university of groningen / faculty of mathematics and natural sciences / artificial intelligence

Comparing Advanced Machine Learning Techniques for Classification Problems

(Bachelorproject)

Ruben van Dijk, R.van.Dijk@rug.nl,
M.A. Wiering*

July 22, 2016

Abstract

This research compares (and combines) multiple advanced machine learning methods: the multi-layer support vector machine (ML-SVM), extreme gradient boosting (XGB), principal component analysis (PCA) and ensemble learning methods (bagging and stacking). These methods are compared on a large dataset for classification. On this dataset, XGB combined with an ensemble method achieved the highest accuracy. The eigenvectors obtained using PCA did not contribute to any promising results for XGB. Combining the dataset with principal components also did not contribute to promising results for XGB. Training the ML-SVM is a time consuming computation, especially on large datasets. The ML-SVM uses particle swarm optimization (PSO) in order to optimize its algorithm. In this research the complexity of the PSO on the ML-SVM is reduced by decreasing the search space of the PSO. These results are compared to the original results of the ML-SVM and the overall accuracy of the less parameter tuned ML-SVM was slightly less than the original, but higher than a single SVM. Also XGB is compared to these results: XGB performed worse than the ML-SVM on smaller datasets.

1 Introduction

An implementation of a support vector machine (Vapnik, 1998, 2013), one that does not use one layer but several, is a new learning algorithm developed by (Wiering and Schomaker, 2013; Wiering, Schutten, Millea, Meijster, and Schomaker, 2013) called the multi-layer SVM (ML-SVM). This step seems only natural after the invention of the back-propagation algorithm that allows the use of multiple layers in perceptrons (known as the MLP) (Rumelhart, Hinton, and Williams, 1985). The ML-SVM is capable of dealing with regression, classification and dimensionality reduction problems, however, this research only focuses on classification problems.

Another new and promising supervised learning algorithm, extreme gradient boosting (XGB), is an implementation of a gradient boosting framework by (Friedman, 2001), but it is more efficient, scalable and portable (Chen and Guestrin, 2016). This XGB algorithm contributed to several Kaggle competition winning solutions, or almost winning a Kaggle competition (Taieb and Hyndman, 2014). These promising results show that XGB is an algorithm worthy to measure the ML-SVM against.

In this research, the ML-SVM and XGB algorithms are compared on how well they perform on the classification of several smaller datasets from the UCI repository (Asuncion and Newman, 2007) and one large dataset from a Kaggle competition. Not all small datasets are binary, they contain at most 7 target classes. In the competition for the large dataset, the machine learning algorithms have to predict whether a customer of the Santander Bank is satisfied or dissatisfied (binary classification) using ± 370 anonymized features. Therefore one of the research questions of this research is: *How does the multi-layer support vector machine perform compared with extreme gradient boosting?* The hypothesis is that XGB is not able to outper-

*University of Groningen, Department of Artificial Intelligence

form the results of the ML-SVM for the smaller datasets, because of the complexity of the ML-SVM. However, due to the simplicity of the XGB algorithm, it is expected that it will outperform the ML-SVM on the large dataset.

The ML-SVM has around 15 metaparameters such as two learning rates for the different layers, which have to be tuned for each individual dataset. In (Wiering and Schomaker, 2013), the amount of time spent on automatically tuning these parameters on the largest UCI dataset is at most two days. Tuning these metaparameters is done by particle swarm optimization (PSO) (Kennedy, 2011). The PSO method used by the ML-SVM is a computational demanding task when a lot of train data is used, as discussed in (Wiering and Schomaker, 2013). For even larger datasets, it is not attractive having to wait longer than two days before having a final result. Hence, another objective of this research is to reduce the complexity of the metaparameter search, done by PSO, for the ML-SVM on classification datasets: *How to reduce the complexity of the multi-layer support vector machine?* The hypothesis for this question is that the accuracy will decrease as the ML-SVM uses a smaller metaparameter search. This follows from the fact that the ML-SVM tries less combinations for the metaparameters it has and thus has a smaller chance of finding the most optimal combination.

The general principles of the SVM, ML-SVM and XGB algorithms will be explained below in order to answer the research questions. Furthermore, the machine learning techniques principal component analysis (PCA), and the ensemble methods bagging and stacking are explained too. These techniques are applied to the XGB algorithm in order to attempt to improve their accuracies in the classification of the large dataset.

