A Competitive Learning Approach to Instance Selection for Support Vector Machines

Mario Zechner* and Michael Granitzer

Know-Center and Graz University of Technology
Inffeldgasse 21a
8010 Graz, Austria
mzechner,mgrani@know-center.at

Abstract. Support Vector Machines (SVM) have been applied successfully in a wide variety of fields in the last decade. The SVM problem is formulated as a convex objective function subject to box constraints that needs to be maximized, a quadratic programming (QP) problem. In order to solve the QP problem on larger data sets specialized algorithms and heuristics are required. In this paper we present a new data-squashing method for selecting training instances in support vector learning. Inspired by the growing neural gas algorithm and learning vector quantization we introduce a new, parameter robust neural gas variant to retrieve an initial approximation of the training set containing only those samples that will likely become support vectors in the final classifier. This first approximation is refined in the border areas, defined by neighboring neurons of different classes, yielding the final training set. We evaluate our approach on synthetic as well as real-life datasets, comparing run-time complexity and accuracy to a random sampling approach and the exact solution of the support vector machine. Results show that runtime-complexity can be significantly reduced while achieving the same accuracy as the exact solution and that furthermore our approach does not not rely on data set specific parameterization of the sampling rate like random sampling for doing so. Source code, binary executables as well as the reformatted standard data sets are available for download at `http://www.know-center.tugraz.at/forschung/knowledge_relationship_discovery/downloads_demos/sngsvm_source_executables`

1 Introduction

Support vector machines (SVM) [1] have been applied successfully in a wide variety of fields in the last decade. In their simplest form SVMs are essentially linear classifiers that try to find a maximum margin hyperplane which separates training samples of two different classes. The SVM problem is formulated as a convex objective function subject to box constraints that needs to be maximized, a quadratic programming (QP) problem. The final solution is based on just a subset of the training samples called support vectors.

* Student at Graz University of Technology

Solving this QP Problem is usually quadratic in time and storage. This is especially prohobitive in large scale domains such as data mining where millions of documents have to be processed. Various attempts at improving the runtime and space requirements have been made. One line of work focuses on reducing the training set to samples which will likely become support vectors of the final solution. Shin and Cho [2] select training samples based on their neighborhood. They exploit the fact that training samples with heterogeneous neighborhoods in terms of a neighbor's class label are likely to be near the hyperplane induced by the SVM and therefore being potential support vectors. As searching the k-nearest neighbors for all training samples has a time complexity of $O(n^2)$ they introduce an approximation schema that brings the time complexity down to $O(b * n)$ where b is the number of samples with heterogeneous neighborhoods. Their approach seems to have problems with well separated training sets. In [3] Yu et al. use clustering as a preprocessing step. The training set is clustered and the resulting centroids provide the training set for the initial SVM. The results are refined by replacing centroids near the decision surface with their associated training samples and retraining the SVM on those training samples. This procedure is repeated until all training samples have been used. Due to the cluster expansion criterion used their approach only works with linear kernels. Another similar but more general algorithm based on clustering was presented by Boley and Cao [4].

In this paper we follow the notion of speeding up SVM training by reducing the number of training samples. Our focus lies on a fast method that has better runtime complexity than $O(n^2)$ and is robust to parameter selection. Our sparsifying neural gas algorithm (SNG), a combination of learning vector quantization [5] and the growing neural gas algorithm, [6] automatically grows a topological network to reconstruct the input data set. The number of final neurons (i.e. nodes in the network) is determined on the fly. To select the relevant training examples for the SVM we first partition the training set with one SNG for each class. This takes a single pass over the complete training set. Next we merge the two sets of neurons and each neuron receives the label of the class it was trained on. In a second pass over the complete training set we establish connections between the closest two neurons for each training sample. This induces a subset of the Delaunay triangulation which we exploit to determine the border regions between classes. By expanding neurons of different class labels with the training samples in their "receptive field", border regions become densely populated while non border regions are thinned out.

