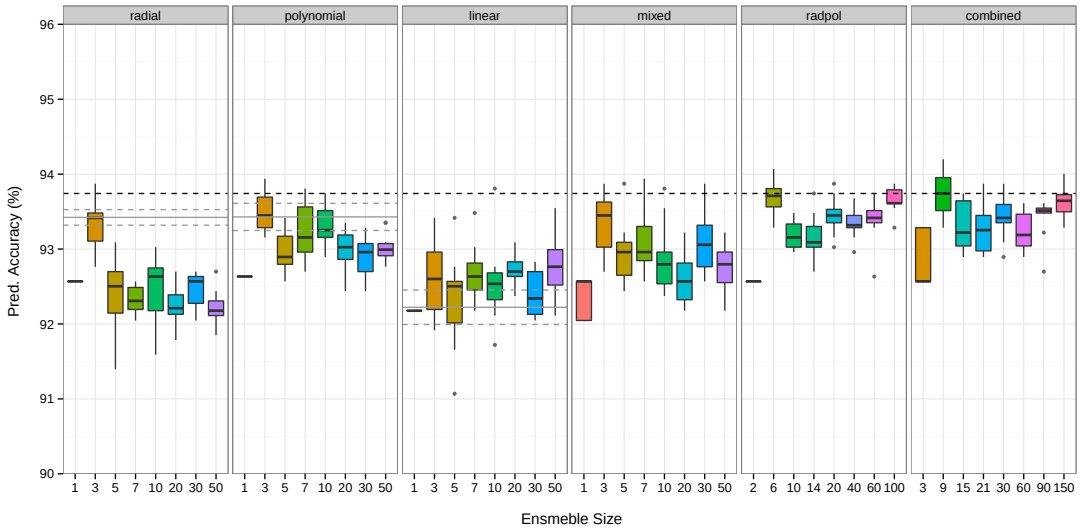


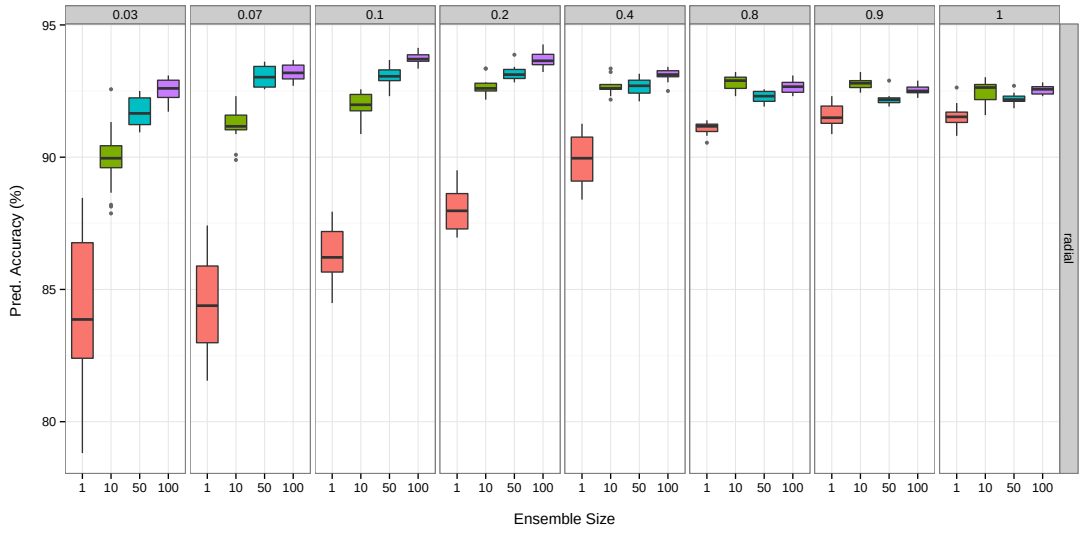
Fig. 6.5: Accuracy on task **Spam** *with SVM-AdaBoost ensemble. Each sub-classifier uses all training sample. On the x axis the ensemble size and on the y axis the prediction accuracy. The group of grey lines show the best mean (± sd) of the single SVM as comparison for each kernel type used. The black dashed line indicates the best mean accuracy obtained from the ensemble formation.*

Table 6.22: Time taken to train (mean sec ±sd) task **Spam** *from 10 experimental runs on every ensemble type and 50 svm versus one single SVM (first row). Marked in bold, the best times for each experiment type.*

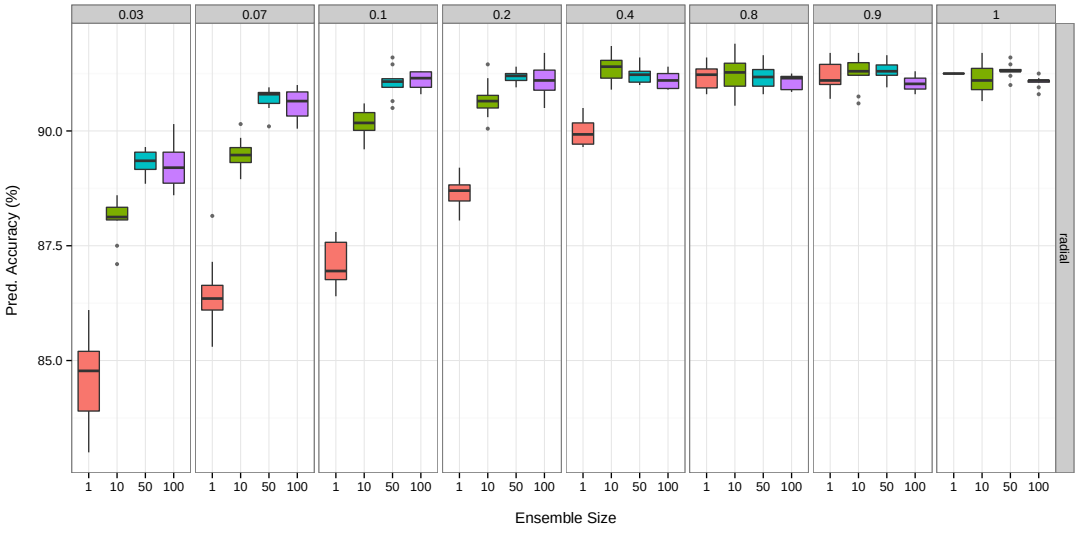
Name	Radial	Polynomial	Linear	Mixed	RadPol	Combined
SVM	1.70 (±0.06)	**1.16 (±0.06)**	2.68 (±0.30)	1.40 (±0.24)	2.86 (±0.11)	5.54 (±0.38)
Ensemble	204.52 (±72.03)	**136.18 (±49.88)**	510.86 (±165.26)	166.62 (±62.82)	340.70 (±121.88)	851.55 (±286.44)

6.2.2 Results using factor bo.size

Figures 6.4 and 6.5 showed that AdaBoost has no improvement if the ensemble size is increased using the full training set. To begin with these experiments, Figure 6.6 shows that the performance on the tasks Spam and Satellite starts to decay between factors $bo.size = 0.4$ and $bo.size = 0.8$, and that if a factor of one is used, there is no improvement from 1 to a 100 SVM ensemble. It is for this reason that the next results are intended to show that, by avoiding this performance degradation with the introduction of the factor $bo.size$, sometimes a better or same accuracy can be obtained from this on the different tasks with even less time. The following plots show the accuracies obtained from the experiments conducted using the full train size but resampling inside AdaBoost with the factor $bo.size$ shown in Table 6.23 used for each data set.



(a) Performance degradation on task Spam.



(b) Performance degradation on task Satellite.

Fig. 6.6: *Performance degradation of SVM-AdaBoost on tasks* **Spam** *and* **Satellite** *by comparing ensemble size against different* bo.size *factors from 0.03 to 1 using Radial kernel. On the x axis the ensemble size (1,10,50,100) and on the y axis the prediction accuracy.*

Table 6.23: bo.size parameter used for each task, where the main goal is to obtain a Sampled Size between 290 and 310.