2 SVM

In order to understand the ML-SVM, first its main component will be explained: the support vector machine. The support vector machine is a promising supervised learning algorithm for regression and classification of datasets (Vapnik, 1998, 2013). Those who are interested in the math should look up the article of (Fletcher, 2009), it explains the math behind classification of SVMs in full detail. In this section it will be explained more generally. Figure 1 shows the general idea of an SVM separating two classes (black and white dots) in two dimensions. Suppose that in a dataset, which is

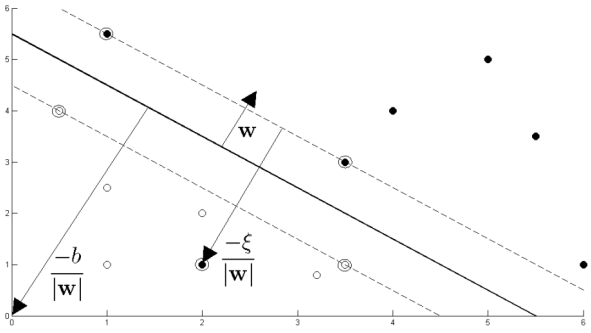

Figure 1: Two classes are separated by the margins and the hyperplane of a support vector machine, a two dimensional example. Image from (Fletcher, 2009)

not necessary linearly separable, there are N training points, then each row (x_i) has a total of D attributes, or has D dimensions. Each x_i belongs to a class c: 1 or -1 ($y_i^c \in \{-1, 1\}$). The variable ξ_i allows misclassified points. In summary, the data looks as follows:

$$\{x_i, y_i\} \text{ where } i = 1 \dots N, y_i^c \in \{-1, 1\}, x_i \in \Re^D \tag{2.1}$$

A hyperplane separating the two classes is described by the following function (similar to figure 1):

$$\vec{w} \cdot \vec{x} - b = 0 \tag{2.2}$$

Note that the sign in front of b is negative here (as in figure 1). The separating hyperplane of the SVM is located in such a way that the closest members of each class are as far away as possible. The goal of the SVM is to select $\vec{w}$ and b such that it satisfies the following constraints:

$$\begin{cases} x_i \cdot \vec{w} - b \geq +1 - \xi_i \text{ for } y_i^c = +1 \\ x_i \cdot \vec{w} - b \leq -1 + \xi_i \text{ for } y_i^c = -1 \end{cases} \tag{2.3}$$

$$\xi_i \geq 0 \ \forall_i \tag{2.4}$$

These two formulae can be combined in one equation by:

$$y_i^c(x_i \cdot \vec{w} - b) - 1 + \xi_i \geq 0 \text{ where } \xi_i \geq 0 \ \forall_i \tag{2.5}$$