The resulting number of training samples becomes the final SVM training set. Results show that this approach yields faster training time while maintaining most if not all of the support vectors of the full SVM solution. In particular, the contributions of our work are as follows:

1. A new, parameter robust neural gas variation called sparsifying neural gas (SNG) that reliably approximates the topology of the dataset in only 2 passes.

2. An instance selection schema based on the SNG for speeding up support vector machine learning while retaining the SVM's accuracy
3. An empirical study on the performance of our approach comparing it to random sampling as a baseline as well as the full solution of the SVM.

We introduce the SNG in section 2 followed by a review of the SVM formalism in section 3. Section 4 discusses the SVM instance selection in detail while we report results of our experiments in section 5. We conclude and give an outlook on future work in section 6.

2 Sparsifying Neural Gas

In its simplest form data reduction can be accomplished by random sampling and results in [7] indicate that this is the best overall model independent sample selection method. Random sampling is capable to reliably remove noise from a data set. However, due to its stochastic nature random sampling may yield unreliable results and might ignore small clusters of important samples. Furthermore, the sampling rate has to be provided as a parameter highly dependent on the specific data set.

A general approach to data reduction is classical vector quantization (VQ) [8]. In vector quantization a set of codebook vectors is determined maximizing a cost function of the quantization error. Usually a the number of code book vectors and thus the level of compression has to be specified a-priori. The growing neural gas (GNG) [9] by Fritzke overcomes this and other problems by adaptively increasing the number of codebook vectors. However, the decision on the final number of codebook vectors is still necessary as the GNG has a fixed insertion rate and other parameter specific properties. Also, a proof of convergence for the GNG is non-trivial given it's adaptive nature.

With the SNG we try to combine some of the favorable features of various VQ methods. For reducing the time complexity and applying the SNG to large data set our goal is to minimize the number of passes over the data. Competitive learning as used in many online vector quantization methods is our starting point. It has the flexibility to quickly adapt to relevant regions of the data set. For the SNG we utilize the Winner-Takes-All strategy starting with a given set $\mathcal{W}$ of k neurons each represented by a weight vector $\mathbf{w_i} \in \mathcal{W} \subset \mathbb{R}^d, 1 \leq i \leq k$ and a data set of n samples $\mathcal{X} = \{\mathbf{x_j}, 1 \leq j \leq n\}$. For each sample $\mathbf{x_s}$ drawn form $\mathcal{X}$ we select the nearest (winning) neuron $\mathbf{w_{win}}$ by

$$\mathbf{w_{win}} = \min_j ||\mathbf{w_j} - \mathbf{x_s}|| \tag{1}$$

and update it's weight vector as $\mathbf{w'_{win}} = \mathbf{w_{win}} + \mathbf{w_\Delta}$ with

$$\mathbf{w_\Delta} = \eta(\mathbf{x_s} - \mathbf{w_{win}}) \tag{2}$$

where η is the so called learning rate driving the adaptiveness of neurons to new input samples.

As we want a growing network which adapts to the data set topology an insertion criteria is needed. In its original version, Fritzke's GNG keeps a local error statistic for each neuron by tracking its online squared error. This error gets updated for a winning neuron by

$$error_{new}(\mathbf{w_{win}}) = error_{old}(\mathbf{w_{win}}) + ||\mathbf{w_{win}} - \mathbf{x_s}||^2 \tag{3}$$

Additionally, each time a fixed number of samples has been presented to the network a new neuron is inserted halfway between the neuron with the maximum accumulated error and its topological neighbor with the highest error. As outlined above the fixed insertion rate is reducing the flexibility of the network especially when only a single pass over the data is performed. Hence, the GNG parameterization becomes very data set specific. The newly created neuron does not explore new space but rather gives its "parent" neurons the opportunity to adapt to new regions. We therefore introduce a simple heuristic that removes the fixed insertion rate and lets the new neuron instantaneously explore new regions.