Name	bo.size	Train Size	Sampled Size
Spam	0.1	3067	306
Satellite	0.067	4435	297
OptDig	0.078	3823	298
Adult	0.01	30162	301
Acoustic	0.003806	78823	300

Optical Digit Recognition

Figure 6.7 shows the accuracies of the task Optical Digit Recognition in respect to the ensemble size of AdaBoost, whereas Table 6.24 shows the mean times, in seconds, taken to build 1 single SVM against an ensemble size of 50 svm with AdaBoost using the different kernel types and the combination of them. For the experiments "RadPol" and "All-Combined", the total time taken to build the ensemble is the sum of the times of their respective kernel experiments is considered.

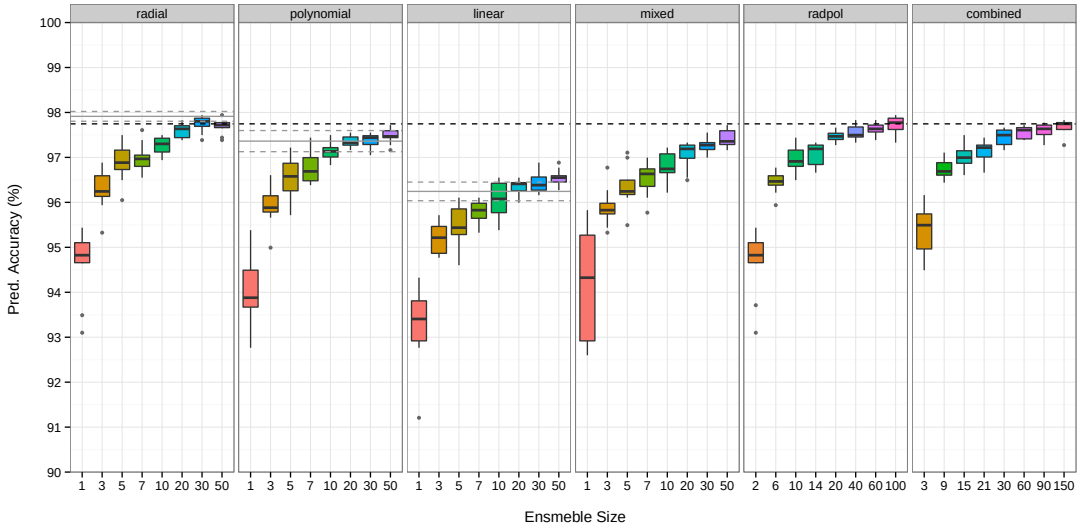


Fig. 6.7: Accuracy on task **Optical Digit Recognition** *with SVM-AdaBoost ensemble. Each sub-classifier uses a training sample of around 300 records. On the x axis the ensemble size and on the y axis the prediction accuracy. The group of grey lines show the best mean (± sd) of the single SVM as comparison for each kernel type used. The black dashed line indicates the best mean accuracy obtained from the ensemble formation.*

Table 6.24: Time taken to train (mean sec ±sd) task **Optical Digit Recognition** *from 10 experimental runs on every ensemble type and 50 SVM versus one single SVM (first row). For the ensemble, each sub-classifier uses the sampled size from table 6.23 and for one SVM all the train size is used. Marked in bold, the best times for each experiment type.*

Name	Radial	Polynomial	Linear	Mixed	RadPol	Combined
SVM	3.13 (±0.09)	1.98 (±0.05)	**1.63 (±0.04)**	2.24 (±0.64)	5.12 (±0.12)	6.75 (±0.13)
Ensemble	13.39 (±0.23)	**11.60 (±0.11)**	12.26 (±0.17)	13.33 (±0.27)	24.99 (±0.27)	37.24 (±0.40)

For the Optical Digit Recognition task, Figure 6.7, shows how the performance increases with SVM-AdaBoost while the ensemble size increases. Notice that the accuracy seems to have room to improve much more, and that the 50 SVM-AdaBoost ensemble gets closer to the best obtained mean accuracy from one SVM of 97.91% with 97.69%. The mean time taken, in seconds, to train the fastest 50 SVM-AdaBoost ensemble was reduced very much (Table 6.24 against 6.21), if the boosting factor is used, by a relation of 35 times to 7 times, both against one SVM.

Satellite

Figure 6.8 shows the accuracies obtained using AdaBoost with different ensemble sizes on task Satellite and Table 6.25 shows the respective mean times, in seconds, taken to build an ensemble of 50 SVM against 1 SVM. For the experiments "RadPol" and "All-Combined", the total time to train the ensemble is the sum of the times of their respective kernel experiments is considered.

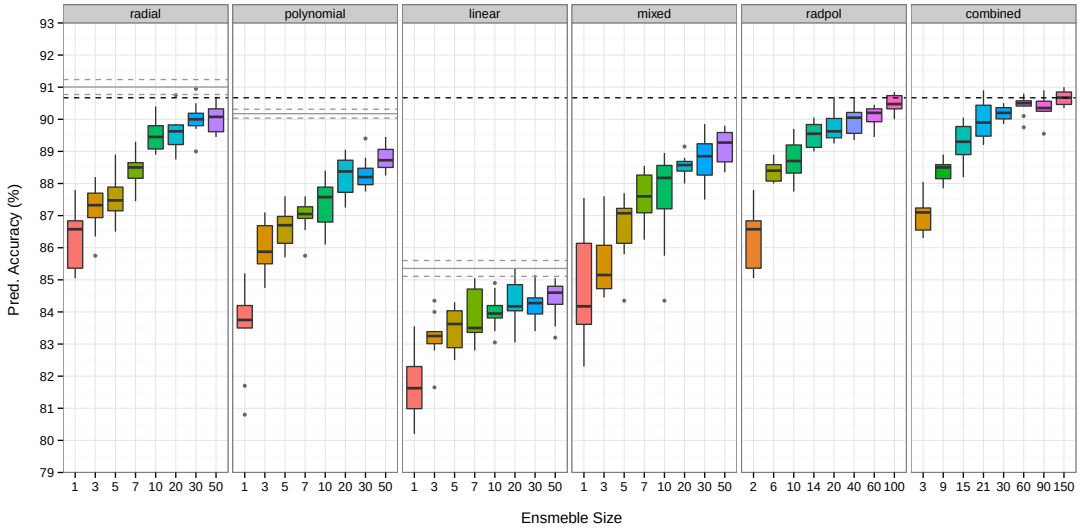


Fig. 6.8: *Accuracy on task* **Satellite** *with SVM-AdaBoost ensemble. Each sub-classifier uses a training sample of around 300 records. On the* x *axis the ensemble size and on the* y *axis the prediction accuracy. The group of grey lines show the best mean (*$\pm$ *sd) of the single SVM as comparison for each kernel type used. The black dashed line indicates the best mean accuracy obtained from the ensemble formation.*

Table 6.25: *Time taken to train (mean sec* $\pm sd$*) task* **Satellite** *from 10 experimental runs on every ensemble type and 50 SVM versus one single SVM (first row). For the ensemble, each sub-classifier uses the sampled size from table 6.23 and for one SVM all the train size is used. Marked in bold, the best times for each experiment type.*

Name	Radial	Polynomial	Linear	Mixed	RadPol	Combined
SVM	2.24 ($\pm$0.03)	**1.32 ($\pm$0.02)**	1.68 ($\pm$0.05)	1.98 ($\pm$0.37)	3.56 ($\pm$0.04)	5.23 ($\pm$0.08)
Ensemble	10.72 ($\pm$1.05)	**7.11 ($\pm$0.76)**	13.43 ($\pm$1.69)	9.34 ($\pm$1.06)	17.83 ($\pm$1.78)	31.26 ($\pm$3.38)

The Satellite results from Figure 6.8, show that the All-Combined performed better (90.67%) than the single kernel ensembles, giving closer results against one SVM (91.00%). Notice that for the single kernel ensembles there is room to improve if more SVMs are used in SVM-AdaBoost, but the All-Combined shows some saturation beyond the combined 30 SVM-AdaBoost ensembles, although a tendency to slowly grow is shown. The time relation to train a 50 SVM-AdaBoost ensemble against one SVM is 5 times more.