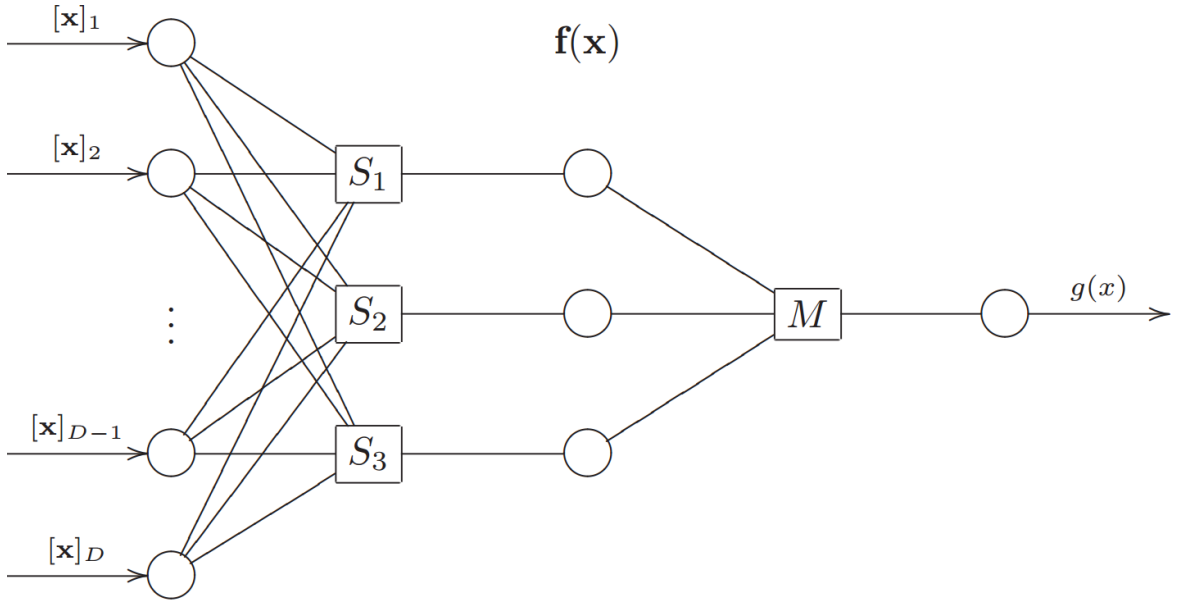

Figure 2: Architecture of a two-layer SVM. In this example, the hidden layer consists of three SVMs S_a, image from (Wiering and Schomaker, 2013).

The support vectors (e.g. $\vec{x} \cdot \vec{w} - b = -1$) of the SVM are determined by the train data points that lie closest to the hyperplane and are on the correct side of the hyperplane. In figure 1 these are surrounded by additional circles. Data points on the incorrect side have a penalty that increases as the data point lies further away from the hyperplane.

The length of a margin is equal to $\frac{1}{||w||}$. Maximizing the margin with respect to the constraints of equation 2.5 results in finding $\min||\vec{w}||$. However minimizing $||w||$ is equivalent to minimizing $\frac{1}{2}||\vec{w}||^2$. The variable C accounts for the penalties (ξ_i) and the size of the margin. The following formula can be constructed:

$$\begin{aligned} &\min \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{N}\xi_i \\ &\text{s.t. } y_i^c(x_i \cdot \vec{w} + b) - 1 + \xi_i \geq 0 \ \forall_i \end{aligned} \tag{2.6}$$

From this formula the $\vec{w}$, b and ξ_i can be found. Lagrange multipliers are used to obtain a primal and dual objective function, more details on the math are in (Fletcher, 2009). Eventually, the tutorial of (Fletcher, 2009) shows how (using Langrange multipliers, differentiating and a quadratic problem solver) the values for $\vec{w}$, ξ_i and b can be obtained.

3 ML-SVM

Now that the general concept of the SVM is clear, the ML-SVM can be explained. An SVM uses a single layer to learn from the data, thus, combining several SVMs leads to the ML-SVM as figure 2 shows. In the paper of (Wiering and Schomaker, 2013) the following formula ($f(\vec{x}|\theta)_a$) is used for computing the hidden layer representation for a SVM S_a for input $\vec{x}$:

$$f(\vec{x}|\theta)_a = \sum_{i=1}^{N}(\alpha_i^*(a) - \alpha_i(a))K_1(x_i, \vec{x}) + b_a \tag{3.1}$$

The variable θ represents trainable parameters in these SVMs and $K(\cdot,\cdot)$ is the kernel function. In (Wiering and Schomaker, 2013) it is argued that each result of a SVM relies heavily on their selected kernel function K_1 (and K_2 for the ML-SVM). The kernel function used in the ML-SVM is a radial basis function that enables the SVMs to classify non-linear data. In equation 3.1, $\alpha_i^*(a)$ and $\alpha_i(a)$ are the coefficients of the support vectors for S_a, they can be adjusted to a local maximum by a gradient ascent learning rule computed by the main SVM

M. This rule is computed by:

$$\alpha_i^c \leftarrow \alpha_i^c + \lambda(1 - \sum_{j=1}^{N} \alpha_j^c \cdot y_j^c \cdot y_i^c \cdot K_2(f(x_i|\theta), f(x_j|\theta))) \tag{3.2}$$

In this equation λ represents the learning rate for α_i^c. The support vector coefficients stay between 0 and C due to another metaparameter c_1, this way the bias constraint is respected. The ML-SVM uses a technique similar to backpropagation that constructs new perturbed train sets for each hidden layer SVM S_a. Each S_a then trains again using the gradient ascent learning rule on these perturbed train sets. Every SVM outputs to the main SVM M which learns to map the output of the SVMs (see equation 3.3).

$$g_c(f(\vec{x}|\theta)) = \sum_{i=1}^{N} y_i^c \alpha_i^c K_2(f(x_i|\theta), f(\vec{x}|\theta)) + b_c \tag{3.3}$$

To enforce symmetry breaks, the different hidden-layer SVMs are randomly initialized and then, in the initialization phase, trained to output a particular target class. The ML-SVM has 27 metaparameters (such as the hidden-layer size) to be tuned. As discussed in the introduction, these metaparameters are to be tuned by PSO. In (Wiering and Schomaker, 2013) the PSO method tries around $1 * 10^6$ combinations of these metaparameters. An effective implementation of PSO is used in order to prevent PSO of trying less promising sets of metaparameters. Eventually, the most promising metaparameter set is used for training the ML-SVM model.