In the SNG we extend the fixed insertion rate by introducing another local statistic for each neuron, namely the number of samples a neuron was the closest to. We call this the hit statistic. Each time a neuron is determined as the winner for an input sample we update its hit statistic simply by

$$hits_{new}(\mathbf{w_{win}}) = hits_{old}(\mathbf{w_{win}}) + 1 \tag{4}$$

Combined with the accumulated squared error in equation 3 we can calculate an approximate mean squared error:

$$mse(\mathbf{w_i}) = error(\mathbf{w_i})/hits(\mathbf{w_i}) \tag{5}$$

Of course we have to account for the case $hits(\mathbf{w_i}) = 0$. As a winning neuron is changing its position and therefore the distance to previously seen samples the mse is a very rough approximation typically underestimating the true mse.

We interpret the online mse as the approximate squared radius of the perceptive field of a neuron, i.e. the average distance from the neuron to the samples it won. A new neuron is inserted if

$$mse(\mathbf{w_{win}}) < ||\mathbf{w_{win}} - \mathbf{x_s}||^2 \tag{6}$$

or in words: if the sample lies outside the perceptive field of the winning neuron. The perceptive field of a new neuron as well as its weight vector have to be initialized. We chose a very simple mechanism here and initialize the position $\mathbf{w_{new}}$ of the new neuron to the current samples position. The new neuron inherits the perceptive field of its "parent" neuron $\mathbf{w_{win}}$:

$$\begin{aligned} \mathbf{w_{new}} &= \mathbf{x_s} \\ error(\mathbf{w_{new}}) &= error(\mathbf{w_{win}}) \\ hits(\mathbf{w_{new}}) &= hits(\mathbf{w_{win}}) \end{aligned} \tag{7}$$

The network topology is initialized by randomly selecting two samples as neurons and setting their *error* and *hits* to zero. We consider the perceptive field of a winning neuron only if it has seen at least ν samples to avoid growing the network in an to early stage. Note that the perceptive field will only grow while the neuron has not seen ν samples yet and shrink otherwise. For avoiding overly dense regions of neurons we introduce a repulsive behavior between neighboring neurons based on their perceptive field. In the course of determining the winning neuron $\mathbf{w_{win}}$ for a sample we also calculate the second nearest neuron $\mathbf{w_{sec}}$. If the receptive fields of the two neurons overlap we want the second winning neuron to move away from the winning neuron by a small fraction called the repulsion rate ρ. The repulsion criterion is given by:

$$mse(\mathbf{w_{win}}) + mse(\mathbf{w_{sec}}) > ||\mathbf{w_{win}} - \mathbf{w_{sec}}||^2 \tag{8}$$

In case the repulsion criterion is meet we update the second neuron vector as $\mathbf{w'_{sec}} = \mathbf{w_{sec}} + \mathbf{r_{\Delta}}$ with

$$\mathbf{r_{\Delta}} = -\rho(\mathbf{w_{win}} - \mathbf{w_{sec}}) \tag{9}$$

This step reduces the overlap of the perceptive fields of two neurons and improves the distribution of the neurons over the data set. It should be noted that occasionally neurons can exist having no sample associated with them. We simply remove those neurons in the final step of the SNG. When all samples of the data set have been presented to the network we perform a second pass over all samples to determine the induced Delaunay triangulation of the network. We again determine the nearest and second nearest neuron for each sample and construct an edge between those neurons. The emerging graph then represents the overall topology of the network. Additionally we note for each sample to which neuron it belongs to. The final result of the SNG is therefore the set of final neurons $\mathcal{W}$, a set of edges $e_{ij} \in \mathcal{E}$ denoting the induced Delaunay triangulation as well as a set $\mathcal{C}_i$ for each neuron $\mathbf{w_i}$ containing the samples $\mathbf{x_s}$ the neuron is closest to. Algorithm 1 depicts the complete procedure of the SNG.

Due to the nature of the SNG there are a couple of caveats. From the insertion rule in inequality 6 it becomes clear that the SNG will aggressively insert new nodes. The SNG does not strictly optimize a cost function as in the case of LVQ or NG and is similar to GNG in that respect.

One downside currently is the absence of a strict theoretical convergence proof of our approach. Additionally, the SNG is order dependent to a small degree. Some areas might be unneccesarily more populated than others due to the approximative nature of the insertion criterion. Also, due to the insertion mechanism the network will adapt to noise. This is a feature shared by many VQ methods and is rather hard to overcome, especially in the case of online VQ.