Spam

Figure 6.9 shows the accuracies obtained using AdaBoost with different ensemble sizes on task Spam and Table 6.26 shows the respective mean times taken to build an ensemble of 50 SVM against 1 SVM. For the experiments "RadPol" and "All-Combined", the total time to train the ensemble is the sum of the times of their respective kernel experiments is considered.

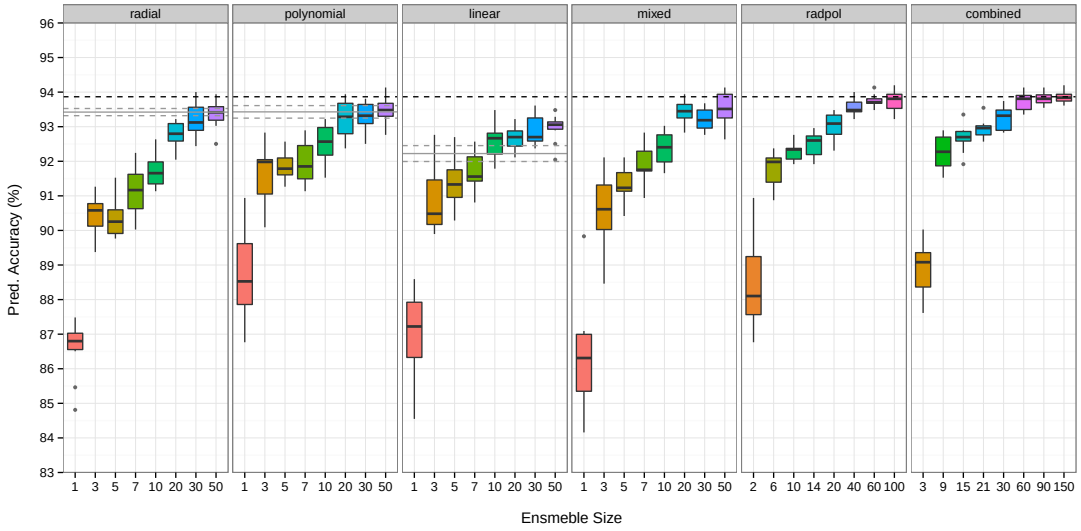


Fig. 6.9: *Accuracy on task* **Spam** *with SVM-AdaBoost ensemble. Each sub-classifier uses a training sample of around 300 records. On the x axis the ensemble size and on the y axis the prediction accuracy. The group of grey lines show the best mean ($\pm$ sd) of the single SVM as comparison for each kernel type used. The black dashed line indicates the best mean accuracy obtained from the ensemble formation.*

Table 6.26: *Time taken to train (mean sec $\pm$sd) task* **Spam** *from 10 experimental runs on every ensemble type and 50 SVM versus one single SVM (first row). For the ensemble, each sub-classifier uses the sampled size from table 6.23 and for one SVM all the train size is used. Marked in bold, the best times for each experiment type.*

Name	Radial	Polynomial	Linear	Mixed	RadPol	Combined
SVM	1.70 ($\pm$0.06)	**1.16 ($\pm$0.06)**	2.68 ($\pm$0.30)	1.40 ($\pm$0.24)	2.86 ($\pm$0.11)	5.54 ($\pm$0.38)
Ensemble	11.45 ($\pm$0.45)	**7.94 ($\pm$0.27)**	16.00 ($\pm$1.05)	9.70 ($\pm$0.49)	19.39 ($\pm$0.65)	35.40 ($\pm$1.41)

The Spam results from Figure 6.9 proved that the All-Combined SVM-AdaBoost ensemble performed better than one SVM, presenting an accuracy of 93.87% against 93.43%. Although the single kernel ensembles presented as well better results than one SVM, it is noticed that the All-Combined presents a saturation beyond the SVM-AdaBoost combination of 30 SVM ensemble. On the other hand, it was shown from Table 6.22 that the time taken to train an ensemble using 100% of the train data or $bo.size = 1$, was 120 times slower than a single SVM, and using a $bo.size = 0.1$ showed that the fastest time performed only 6 times slower than a single SVM, which is a remarkable difference to perform better in accuracy.

Adult

Figure 6.10 shows the accuracies obtained using AdaBoost with different ensemble sizes on task Adult and Table 6.27 shows the respective mean times taken to build an ensemble of 50 SVM against 1 SVM. For the experiments "RadPol" and "All-Combined", the total time to train the ensemble is the sum of the times of their respective kernel experiments is considered.

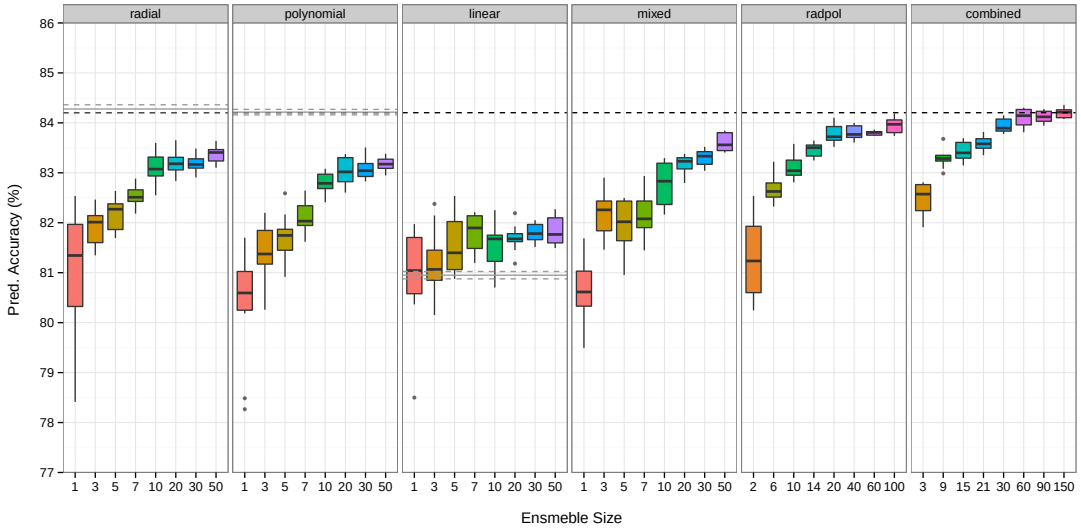


Fig. 6.10: *Accuracy on task* **Adult** *with SVM-AdaBoost ensemble. Each sub-classifier uses a training sample of around 300 records. On the* x *axis the ensemble size and on the* y *axis the prediction accuracy. The group of grey lines show the best mean (*$\pm$ *sd) of the single SVM as comparison for each kernel type used. The black dashed line indicates the best mean accuracy obtained from the ensemble formation.*

Table 6.27: *Time taken to train (mean sec* $\pm sd$*) task* **Adult** *from 10 experimental runs on every ensemble type and 50 SVM versus one single SVM (first row). For the ensemble, each sub-classifier uses the sampled size from table 6.23 and for one SVM all the train size is used. Marked in bold, the best times for each experiment type.*

Name	Radial	Polynomial	Linear	Mixed	RadPol	Combined
SVM	181.71 ($\pm$1.36)	514.85 ($\pm$31.56)	276.71 ($\pm$44.69)	**135.00 ($\pm$52.26)**	696.56 ($\pm$31.02)	973.27 ($\pm$69.73)
Ensemble	35.40 ($\pm$0.35)	**15.30 ($\pm$0.10)**	16.18 ($\pm$2.09)	25.09 ($\pm$1.76)	50.69 ($\pm$0.34)	66.87 ($\pm$2.16)

The Adult task presented in Figure 6.10, showed a mean best accuracy of 84.20% for SVM-AdaBoost ensemble against 84.40% for one SVM, where again the All-Combined showed better results than every single kernel and mixed ensemble. The remarkable difference with this task is the time taken to train the SVM-AdaBoost, where it outperforms 12 times faster than one SVM, and the accuracy outcome is very close to the best mean from one SVM.