4 XGB

As discussed earlier, XGB is an improved implementation of the gradient boosting algorithm. Therefore, the gradient boosting algorithm is explained in order to understand the extreme gradient boosting algorithm itself. The tutorial of the author of XGB is used (Chen and Guestrin, 2016).

Classification and regression trees (CART) are used instead of decision trees (only leaves have decision values in CART). Figure 3 shows how a tree could be composed. The models in this figure try to predict whether each person likes video games

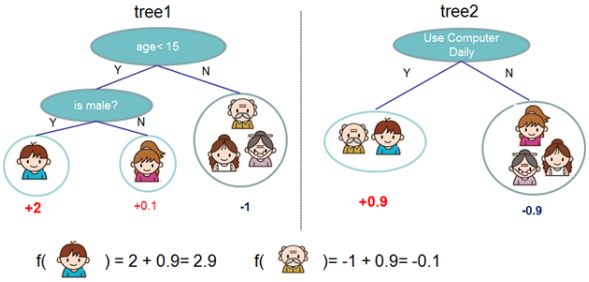

Figure 3: An example of two classification and regression trees. Image from (Chen and Guestrin, 2016)

or not. Each leaf in a tree has a value that represents the prediction of the target. E.g. the value 2.9 is obtained by summing over the prediction values of all tree leaves in which that person occurs. Some parameters used in this algorithm are: K, the number of trees, f a function that is in the set of all possible CARTs. The main model that is to be optimized by the algorithm look as follows:

$$Obj(\Theta) = l(\Theta) + \Omega(\Theta) \tag{4.1}$$

l is a differentiable convex loss function that computes the error between the predicted value and the actual value (e.g. square loss). Ω is a function to control the complexity of the model. Adjusting this formula for gradient boosting then gives some more parameters: T, the number of leaves in the tree and w representing the leaf weights. The models for gradient boosting and random forests (Liaw and Wiener, 2002) do not differ from each other. The distinction between those algorithms is made in the training part. The parameters to be learned cannot be learned using Euclidean space optimization methods. An additive ensemble strategy, boosting (Freund and Schapire, 1995), is used in order to compute these parameters. Boosting adds one tree at a time and compensates or improves for what it has learned previously. Therefore, boosting is able to focus on the bad parts of its own model. In one iteration the algorithm tries to find a classification tree that improves the current model of classification trees the most. These details can be found in (Chen and Guestrin, 2016). However, the general principle for XGB should be clear. A notable feature of XGB is that it is able to deal with sparse matrixes which makes it 64 times as fast.

5 Principal Component Analysis and Ensemble Learning

In this section the basic principles of principal component analysis and two ensemble learning methods stacking and bagging (used in this research) will be explained.

5.1 Principal Component Analysis

Prior to performing PCA (Pearson, 1901), the data is transformed (or preprocessed) such that each feature is normalized (feature scaling). Normalization is done by computing the minimal and maximal values over each feature. For each feature each element is subtracted by that feature's minimal value and divided by its maximal value minus its minimal value:

$$x' = \frac{x - \min(x)}{max(x) - min(x)} \tag{5.1}$$

This causes each feature to range from [0,1] and scale proportionally. After normalization, PCA computes the eigenvalues of the covariance matrix and sorts the components on importance (highest eigenvalue). Thus, the newly obtained matrix consists maximally of n features where n is the amount of features in the dataset. For reference figure 4 shows centered data of two PCA components (that does not range from [0,1]). Normally, PCA is used in order to reduce dimensionality and eliminate noise of the dataset. In the case of this research 360 features are reduced to 100 and 200 components. This research also increased the dataset by adding the obtained PCA features (150, 200, 250 and 300) to the original dataset.

5.2 Ensemble Learning Methods: Bagging and Stacking

Bagging (or Bootstrap AGGregating by (Breiman, 1996)) and stacking (Wolpert, 1992) are similar in that both methods use predicted output files by a classification algorithm. Multiple classification methods, that differ in the algorithm used or just a parameter value, generate different predicted output files.