Finally, three parameters have to be specified, namely the learning rate η, the repulsion rate ρ and the number of samples ν to be seen minimally by a neuron to consider its perceptive field for insertion. In our experiments we have tested various combinations of these parameters ranging from 0.005 to 0.1 for η resp. ρ and from 2 to 20 for ν . We observed that the results of the algorithm do

not depend strongly on the choice of parameters. We chose $\eta = 0.05$, $\rho = 0.005$ and $\nu = 5$ for all of our experiments.

Algorithm 1 Simplifying Neural Gas

Input: $\mathcal{X}, \eta, \rho, \nu$
Output: $\mathcal{W}, \mathcal{C}, \mathcal{E}$ with
$\mathcal{W} = \{\mathbf{w_1}, ..., \mathbf{w_k}\}$
$\mathcal{C} = \{\mathcal{C}_1, ..., \mathcal{C}_k\}$ with $\mathcal{C}_i = \{\mathbf{x_j} \in \mathcal{X} : ||\mathbf{w_i} - \mathbf{x_j}|| < ||\mathbf{w_s} - \mathbf{x_j}||, i \neq s\}$
$\mathcal{E} = \{e_{ij} : \exists \mathbf{x_s} \in \mathcal{X} s.t.$
$||\mathbf{w_i} - \mathbf{x_s}|| < ||\mathbf{w_u} - \mathbf{x_s}||, \forall u \neq i, \mathbf{w_i}, \mathbf{w_u}, \in \mathcal{W}$ *and*
$||\mathbf{w_j} - \mathbf{x_s}|| < ||\mathbf{w_v} - \mathbf{x_s}||, \forall v \neq j, \mathbf{w_j}, \mathbf{w_v}, \in \mathcal{W} \setminus \{\mathbf{w_i}\}\}$
Begin SNG
$\mathcal{C} = \emptyset$, $\mathcal{E} = \emptyset$, $\mathcal{W} = \{$ chose 2 random samples from $\mathcal{X}$ $\}$
$\forall_{1 \leq i \leq 2}\ error(\mathbf{w_i}) = 0,\ hits(\mathbf{w_i}) = 0$
for all $\mathbf{x_s} \in \mathcal{X}$ **do**
$\mathbf{w_{win}} = argmin_{w_i} ||\mathbf{w_i} - \mathbf{x_s}||, \mathbf{w_i} \in \mathcal{W}$
$\mathbf{w_{sec}} = argmin_{w_j} ||\mathbf{w_j} - \mathbf{x_s}||, \mathbf{w_j} \in \mathcal{W} \setminus \{\mathbf{w_{win}}\}$
if $hits(\mathbf{w_{win}}) > \nu \wedge mse(\mathbf{w_{win}}) < ||\mathbf{w_{win}} - \mathbf{x_s}||^2$ **then**
$\mathbf{w_{new}} = \mathbf{x_s}$
$error(\mathbf{w_{new}}) = error(\mathbf{w_{win}})$
$hits(\mathbf{w_{new}}) = hits(\mathbf{w_{win}})$
else
$\mathbf{w_{win}} = \mathbf{w_{win}} + \eta(\mathbf{x_s} - \mathbf{w_{win}})$
$error(\mathbf{w_{win}}) = error(\mathbf{w_{win}}) + ||\mathbf{w_{win}} - \mathbf{x_s}||^2$
$hits(\mathbf{w_{win}}) = hits(\mathbf{w_{win}}) + 1$
if $mse(\mathbf{w_{win}}) + mse(\mathbf{w_{sec}}) > ||\mathbf{w_{win}} - \mathbf{w_{sec}}||^2$ **then**
$\mathbf{w_{sec}} = \mathbf{w_{sec}} - \rho(\mathbf{w_{win}} - \mathbf{w_{sec}})$
end if
end if
end for
for all $\mathbf{x_s} \in \mathcal{X}$ **do**
$\mathbf{w_i} = argmin_{w_i} ||\mathbf{w_i} - \mathbf{x_s}||, \mathbf{w_i} \in \mathcal{W}$
$\mathbf{w_j} = argmin_{w_j} ||\mathbf{w_j} - \mathbf{x_s}||, \mathbf{w_j} \in \mathcal{W} \setminus \{\mathbf{w_i}\}$
$\mathcal{E} = \mathcal{E} \cup e_{ij}$, $\mathcal{C}_i = \mathcal{C}_i \cup x_s$
end for
End SNG