Acoustic

Figure 6.11 shows the accuracies obtained using AdaBoost with different ensemble sizes on task Acoustic and Table 6.28 shows the respective mean times, in minutes, taken to build an ensemble of 200 SVM against 1 SVM. For the experiments "RadPol" and "All-Combined", the total time to train the ensemble is the sum of the times of their respective kernel experiments is considered. Since there is noticed a tendency on the accuracy to keep increasing while the ensemble size increases, it was decided, for demonstration purposes, to extend this experiment to 200 SVM.

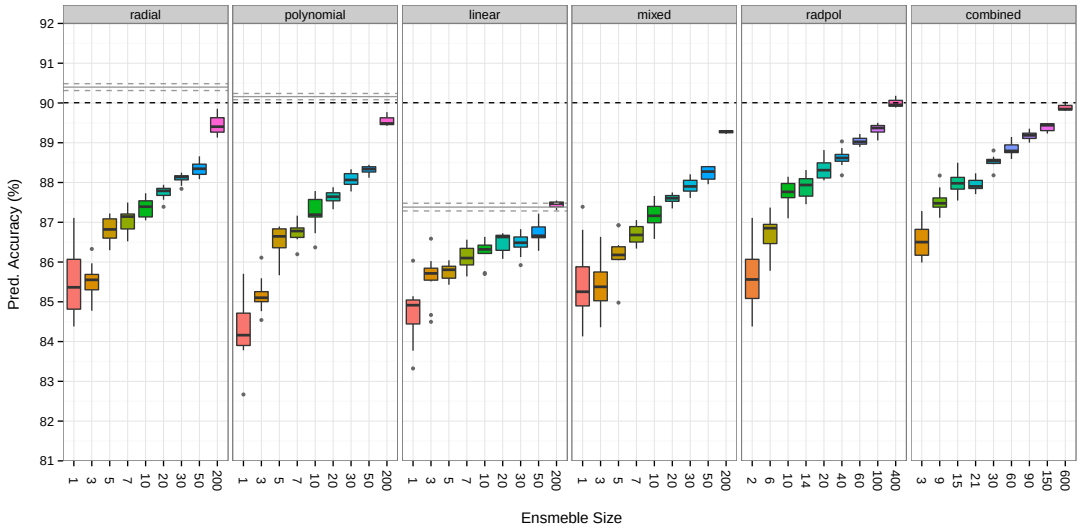


Fig. 6.11: *Accuracy on task* **Acoustic** *with SVM-AdaBoost ensemble. Each sub-classifier uses a training sample of around 300 records. On the* x *axis the ensemble size and on the* y *axis the prediction accuracy. The group of grey lines show the best mean (*$\pm$ *sd) of the single SVM as comparison for each kernel type used. The black dashed line indicates the best mean accuracy obtained from the ensemble formation.*

Table 6.28: *Time taken to train (mean min* $\pm$*sd) task* **Acoustic** *from 10 experimental runs on every ensemble type and 200 SVM versus one single SVM (first row). For the ensemble, each sub-classifier uses the sampled size from table 6.23 and for one SVM all the train size is used. Marked in bold, the best times for each experiment type.*

Name	Radial	Polynomial	Linear	Mixed	RadPol	Combined
SVM	50.97 ($\pm$6.11)	439.86 ($\pm$23.38)	**16.66 ($\pm$1.59)**	35.26 ($\pm$22.94)	490.83 ($\pm$29.49)	507.49 ($\pm$27.90)
Ensemble	12.01 ($\pm$0.80)	8.12 ($\pm$0.57)	**7.04 ($\pm$0.89)**	9.51 ($\pm$0.85)	20.13 ($\pm$1.33)	27.17 ($\pm$1.44)

The Acoustic task from Figure 6.11, being the biggest data set used in this research, gave with one SVM a better accuracy of 90.70% against 90.01% from the SVM-AdaBoost, obtained from the RadPol ensemble. Nevertheless, for this case, the All-Combined and RadPol ensembles do note show any saturation, giving room for the SVM-AdaBoost to keep improving by using bigger ensemble sizes, as shown with the 200 SVM-AdaBoost ensemble size. Here the remarkable time difference is noticed again by an outcome of 7 times faster training time than one SVM, where if the ensemble size is increased, a closer accuracy to the one SVM can be obtained.

6.2.3 General comparison between Full Train against bo.size experiments inside SVM-AdaBoost

Figure 6.12 shows the number of support vectors on every weak classifier used by an SVM-AdaBoost ensemble with 50 SVM, against the different proposed boosting factors. Notice that the bigger the *bo.size*, the more support vectors are used per weak classifier, where if the maximum possible *bo.size* factor is used to build the ensemble, almost the whole training set of the Spam task is used. With this results, it is observable that the SVM-AdaBoost ensemble using the whole training set is overfitted, showing no improvement on the prediction accuracy, as shown from Figure 6.5.

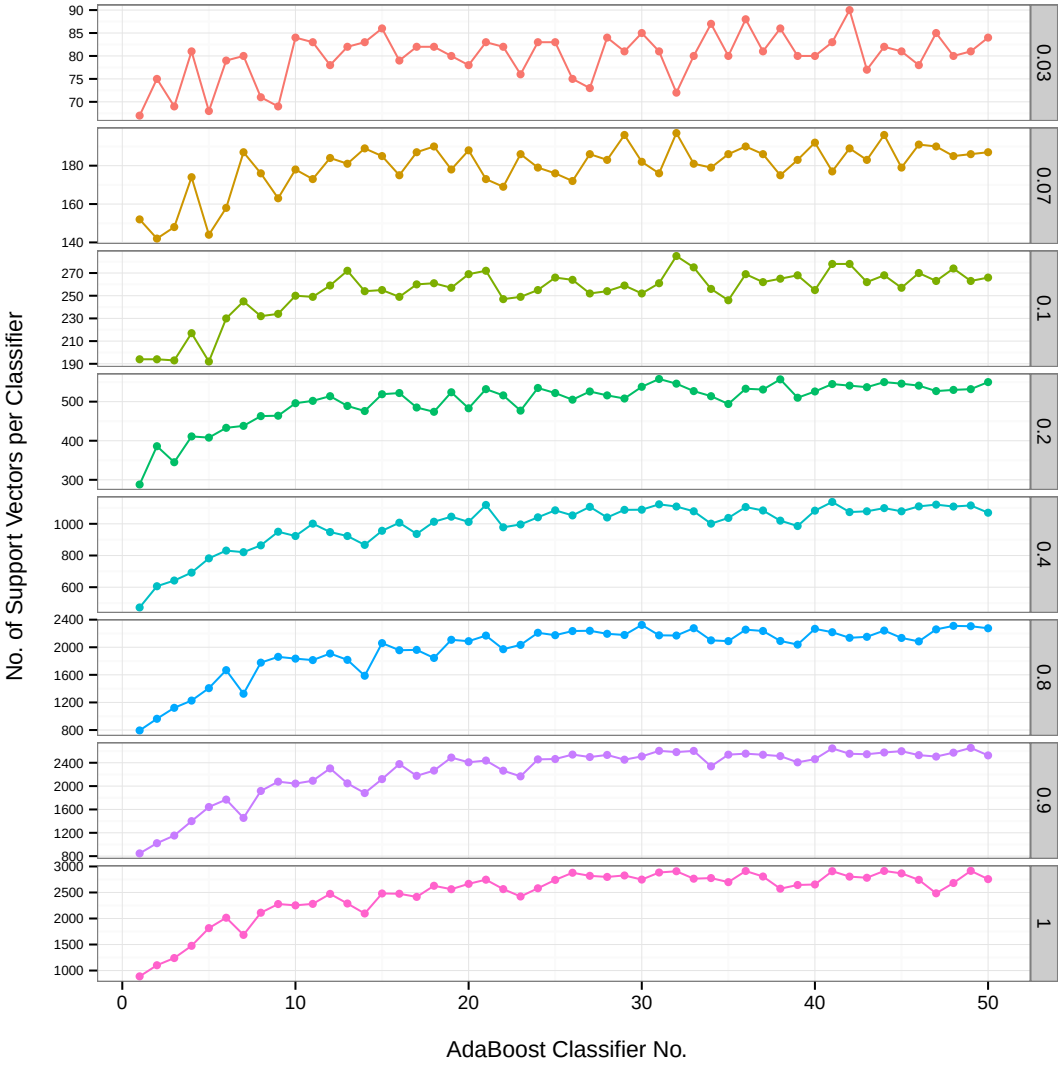
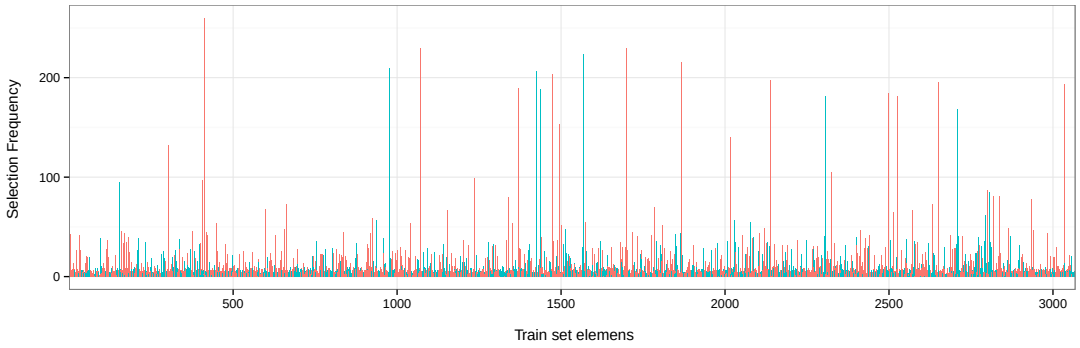