In bagging these output files are added to each other and then the average is computed. Eventually, a threshold is used for deciding the final class.

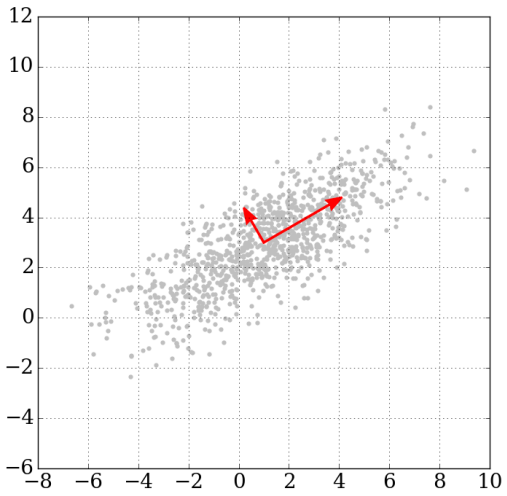

Figure 4: An example of two principal components in a principal component analysis. Image generated in Python.

This final file is an average of all models provided to the bag. For more classes all probabilities for each class have to be taken into account (and summed over). In general bagging reduces the variance and prevents overfitting.

In the stacking ensemble learning method, the output files generated by each model become new features. Hence, each output file is a new feature. The newly obtained files are used to train the model on (for example XGB).

6 Experiments and Results

In this section the performances of the algorithms ML-SVM and XGB in combination with other machine learning techniques are compared. Afterwards, the results of reducing the complexity of the ML-SVM will be shown and discussed.

6.1 Experiments on Comparing the ML-SVM and the XGB algorithms on a Kaggle competition

The train and test datasets from the Kaggle competition contained information from ±75000 persons

and there were ±370 features for each person (e.g. age). This resulted in ±28 million elements in each dataset. In order to reduce the amount of data, some features were removed. The features removed in both datasets were features that contained the same value for each person in the train dataset, e.g. a column that only has the value 0 in it. In the same way for the datasets, duplicate columns were removed, thus if column 1 was equal to column 2 in the train dataset, column 2 got removed in both sets. This reduced the dataset size with ±17%.

For experiments that used PCA, a normalization is done over the train matrix, then the train matrix was sorted on the obtained eigenvalues (see the section about PCA), afterwards the same transformation (normalization and sort) of the train dataset is applied to the test data. In this research, features obtained by PCA were used in order to train and test on, or were used in combination with the original train and test data (stack), however this increased the amount of dimensions from 300 to 300 + {100, 150, 200, 250, 300}. For the stacking ensemble method, XGB was used such that it generated its own train dataset. This was done by letting XGB train and test using the train dataset for both. Eventually, all files obtained (containing output probabilities) were combined in a new stack training dataset. At last for the bagging ensemble method, output files of multiple cross validations for XGB were combined (and taken the mean of) or the output files that have high scores were combined. Each individual XGB run is based on 560 classification trees that have a maximum depth of 5. The results can be shown in table 1.

Table 1 shows a public and private score for the 13 highest scoring accuracies (based on the private scores). The public and private score are used in the competition to prevent overfitting models. The public score of each model is based on half of the test data, and after the competition has finished the final private scores are calculated on the other half of the data, these scores determined the winners of the competition.

As can be seen from table 1, a bag of XGB runs that used 50% of the training data and 20 cross validations performed the best, using a lower amount of train data resulted in lower accuracies. A single run of a XGB using all train data also performed really well, this might be explained by the fact that XGB itself already is a kind of bag itself (of 560 classification trees). Another bag of the 5 best files then performed the best, this bag of best files was based on older obtained results with lower scores. Ideally, a bag over the best files should also have been computed on newer files with higher scores. However, this was not possible because of the submission limit of Kaggle which was 5 submissions per day. The ML-SVM did perform well, but obtaining results did take too long. In this experiment, the ML-SVM was (slowly) trained on 5% of the data. The results from the ML-SVM in table 1 distincts from ML-SVM(2), because another set of metaparameters found by PSO were used. Adjusting the ratio train and test data, reducing or increasing data using PCA, stacking (by definition the ML-SVM is already a kind of bag of SVMs) or applying some other machine learning techniques that involve transforming the data are not tested in this research, due to the time consuming computations of the ML-SVM. At last, the scores of combining the training data with PCA features did perform worse than leaving these PCA features out, however only using PCA features did perform even worse (and therefore are excluded from table 1).