3 Support Vector Machine Learning

In this section we will briefly review SVM learning, first from a geometrical view followed by its formulation as an optimization problem. Our work is motivated by the geometrical interpretation of SVMs.

In supervised learning scenarios for binary classification we are typically given a set of vectors $\mathcal{X} = \{\mathbf{x_1}, ..., \mathbf{x_n}\}$ with $\mathbf{x_i} \in \mathbb{R}^d$ accompanied by a set of labels $\mathcal{Y} = \{y_1, ..., y_n\}$ with $y_i \in \{-1, 1\}$ where $1 \leq i \leq n$ and d is the number of

dimensions. This two sets combined are called the training set from which we want to learn a discriminative function $f(\mathbf{x})$ that returns the correct label for a sample $\mathbf{x}$.

SVMs arrive at the function $f(\mathbf{x})$ by constructing a hyperplane that separates the two classes in the training set with the largest margin. Depending on which side of the hyperplane $\mathbf{x}$ lies $f(\mathbf{x})$ will return the predicted label of the sample $\mathbf{x}$. The solution of the hyperplane is based on vectors in the border regions between the two classes. This vectors are called support vectors. Additionally they can cope with non-separable cases by relaxing the constraints of the optimization problem for separable cases and achieve non-linear classification behavior by incorporating the so called kernel-trick.

A large number of training samples does not contribute to the final solution of the SVM. The discriminating hyperplane is located in the border regions of the two classes. Therefore it is natural to only provide those training samples to the SVM that are within this border region and omit the rest of the samples or replace large sets of them by a representative. The solution of the SVM should not differ provided we present it all support vectors.

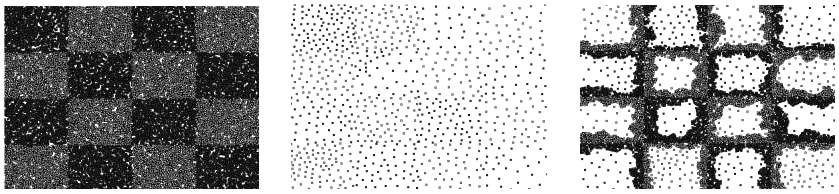
4 SNG Instance Selection

The goal of our work is it to effectively reduce the size of the training set for SVM learning. Reflecting section 3, SVM's chose only samples along the discriminative hyperplane to become support vectors. Also, the hyperplane effectively separates the convex hulls of the two classes of the training set. We therefore conclude that only areas of neighboring classes close to each other have to be considered. After discussing how to determine those areas in input space $\mathbb{R}^d$ by using neighborhood properties, we show that the derived properties also hold in feature space when using non-linear kernels.

In our approach we model each class of the training set with a separate instance of SNG, called class specific SNG in the following. We name the set of neurons grown on the positive examples $\mathcal{W}^+$ and the set of neurons generated on the negative training examples $\mathcal{W}^-$. We merge the retrieved sets W^+ and W^- into a final set $\mathcal{W}$ for which we calculate the induced Delaunay triangulation $\mathcal{E}$ in a second pass over the complete training set. As we are only interested in the triangulation induced by $\mathcal{W}$ we omit this calculation step when we first grow the class specific SNGs. While calculating the induced Delaunay triangulation we also note for each sample which neuron of the combined set $\mathcal{W}$ it was closest to resulting in a set $\mathcal{C}_i$ of training samples $\mathbf{x_i}$ for each neuron $\mathbf{w_i} \in \mathcal{W}$. We then find the set of all edges $e_{ij} \in \mathcal{E}$ connecting neurons $\mathbf{w_i}$ and $\mathbf{w_j}$ that do not originate from the same class specific SNG. Topologically those neurons and their respective sets of samples $\mathcal{C}_i$ and $\mathcal{C}_j$ make up the part of the border region between the two classes. We merge $\mathcal{C}_i$ and $\mathcal{C}_j$ to construct the final training set $\mathcal{T}$. Additionally, neurons not in the border region are also added to the training set, but their associated examples are ignored. Finally we train the SVM on the reduced training set $\mathcal{T}$. Figure 1 depicts this process on a two-dimensional