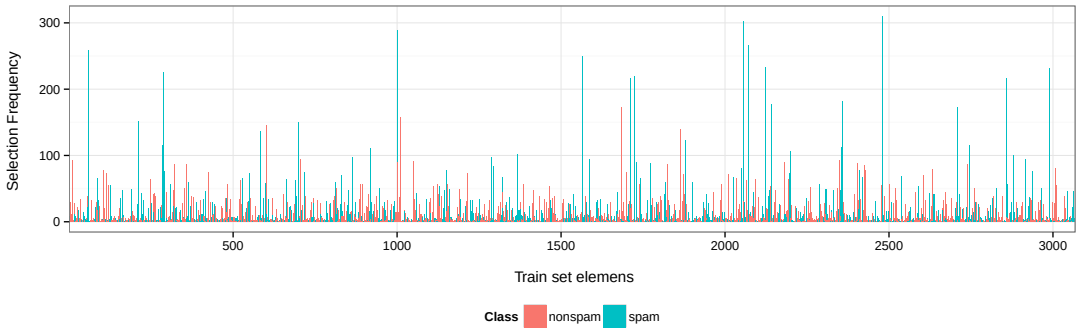
Fig. 6.12: Amount of support vectors (y axis) for each weak classifier in the SVM-AdaBoost ensemble (x axis) on task Spam against the boosting factor bo.size (boxes) using a 50 SVM ensemble.

Figures 6.13 and 6.14 show the frequency selection of every train sample of the Spam task with different *bo.size* factors. Both figures show the same results, but with different point of views. Figure 6.13 shows on the x axis the train set elements and on the y axis the number of times every element was selected to train every single weak classifier on an SVM-AdaBoost ensemble with 10 SVM and 100 SVM. Figure 6.14, on the other hand, shows on the y axis the amount of elements of the train set against the frequency of selection on the x axis. Notice that for a *bo.size* factor of 1, many elements were selected more than 100 times, and from Figure 6.14(a), it is shown that at least 100 elements were selected between 2 and 9 times and even some elements were chosen 230 times.

On the other hand, Figures 6.13(b) and 6.14(b) show that, for a $bo.size$ of 0.1 with an ensemble size of 100 SVM, the diversity of selection presented similar results as the experiment with ensemble size of 10 SVM and $bo.size$ of 1. This shows a big difference when a $bo.size$ factor of 1 against 0.1 is used for SVM-AdaBoost, where the most elements are selected only once for the second case.

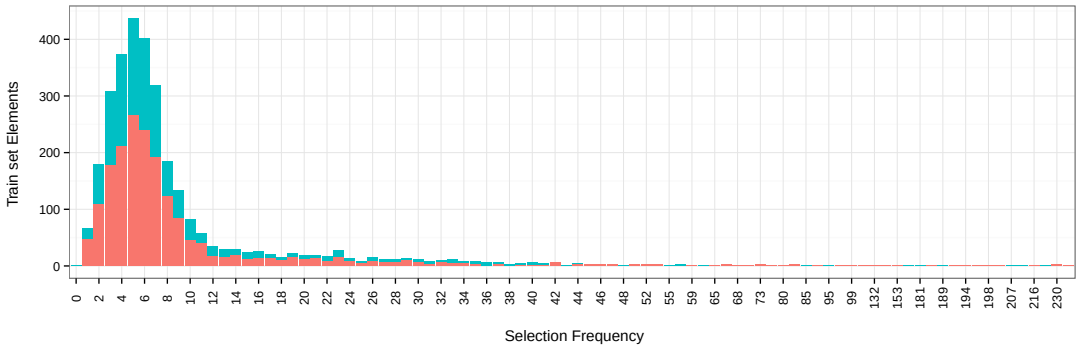


(a) Selection frequency for $bo.size = 1$ and 10 SVM.

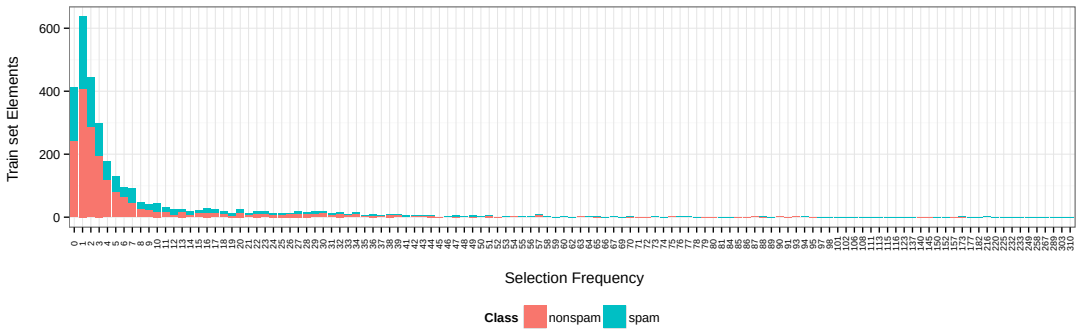


(b) Selection frequency for $bo.size = 0.1$ and 100 SVM.

Fig. 6.13: *Selection frequency (y axis) of train elements (x axis) by SVM-AdaBoost resample process before training the next weak classifier on Spam task using a 10 and 100 SVM ensemble respectively. Plot (a) shows the number of selected cases using a boosting factor of* 1 *with 10 SVM and (b) shows the cases using a factor of* 0.1 *with 100 SVM. Colors are only for showing purposes and do not play a role on the demonstration of the selection frequency of the experiment.*



(a) Selection frequency for $bo.size = 1$ and 10 SVM.



(b) Selection frequency for $bo.size = 0.1$ and 100 SVM.

Fig. 6.14: *Analogue point of view from Figure 6.13, presenting the selection frequency (x axis) of total number of train elements (y axis) by SVM-AdaBoost resample process before training the next weak classifier on Spam task using a 10 and 100 SVM ensemble respectively. Plot (a) shows the number of selected cases using a boosting factor of* 1 *with 10 SVM and (b) shows the cases using a factor of* 0.1 *with 100 SVM. Colors are only for showing purposes and do not play a role on the demonstration of the selection frequency of the experiment.*

7 Discussion

7.1 SVM Bagging

7.1.1 Early Investigations

In the beginning of this study and while developing the SVM bagging algorithm and performing the usual testing, it was discovered with the spam data set that combining different kernels in one bagging ensemble can have a significant positive effect on the overall accuracy. This matter, as it was not found to be researched in the viewed literature, has thus been made an additional investigation topic for this study, beside the main goal to fit the SVM algorithm to the needs of big data. The SVM bagging algorithm was introduced to reach the desired goals and this Section is to discuss in detail which of the initial ideas and presumptions made in Section 3.2.1 were right and visible in the numerous experiments discussed in Section 6.