For reference: The winner of the Kaggle competition achieved an accuracy of 0.844838 and 0.829072 on the public and private scores respectively. On overall the ML-SVM was not able to achieve high enough scores, especially because it took to long

Table 1: Public and private scores on the Santander Customer Satisfaction dataset sorted on private scores. CV denotes the amount of cross-validations used.

Kind of submission	Public	Private
Bag 50% train, 20 CV	0.840959	0.827225
Normal XGB	0.840324	0.827165
Bag best 5 files	0.840595	0.827160
Bag 20% train	0.840736	0.827135
ML-SVM	0.838860	0.827112
ML-SVM(2)	0.839098	0.827083
XGB with 300 PCA	0.838957	0.826942
XGB with 250 PCA	0.839088	0.826527
XGB with 200 PCA	0.838901	0.826480
XGB with 150 PCA	0.839018	0.826020
Bag 10% train, 100 CV	0.841211	0.824441
Stack 20% train, 20 CV	0.838722	0.824135
Stack 50% train, 20 CV	0.837442	0.822212

for the ML-SVM, training only on 5% of the data, to achieve the scores shown in table 1. XGB on the other hand provided more promising results and was able to provide these results within fair amounts of time such that the amount of daily submissions on Kaggle were utilized maximally. This also suggests that reducing the complexity of the ML-SVM might improve the results obtained by the ML-SVM.

6.2 Experiments for Reducing Complexity of the ML-SVM

In (Wiering and Schomaker, 2013), the ML-SVM uses $1 * 10^6$ combinations of metaparameters in its search for the optimal combination for that particular dataset. Reducing this amount of evaluations to $1 * 10^4$ and setting around 15 metaparameters on default values speeds up the metaparameter search 100 times. The decision for which metaparameters to set on default is based on experience. In one experiment the metaparameter that determines the amount of hidden-layer SVMs used is returned to being a variable metaparameter.

The results of this experiment can be found in table 2. The second column shows the results obtained by the original ML-SVM in (Wiering and Schomaker, 2013). The third column (ML-SVM 1) shows the results for the complexity reduced ML-SVM for each dataset. The fourth column (ML-SVM 2) also shows results for a complexity reduced ML-SVM, but the layer size variable of the ML-SVM is not set on a default value. For reference, the accuracies of the ML-SVM on each UCI dataset are also compared with XGB, these are found in the last column of table 2. The results of XGB are obtained using the default metaparameter values from the Kaggle competition. Also a metaparameter tune was conducted by parameter tuning 6 metaparameters in the XGB algorithm. The amount of metaparameter combinations is 174+(7*nFeatures). The XGB algorithm is cross validated 100 times for each variable (10 times on different data subsets and for each subset 10 iterations of the XGB itself). Therefore, the total amount of metaparameter computations was ±17400. The optimal metaparameters obtained then were 10000 times cross validated. However, the results from parameter tuning XGB were worse on average than the values used from the competition, therefore these are used in this research.

Table 2 shows that reducing the search space for PSO and setting metaparameters on default values results in a accuracy loss of 0.6% (on average). However the algorithm now is 100 times as fast. The speed-up might be valuable when larger datasets are used (e.g. in Kaggle competitions) or deadlines are set. Excluding the layer-size variable as a default value for the same amount of search space evaluations led to an accuracy loss of 1.2%, this also means that all parameter combinations are more dependent on each other. Finding more optimal combinations requires more fine-tuning of parameters. The results suggest that the layer size of the ML-SVM could be set on a default value. The default value for the layer size used in this research was 20. All ML-SVM methods on average have higher accuracies that the XGB algorithm has. This shows that the ML-SVM is able to outperform the XGB algorithm and a single SVM which had an average accuracy over the same datasets of 87.5 in (Wiering and Schomaker, 2013). The standard errors for the less parameter optimized ML-SVMs and XGB are larger than the original ML-SVM, this is explained by the fact that the model trained by the original ML-SVM has more cross validations and therefore is more stable and reliable.

7 Discussion and Future Work

For deciding which algorithm (ML-SVM or XGB) performs best, it can be concluded that it depends on the size of the train dataset. On the smaller datasets ML-SVM outperforms the XGB algorithm and for a really large dataset XGB is outperforming the ML-SVM, because the ML-SVM is too slow to be trained on the whole dataset. In addition to the accuracy of the trained model, its complexity does matter too. In the Kaggle competition it took the ML-SVM too long in order to provide all original output files. The less parameter optimized ML-SVM is faster but still too slow. The complexity of the ML-SVM is of $O(N^2)$ for N train examples. A suggestion for future work is to improve how examples are chosen to be trained in order to decrease the complexity for the amount of training examples.