toy example. Note that our approach is currently restricted to the binary case. However, an extension to the multi-class case should be straightforward.

Due to space constraints we refrain from giving a proof that this schema holds for various kernels and refer the reader to [10]. Therein the authors show that the neighborhood properties in input space also hold when radial basis function kernels or polynomial kernels are used.



(a) original (b) approximation (c) expanded approximation

Fig. 1. (a) The checkerboard dataset. (b) A sparse representation of the checkerboard dataset produced by growing one SNG for each class. (c) The expanded representation of the checkerboard dataset. Neurons of opposite classes that are the nearest neighbor to a sample get replaced by all samples they are nearest to respectively

5 Experiments

We tested our approach on synthetic as well as standard test data sets using 10-fold cross validation for every experiment. Each data set was split into 10 stratified subsets and all classifiers were trained on the same fold combinations. We compare our approach with the complete SVM solution on the full dataset as well as to stratified random sampling with a sampling rate of 10%,30%, 50% and 70%. In addition we add a sampling rate R* for each run. R* is determined based on the number of examples selected by our SNG based approach and thus should provide a direct comparison between SNG and Random Sampling.

As evaluation criterions we use accuracy (*Accuracy*), the number of support vectors (*SVs*) generated and the time it took to train (*Total Time*) the SVM. We present the mean of these values for each data set as well as their standard deviation. Additionally we report the time taken for growing the SNGs (*SNG Time* and the percentage of the original data set taken for training (*Samples*). We exclude I/O times from all of our timings.

For all data sets we used the RBF kernel. Instead of specifying sigma we replace the term $\frac{1}{2\sigma^2}$ with γ. Table 1 gives an overview of the data sets and the parameters used for training. For the SNGs we used $\eta = 0.05$, $\rho = 0.005$ and $\nu = 5$ for all data sets.

The system was implemented and tested in Java using the Sun Server VM at version 1.6.0.7. As an SVM solver we utilized the Java implementation of

LIBSVM [11] at version 2.88. We did not enable shrinking. Our test system consisted of an eight-core Intel Xeon E5420 CPU at 2.5Ghz with 32GB of RAM using only a single core for the test runs. Data sets and source code are available for download [1].

We tested on the following data sets:

- **Adult** a census income data set with 48842 samples and 14 dimensions. We only used the training set for cross validation consisting of 32560 samples.
- **Banana** a synthetic two-dimensional dataset with a lot of noise and overlap of the class boundaries.
- **Checkboard** a synthetic two-dimensional data set as depicted in figure 1(a)
- **Pendigits** a handwritten digits data set. We tested digit 2 against the rest in our experiments.
- **USPS** a postal zip code data set consisting of images with 16x16 pixels. We tested class "0" against the rest
- **Waveform noise** a synthetic data set with 20 dimensions of real data and 20 dimension of uniform noise.

Name	Instances	Dimensions	γ	C
Adult	32560	122	0.005	100
Banana	5299	2	0.5	316
Checkerboard	100000	2	50	100
Pendigits	7493	16	0.0005	100
USPS	9298	256	2	10
Waveform	4999	40	0.05	1

Table 1. The data sets experiments were performed on and the parameters used

Table 2 presents our experimental results on the above described data sets.

On the Adult data set our approach reduced the data set to about 65% of the original size and outperformed random sampling as well as the full solution in terms of precision. In respect to the full SVM our approach was around 400 seconds faster on average, a 22% speedup. The number of support vectors roughly equals that of the full SVM solution indicating that the reduced data set contained nearly all of them. Our preprocessing approach via the SNGs takes up around 5% of the total runtime which is acceptable.