7.1.2 Result Summary

The different experiments for the SVM bagging algorithm covered different sample sizes, ensemble sizes, kernels and their different combinations and different aggregation methods. Also the best gain for the different kernels(radial, linear, polynomial) were computed for comparison. Table 7.1 summarizes the results of most tests and highlights the best overall gain and the best ensemble type for each data set. The term in the brackets gives the difference to the gain of a single SVM.

Table 7.1: Bagging tests summary result table showing the best gains for each data set, the kernel types and the respective sample size or ensemble size. In the brackets the difference between the best single SVM and the best ensemble from each test is displayed

Data Set	Single SVM	SST max gain	SST type	SST size	EST max gain	EST type	EST size
Spam	93.68	**94.60(+0.92)**	LinRadPol	2700	**93.73(+0.05)**	LinRad	30
Satlog	**90.65**	90.48(-0.17)	Radial	2700	87.81(-2.84)	Radial	40
Optdig	97.94	**98.05(+0.11)**	Radialx3	2700	96.44(-1.5)	Radialx3	30
Adult	**85.10**	84.99(-0.11)	Radial	4000	84.86(-0.24)	Radialx3	30
Acoustic	**79.59**	78.81(-0.78)	Radialx3	4000	75.65(-3.94)	Radialx3	40
Acoustic Binary	**90.70**	90.28(-0.42)	RadPol	4000	89.27(-1.43)	LinPol	30

7.1.3 Influence of the Sample Size

The sample size was reduced to achieve the goal to significantly reduce the computation time, but it was still intended to maintain the overall accuracy of a single SVM. As in bagging unstable predictors deliver the overall best accuracy, the assumption was made, that by reducing the sample size we also get a more unstable predictor and with that, a good overall accuracy. The experiments show that, while there might still be this positive effect, the negative effect of giving each predictor less data is obvious dominating, causing a decreasing accuracy by using a reduced sample size. This may be caused by making each predictor weak in their class prediction. In the literature it is also stated that using bagging with stable predictors, the accuracy gets mildly smaller [1]. Following this statement, using nearly all of the training data for each SVM and thus having stable predictors should have a negative effect on overall bagging accuracy. This effect, thus small, is visible for the optdig and spam dataset for the polynomial kernel. But it is not sure that this really goes back to this effect or has another cause, so it is hard to proof the initial assumption. Overall the reduction of the sample size leads in all experiments to an visible and distinct reduction of the gain, but to a significant gain in computation performance, which was in the end the main goal of this study. The reduction is therefore a trade-off between gain and computation time and thus introducing an optimization problem. In terms of big data the view is a little different, as the significant reduction of the sample size may be needed to have a training in feasible time and therefore allows the analysis of data sets, which were not graspable before.

7.1.4 Influence of Different Kernels

It was tried to prove the assumption that combining different kernels in one ensemble is beneficial to the accuracy, which was made on basis of the knowledge obtained by using different kernels with the spam data set. This was conducted by investigating the behavior of different kernels and their combinations on other data sets. The results for the spam data set show, as initially stated, that the combination of different kernels is able to outperform a single SVM trained with the complete data, even with only by using only a fraction(10%) of the data per SVM in the ensemble. This effect is to be explained by the behavior of bagging, which prefers unstable predictors. Using different kernels in one ensemble introduces big differences between the single predictors. But this effect can only be observed for two out of six data sets, namely spam and acoustic binary. This makes it somehow difficult to explain. Both data sets are two-class problems and this type seem to work better with bagging and the probability voting method, but it is not visible if the behavior is caused by this fact only. In combination with the knowledge received from the sample size tests, the reasonable assumption my be stated that that introduced SVM bagging works great with unstable predictors, here introduced by the use of different kernels, together with strong prediction classifiers. But with the concluded experiments it is hard to tell if combining kernels should be the preferred method for analyzing data, and thus it only remains an option which has great potential for some data sets and should be further researched.

7.1.5 Influence of the Ensemble Size

The influence of the number of SVM predictors per bagging ensemble was another point in the investigation. The presumption was, that with rising number of SVMs, the accuracy increases, as the complete ensemble gets more information about the data. Also if not increase, at least it should not decrease and stay nearly stable instead upon a certain number of predictors. The experiments now have shown that the behavior is somehow different. First of all, in comparison to the sample size oder the type of kernel, the influence of the ensemble size for an size above 10 is rather low. So increasing the number of SVMs in this region leads only to a small increase in accuracy and has a somehow a hard cap, in the experiments often observed by a size of 30. It is also to say, even if not shown here to keep the already lengthy experiments section short, that an ensemble size below 10 has also a negative influence and a little faster decreasing curve of accuracy down to one SVM. This might also be explained with the here often discussed stability and instability of the predictors in the investigated bagging algorithm. If a lot of SVMs are used in one ensemble, it is two assume that the mean difference between the used predictors gets smaller and therefore the accuracy decreases mildly. But then it would be also to assume that for combining different kernels the same effect would kick in later, because with the use of them the difference in the learning set is greatly increased. But the behavior in terms of the ensemble size is due to the experiments not influenced by the type of kernel, so that it is not save to stick to the above described explanation of the behavior. But as the difference in accuracy is quite small as mentioned, it is to point out that to the use of a sample size between 10 to 30 gives the best results.

7.1.6 Majority vs Probability Voting

The change of the aggregation method from the often used majority voting to probability voting was due to the idea, that it is beneficial for the bagging algorithm to favor strong predictors. The experiments now have indicated that first of all that there is a difference between two-class and multi-class classification problems. For two-class problems, the above stated presumption seems to be valid, as the gain in accuracy for the probability voting method is quite high. So it is also save to assume, that strong predictors have a beneficial influence on the accuracy and should be supported. This is also visible in the sample size and kernel experiments. If it comes to multi-class problems the picture is different, as the gain more or less stagnates on one level or the best method changes from case to case. a possible explanation is as follows: If it comes to multi-class problems, the probability aggregation does not work this well, as there are too many different classes in the summing up of the probabilities. It is to assume that it then works more indifferent to a simple majority voting, as it is to expect that there is always one or two classes dominating the voting, regardless of the aggregation method. So it is to state, that for two-class problems, it is very beneficial to stick to majority voting, while for multi-class problems, it makes not much of a difference.

7.1.7 Optimization and Tuning

The optimization and tuning of the algorithm was not a main topic of this report, but it has always an important role if it comes to the goodness of the reached accuracy, specially to somehow complex algorithms like SVM bagging

with a lot of different parameters. The here used tuning was somehow simple, as it used the internal estimation for the gamma value of the radial kernel and a more or less *rule-of-thumb* approach for the cost. It was tried to have a simple tuning for single SVMs with these parameters for each kernel and data set, but early in the experiments it was learned, that the SVM bagging behaves different from a single kernel, so that the single SVM tuning parameters are not applicable. The optimization and tuning are therefore important topics for future research.

7.2 AdaBoost

7.2.1 AdaBoost Result Summary

Table 7.2: Mean prediction accuracies (%±sd) on all tasks with 10 experimental runs on every ensemble type and size versus one single SVM (first column). Each sub-classifier uses a training sample size of around 300 records. The best of all experiments are in bold font and the best of the SVM-AdaBoost ensemble are underlined.