Furthermore, it seems that the ensemble method bagging also improved the results the most in this

Table 2: The means and their standard errors of the error rate on each dataset for the original, optimized 1, optimized 2 ML-SVMs and XGB. There are 1000 random cross validations performed on the original ML-SVM and 100 on the others.

Dataset	ML-SVM(orig)	ML-SVM 1	ML-SVM 2	XGB
Hepatitis	85.1±0.1	84.5±0.8	84.3±0.8	61.9±1.2
Breast Cancer W.	97.0±0.1	97.2±0.2	97.0±0.2	96.4±0.2
Ionosphere	95.5±0.1	94.8±0.4	95.6±0.4	93.7±0.4
Ecoli	87.3±0.2	86.0±0.6	85.2±0.6	85.9±0.6
Glass	74.0±0.3	72.3±0.9	70.0±1.0	79.0±0.5
Pima Indians	77.2±0.2	77.4±0.5	77.6±0.5	76.4±0.6
Votes	96.8±0.1	96.6±0.2	95.9±0.3	96.0±0.3
Iris	98.4±0.1	97.8±0.4	97.9±0.4	93.2±1.5
Average	88.9	88.3	87.9	85.3

research on the large dataset. The train test ratios used are 10/90 20/80 and 50/50 in which increasing the train size resulted in an accuracy increase, using even more train data might also have led to higher accuracies. PCA and stacking did perform worse than a single XGB run. All XGB runs for the large dataset in this research used the same metaparameter values. In order to validate that stacking or PCA methods on the XGB algorithm always perform worse (on average), the metaparameters could be tuned like the ML-SVM algorithm does. However, this will increase the complexity of the XGB.

Reducing the search space for metaparameters of the ML-SVM and setting these metaparameters on default results in an average accuracy loss on all small classification datasets used in this research. Including an extra tunable parameter in the search space may decrease the accuracy. In this case it meant that putting the amount of SVMs used in the layer size as a default value, improved the search for other variable metaparameters. It might be interesting to find out if there are more metaparameters that can be set on default values such that an accuracy increase is obtained. However most metaparameters such as the learning rates should not be set on default.

Due to the speedup of the complexity reduced ML-SVM it will be faster in finding results for classification datasets. When deadlines are to be met, the original ML-SVM simply takes too long to find its optimal metaparameters. The complexity reduced ML-SVM also has a complexity of $O(N^2)$. A method to improve the complexity is to learn on a small part of the dataset and focus on the bad trained examples, similar to XGB. Therefore, data pruning methods (Angelova, 2004) also can be examined in future work in order to reduce the complexity of the ML-SVM.

References

A. Angelova. *Data pruning*. PhD thesis, California Institute of Technology, 2004.

A. Asuncion and D. Newman. Uci machine learning repository, 2007.

L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *arXiv preprint arXiv:1603.02754*, 2016.

T. Fletcher. Support vector machines explained. *Online. http://sutikno. blog. undip. ac. id/files/2011/11/SVM-Explained. pdf.[Accessed 06 06 2013]*, 2009.

Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

J.H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

J. Kennedy. Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer, 2011.

A. Liaw and M. Wiener. Classification and regression by randomForest. *R news*, 2(3):18–22, 2002.

K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

S.B. Taieb and R.J. Hyndman. A gradient boosting approach to the Kaggle load forecasting competition. *International Journal of Forecasting*, 30 (2):382–394, 2014.

V.N. Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

V.N. Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

M. A. Wiering and L.R.B. Schomaker. Multi-layer support vector machines. In A. Argyriou J.A.K. Suykens, M. Signoretto, editor, *Regularization, Optimization, Kernels, and Support Vector Machines: CRC Machine Learning and Pattern Recognition Series, Boca Raton, USA, Oct 2014*. Chapman & Hall, 2013.

M.A. Wiering, M. Schutten, A. Millea, A. Meijster, and L.R.B. Schomaker. Deep support vector machines for regression problems. In *Proceedings of the International Workshop on Advances in Regularization, Optimization, Kernel Methods, and Support Vector Machines: theory and applications*, 2013.

D.H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.