In case of the banana data set we find a similar behavior. The data set is reduced to around 55% of the original size while preserving all support vectors. We slightly outperform the full SVM in terms of precision while being around 30% faster. Again the preprocessing only takes roughly 5% of the total runtime.

[1] `http://en.know-center.at/forschung/knowledge_relationship_discovery/downloads_demos`

	Accuracy	*Total Time*	*SVs*	*Samples (%)*	*SNG Time*
Adult					
R10	0.8414 ± 0.0059	7.3610	1061 ± 35	10	-
R30	0.8432 ± 0.0064	61.3676	3161 ± 38	30	-
R50	0.8461 ± 0.0058	243.4072	5222 ± 50	50	-
R70	0.8474 ± 0.0058	545.9942	7234 ± 54	70	-
R*	0.8463 ± 0.0060	593.8760	7430 ± 126	65,1296	-
SSVM	$\mathbf{0.8482 \pm 0.0059}$	857.0005	10115 ± 39	65,1296	47,2251
SVM	0.8479 ± 0.0060	1245.3783	10222 ± 28	100	-
Banana					
R10	0.8883 ± 0.0078	0.0506	107 ± 11	10	-
R30	0.9008 ± 0.0081	0.1826	312 ± 14	30	-
R50	0.9038 ± 0.0092	0.4291	527 ± 19	50	-
R70	0.9038 ± 0.0080	0.8235	730 ± 21	70	-
R*	0.9034 ± 0.0107	0.6363	639 ± 32	55,1293	-
SSVM	$\mathbf{0.9060 \pm 0.0074}$	1.0889	1045 ± 8	55,1293	0,0622
SVM	0.9059 ± 0.0073	1.5763	1046 ± 8	100	-
Checkerboard					
R10	0.9912 ± 0.0007	0.9182	441 ± 11	10	-
R30	0.9955 ± 0.0007	4.7641	933 ± 21	30	-
R50	0.9970 ± 0.0006	13.1338	1348 ± 10	50	-
R70	0.9976 ± 0.0006	25.4859	1687 ± 16	70	-
R*	0.9960 ± 0.0010	6.0691	1000 ± 28	29,6086	-
SSVM	$\mathbf{0.9980 \pm 0.0003}$	17.5875	2156 ± 5	29,6086	4,0748
SVM	0.9980 ± 0.0004	50.6111	2156 ± 6	100	-
Pendigits					
R10	0.9939 ± 0.0052	0.0875	372 ± 11	10	-
R30	0.9975 ± 0.0028	0.3271	704 ± 16	30	-
R50	0.9983 ± 0.0013	0.6674	878 ± 16	50	-
R70	0.9981 ± 0.0015	1.0493	1012 ± 32	70	-
R*	0.9969 ± 0.0019	0.1790	564 ± 37	18,2717	-
SSVM	0.9983 ± 0.0012	0.6843	606 ± 38	18,2717	0,4887
SVM	$\mathbf{0.9984 \pm 0.0013}$	1.8018	1151 ± 11	100	-
USPS					
R10	0.9922 ± 0.0028	0.2732	90 ± 6	10	-
R30	0.9949 ± 0.0025	1.3710	150 ± 12	30	-
R50	0.9963 ± 0.0022	2.8140	183 ± 8	50	-
R70	0.9959 ± 0.0024	4.6877	217 ± 8	70	-
R*	0.9940 ± 0.0029	0.8177	121 ± 5	19,6429	-
SSVM	0.9957 ± 0.0017	4.8140	207 ± 9	19,6429	3,3970
SVM	$\mathbf{0.9964 \pm 0.0017}$	9.1343	261 ± 7	100	-
Waveform					
R10	0.8499 ± 0.0154	0.0771	421 ± 3	10	-
R30	0.8868 ± 0.0139	0.5651	1057 ± 9	30	-
R50	0.8928 ± 0.0106	1.4878	1561 ± 16	50	-
R70	0.8970 ± 0.0079	2.9207	2023 ± 11	70	-
R*	0.8986 ± 0.0107	4.2489	2345 ± 86	76,4933	-
SSVM	$\mathbf{0.9014 \pm 0.0085}$	5.3144	2466 ± 28	76,4933	0,5186
SVM	0.8992 ± 0.0089	6.6437	2642 ± 8	100	-

Table 2. Results on various data sets. R10 to R70 are the results for random sampling from 10% to 70%, R* is the result for random sampling with the same number of samples as the SNG sampling. SSVM is the result for the SNG based sampling approach and SVM is the full solution.