Name	SVM	Radial	Polynomial	Linear	Mixed	RadPol	Combined
Spam	93.43 (±0.18)	93.34 (±0.39)	93.47 (±0.37)	92.96 (±0.41)	93.54 (±0.46)	93.75 (±0.18)	**93.87 (±0.19)**
Satellite	**91.00 (±0.23)**	90.03 (±0.43)	88.78 (±0.39)	84.42 (±0.61)	89.16 (±0.56)	90.47 (±0.30)	90.67 (±0.23)
OptDig	**97.91 (±0.11)**	97.75 (±0.18)	97.48 (±0.17)	96.55 (±0.18)	97.42 (±0.21)	97.74 (±0.19)	97.69 (±0.17)
Adult	**84.40 (±0.13)**	83.37 (±0.18)	83.17 (±0.15)	81.84 (±0.28)	83.61 (±0.19)	83.95 (±0.15)	84.20 (±0.10)
Acoustic	**90.70 (±0.09)**	89.46 (±0.37)	89.56 (±0.18)	87.44 (±0.12)	89.28 (±0.05)	90.01 (±0.15)	89.89 (±0.12)

Table 7.3: Mean times taken (seconds in logarithmic scale ±sd) to predict and test on all tasks with 10 experimental runs on every ensemble type and 50 SVM versus one single SVM (first column). Each sub-classifier uses a training sample size of around 300 records. The fastest of all experiments are in bold font and the fastest of the SVM-AdaBoost ensemble are underlined. Acoustic task was developed building 200 SVM.

Name	SVM	Radial	Polynomial	Linear	Mixed	RadPol	Combined
Spam	**0.15 (±0.05)**	2.44 (±0.04)	2.07 (±0.03)	2.77 (±0.07)	2.27 (±0.05)	2.96 (±0.03)	3.57 (±0.04)
Satellite	**0.27 (±0.01)**	2.37 (±0.10)	1.96 (±0.11)	2.59 (±0.12)	2.23 (±0.11)	2.88 (±0.10)	3.44 (±0.11)
OptDig	**0.49 (±0.03)**	2.59 (±0.02)	2.45 (±0.01)	2.51 (±0.01)	2.59 (±0.02)	3.22 (±0.01)	3.62 (±0.01)
Adult	4.83 (±0.41)	3.57 (±0.01)	**2.73 (±0.01)**	2.78 (±0.14)	3.22 (±0.07)	3.93 (±0.01)	4.20 (±0.03)
Acoustic	6.90 (±0.10)	6.58 (±0.07)	6.19 (±0.07)	**6.04 (±0.12)**	6.34 (±0.09)	7.10 (±0.07)	7.40 (±0.05)

Several experiments were proposed as an overview of the SVM-AdaBoost ensembles. First it was shown from tasks Spam and Optical Digit Recognition (Figures 6.5 and 6.4), by using 100% of the train set, that by boosting strong learners, as Wickramaratna et al. propose, the prediction accuracy deteriorates or does not increase and the times taken (Tables 6.22 and 6.21) to build the ensembles is sometimes 100 times more than the time taken to train one SVM. For this reason, the introduction of a boosting factor $bo.size$ showed that by weakening those strong classifiers a better or close accuracy performance from AdaBoost is obtained in comparison with one SVM and in less time. It was shown in Figure 6.6 from tasks Spam and Satellite that, if less samples are used to train each ensemble classifier, the better the performance of AdaBoost, the faster the time it takes to train the ensemble and the closer the accuracy results of one SVM, where it was noticed that beyond the factor $bo.size = 0.4$, the accuracy performance starts to stabilize or decay with bigger ensemble sizes. The different boosting factors $bo.size$ for the different tasks are shown in Table 6.23.

Tables 7.2 and 7.3 show the compiled outcomes from the SVM-AdaBoost experiments on all tasks against one SVM results. From Table 7.2, it is noticed that the SVM-AdaBoost only obtained a best accuracy than one SVM on the Spam task, but notice as well that for almost all tasks, the All-Combined results perform better than the independent kernel ensembles and gave a lower standard deviation, achieving as well closer results to the ones obtained with one SVM. Concerning the times, Table 7.3 shows that, for the smaller tasks used in this research, one SVM is always faster, but for bigger data sets, the best times were achieved by the SVM-AdaBoost ensemble and showing also smaller standard deviations than one SVM. It can be said that, for bigger data sets, there is

always room to increase the accuracy obtained by incrementing the All-Combined SVM-AdaBoost ensemble size without investing too much time on training, whereas on the other hand, for smaller data sets, it is possible to obtain good results by investing a little more time in comparison with one SVM.
It was shown as well, that between the many factors that influence the prediction accuracy of SVM, the amount of support vectors used to train the model plays an important role, which can determine if the model will over-fit or will give reliable predictions. On the other hand, AdaBoost calculates a weight for every sample on the train set depending on the previous goodness of classification α, and based on this, a new set is resampled with replacement to train the next weak classifier in the ensemble. The combination of these two factors; the number of support vectors and the frequency of sample selection from the original train set due to resampling and weighting in AdaBoost, over-fitted in general the whole ensemble. This has proven that, with a bad combination of number of samples and Boosting, the prediction accuracy deteriorates because of overfitting in the SVM classifiers (which are not sufficiently weak, but strong classifiers) and the resampling process with replacement inside the algorithm, where the introduction of the precise $bo.size$ factor can lead to better results for an SVM-AdaBoost ensemble.

7.2.2 Conclusions AdaBoost

Building SVM-Ensembles has proven that with a full train size, AdaBoost does not yield in any improvement for any size of data set. This lead the experiments to try to weaken the train set inside the SVM-Ensemble with the introduction of the boosting factor $bo.size$, where an improvement was noticed if the ensemble size increased. Some cases showed that there was saturation with the "All-Combined" option when using big ensemble sizes and non or little improvement was observed. It can be said that, by using the boosting factor, big data sets showed a better improvement whenever more SVMs were used to build the ensemble, where the required training time taken was less than one SVM and the accuracy obtained was the same or close to it. When SVMs are used in combination with AdaBoost, over-fitting was observed with any train size. The boosting factor helped avoid this problem by reducing the training set used for the next prediction and calculate the new error and weights based on the predicted samples with the full train. This over-fitting was observed mainly by the increment of support vectors used for each weak classifier, where always at the last iteration, there was noticed that all samples were selected. Also SVM-Ensembles with single kernels delivered good results, but not better than one single SVM. The combination of the single kernel ensembles always showed better results than their source ensembles, even when one of them didn't proved the accuracy close to the other, demonstrating that this combination is more reliable and more stable. In general it can be said that SVM-Ensembles with AdaBoost bring improvements on the performance of the method, but on accuracy results, it only delivered similar, but not better outcome for some data sets, whereas for big data sets, good results can be obtained if the ensemble size is big requiring less training time than one SVM.

8 Conclusion

After discussing the motivation and goal of reducing the computational effort of SVM model training, the different approaches from current research works on how to make SVMs a good choice for big data tasks were shown. In addition to this current research, the current case study is an extended analysis of the two ensemble based methods, Bagging and AdaBoost, proposed in this paper as a promising solution to achieve the current goal. These implementations were then tested in an extensive experimental evaluation covering different well-known data sets. The results of these experiments have demonstrated that bagging and AdaBoost SVM ensembles are capable of reaching good results, while at the same time reducing the amount of computational time needed. Although for the most investigated data sets, the quality of a single SVM could not be reached, but the difference in gain is low. Due to the significant sample size reduction for the AdaBoost and bagging algorithm and the ability to parallelize the complete bagging process, the reduction of computational effort can be tremendous for large data sets. If the computational time is a critical factor even before the best overall gain, AdaBoost and bagging are seen as suitable methods for ensemble-based SVM classification.
Both methods proved that they have advantages and disadvantages in different categories. On accuracy, both showed that a combined ensemble of different kernels is able to outperform ensembles of single kernels and, for some data sets, the accuracy of one SVM. Bagging presented this results whenever a high sample size was used. On the other hand, AdaBoost apparently showed accuracy saturation after an specific ensemble size for medium/small data sets, but for large data sets, this behavior was not noticed, approaching to the one of a single SVM with a better training time as well. On the training time, one of the biggest advantages of Bagging is that it is highly

parallelizable, making it suitable to distribute the different tasks in large clusters. For AdaBoost this is not possible, but the introduction of the boosting factor $bo.size$ helped achieve to reduce the training time of each SVM in the ensemble, leaving room for accuracy improvement avoiding over-fitting. For bagging the accuracy decays as the sample size gets smaller, and the gain does not improve by adding more SVMs to the ensemble, reaching somehow a maximum limit. AdaBoost showed also this behavior whenever the training size of the ensemble was close to 100%, where over-fitting was observed yielding to no better results as the ensemble size is increased. Bagging showed that to achieve a high accuracy the sample size has to be high, which is an indication that is prefers strong predictors. This concludes in higher training times per SVM in the ensemble, whereas AdaBoost did show as well that for medium/small data sets the training time was not better than one single SVM.

9 Future Work

This study was not able to tackle all the questions and has also raised some new ones. The following points are seen as very interesting and should be investigated in future works:

- It was shown that combining different kernels can be beneficial for some data set, but the cause is still unclear. Future studies should try to find the cause of the good performance by conducting respective tests with suitable data sets.
- The results for the SVM bagging indicate that there is a connection between the stability of the learners and the strength of the predictors. Good accuracy in SVM bagging seem to be obtained by combining unstable learners with strong predictors, if this is a true assumption and if it can be generalized in some way would be an interesting topic for future research.
- The different SVM kernels show great varieties in the reached accuracy, it may be interesting to see if ensemble methods could be used to identify the best kernels for each data set before conducting the complete runs. This could be done on basis of the type of data, e.g. if it is separably and has linear features linear kernels would be a good choice, or on basis of preliminary runs. Then for the ensemble only the best kernels could be used, to save computation time and achieve the best results.
- The tuning of the parameters is always a very important topic. The introduction of different kernels with their respective parameters in one ensemble and also the parameters for each algorithm(bagging/AdaBoost) makes the tuning a challenging task with a high potential to improve the accuracy. There may also be some correlating of the kernels effects which could be identified and could bring these algorithms to a whole different level.
- The experiments have shown that for most data sets, the accuracy of a single SVM could not be reached, but the computation time could be reduced significantly. It would be interesting to see a comparison of a single SVM vs the ensemble methods by giving both methods the same computation time on a single thread/core. It is to be expected that given a multi-core CPU or cluster, the ability to parallelize is dominating the computation time. But this matter could also be object of future research.
- The $bo.size$ factor in the SVM-AdaBoost algorithm showed that it can improve the prediction of the ensemble by using the right factor in relation with the size of the data set. A deep analysis with different data sets on the performance of SVM-AdaBoost would show that an optimization of this parameter could benefit the prediction.

References

1. Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
2. Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
3. B. Caputo, K. Sim, F. Furesjo, and A. Smola. Appearance-based object recognition using svms: Which kernel should i use? In *Proc of NIPS workshop on Statistical methods for computational experiments in visual processing and computer vision, Whistler*, volume 2002, 2002.
4. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
5. Marco F Duarte and Yu Hen Hu. Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):826–838, 2004.
6. A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.
7. Yoav Freund and Robert Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.
8. Steve R. Gunn. Support vector machines for classification and regression. *ISIS technical report*, 14, 1998.
9. Lutz Hamel. *Knowledge discovery with support vector machines*. John Wiley & Sons, Hoboken and N.J, 2009. ISBN 978-0-470-37192-3.
10. Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: Data mining, inference, and prediction*. Springer, New York, 2 edition, 2009. ISBN 978-0-387-84858-7.
11. Simon Haykin. A comprehensive foundation. *Neural Networks*, 2, 2004.
12. Martin Hilbert and Priscila López. The world technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011.
13. Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004. URL `http://www.jstatsoft.org/v11/i09/`.
14. Alexandros Karatzoglou, David Meyer, and Kurt Hornik. Support vector machines in r. 2005.
15. Hyun-Chul Kim, Shaoning Pang, Hong-Mo Je, Daijin Kim, and Sung Yang Bang. Constructing support vector machine ensemble. *Pattern recognition*, 36(12):2757–2767, 2003.
16. Lubor Ladicky and Philip Torr. Locally linear support vector machines. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 985–992, 2011.
17. Xuchun Li, Lei Wang, and Eric Sung. Adaboost with svm-based component classifiers. *Engineering Applications of Artificial Intelligence*, 21(5):785–795, 2008.
18. David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2012. URL `http://CRAN.R-project.org/package=e1071`. R package version 1.6-1.
19. Oliver Meyer, Bernd Bischl, and Claus Weihs. Support vector machines on large data sets: Simple parallel approaches. 2013.
20. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL `http://www.R-project.org/`. ISBN 3-900051-07-0.
21. Robert E Schapire. A brief introduction to boosting. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1401–1406. LAWRENCE ERLBAUM ASSOCIATES LTD, 1999.
22. Kai Ming Ting, Jonathan R Wells, Swee Chuan Tan, Shyh Wei Teng, and Geoffrey I Webb. Feature-subspace aggregating: ensembles for stable and unstable learners. *Machine Learning*, 82(3):375–397, 2011.
23. Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(1):363, 2006.
24. Giorgio Valentini. Random aggregated and bagged ensembles of svms: an empirical bias–variance analysis. In *Multiple Classifier Systems*, pages 263–272. Springer, 2004.
25. Shi-jin Wang, Avin Mathew, Yan Chen, Li-feng Xi, Lin Ma, and Jay Lee. Empirical analysis of support vector machine ensemble classifiers. *Expert Systems with Applications*, 36(3):6466–6476, 2009.
26. Jeevani Wickramaratna, Sean Holden, and Bernard Buxton. Performance degradation in boosting. *Multiple Classifier Systems*, pages 11–21, 2001.
27. Graham J. Williams. *Data mining with Rattle and R: The art of excavating data for knowledge discovery*. Springer, New York, 2011. ISBN 978-1-4419-9890-3.
28. Hwanjo Yu, Jiong Yang, Jiawei Han, and Xiaolei Li. Making svms scalable to large data sets using hierarchical cluster indexing. *Data Mining and Knowledge Discovery*, 11(3):295–321, 2005.

29. Ji Zhu, Saharon Rosset, and Trevor Hastie. A new multiclass generalization of adaboost. *Ann Arbor*, 1001: 48109.
30. Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie. Multi-class adaboost. *Ann Arbor*, 1001(48109):1612, 2006.

A AdaBoost Important Files

In the following, the most important code files in the Src.d folder for the SVM-AdaBoost algorithm are listed, which can further be used for future research:

Source functions in Subfolder .../Src.d/SVM Forest

- *SVMAdaBoost.R*: Source file for all code used in SVM-AdaBoost.
- *AdaBoost_ import results.R* File to analyse results from excel files generated on the experimental loops.

Experiments in Subfolder .../Src.d/

- *TestAdaBoost_ AdultData.r*: File to run experiments on Adult data set.
- *TestAdaBoost_ Satellite.r*: File to run experiments on Satellite data set.
- *TestAdaBoost_ Optdigit.r*: File to run experiments on Optical Digit Recognition data set.
- *TestAdaBoost_ AcousticData.r*: File to run experiments on Acoustic data set.
- *TestAdaBoost_ SPAMData.r*: File to run experiments on Spam data set.

B SVM Bagging Important Files

In the following the most important code files in the Src.d folder for the SVM bagging are listed, these can be used to perform future research:

Subfolder SVM Forest

- *SVMforestParallel.R* SVM bagging main functions and the main test loop
- *data_ sets.R* script for reading in the data, preparing it and dividing it into test and training sets
- *ParallelInit.R* Sources the libarys needed for parallel execution of the algorithm
- *BagPlotTable.R* Methods for creating the result tables
- *BagTest** where * is the name of a data set The run script for the sample and ensemble size experiments, including all parameter settings
- *BaggingTests* Test Playground covering different experiments, not sorted, maybe interesting to get ideas

Result Files
The result files and plots can be found in the respective data set folder, the naming is as follows: dataset_testmethod_startvaluetestvariable_end
e.g. Adult_SST_500_3k_10ES. Most results are in .csv table format including the features: type, enssize, samplesize, linear.gain, polynomial.gain, radial.gain, ensemble.gain, radialx3.gain, radpol.gain, linpol.gain, linrad.gain, and seed. they also come with plots in .pdf plot.