For the checkerboard data set we are again on par with the full SVM solution. The data set is reduced to around 30% of its original size and still preserves all of the support vectors. The runtime is reduced to roughly a third of that of the full SVM solution. Interestingly 70% random sampling does not comes close to this result and has a bigger total runtime than our approach. The preprocessing stage does take up around 23% of the total runtime which is due to the SVM converging faster as the data set is more easily seperable than the adult or banana data set.

On the Pendigits data set we can observe a very interesting behavior. Our approach comes in second in terms of precision, however the the difference is nearly neglect able. The data set is reduced to 18% of its original size yielding a much sparser solution than the full SVM. The number of support vectors is nearly halfed while the runtime is around a third of that of the full SVM. Interestingly randomly sampling 50% of the data set yielded a comparable precision but a more complex model in terms of the number of support vectors. The preprocessing stage takes a lot of the total runtime, in this case 71%. This is again due to the SVM being able to converge rapidly as the classes are easily seperable.

Our approach performed the worst on the USPS data set. Even tough it only took half of the time it took to solve the full SVM problem it was outperformed by random sampling at 50% and 70% both in terms of runtime and precision. The SNG takes up around 70% of the total runtime as calculating the euclidian distance in such highdimensional spaces is expensive. We are however still outperforming random sampling at 10% and 30% in terms of precision as well as random sampling with the same number of samples as our approach.

Finally on the Waveform data set our approach performs again better than the full SVM solution both in terms of precision and runtime. This data set is very noisy so the SNG approach only reduces the data set by around 24%. The number of support vectors is slightly less than that of the worse performing full SVM. Random sampling does not perform very well on this data set overall. As with the adult data set our approach can cope very well with the noise and even produces a solution outperforming the full SVM. The runtime is reduced by around a sixth of that of the full SVM which is not enormous but acceptable given that the solution performs better.

6 Conclusion & Future Work

We could demonstrate that our approach reduces the runtime for solving SVM problems while maintaining or improving the classification performance of the resulting classifier. Especially noisy data sets are handled very well by our approach. The SNG has a faster training time than the full SVM while achieving similar accuracy levels. Random Sampling is usually faster in training, but the sampling rate has to be known a-priori in order to reliably achieve a similar accuracy than the full SVM. Given the results on the USPS data set we consider adapting the SNG to remove the costly distance measurements which increase the runtime for high dimensional problems. One approach would be similar in

spirit to spherical k-means clustering where the dot product is used instead of the euclidean distance. This would also open up the domain of text classification to our approach. To further reduce the number of samples provided to the SVM we could try to exploit the induced Delaunay triangulation better. Instead of expanding the neuron by all its associated samples we could further divide this set of samples. However, this adds additional runtime cost which would have to be balanced. The SNG itself could also be used for other machine learning related tasks. We will investigate its use as a fast partitioning method to speed up k-NN classification. This approach would fall in the domain of approximative k-NN search. Hierarchies of SNGs could further speed up the the process. Yet another use for the SNG could be in clustering applications and approximate singular value decomposition which also suffers from a high runtime complexity.

Acknowledgement

The Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Ministry of Transport, Innovation and Technology, the Austrian Ministry of Economics and Labor and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

References

1. V. N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.
2. H. Shin and S. Cho. Fast pattern selection for support vector classifiers. In Kyu-Young Whang, Jongwoo Jeon, Kyuseok Shim, and Jaideep Srivastava, editors, *PAKDD*, volume 2637 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2003.
3. H. Yu, J. Yang, and J. Han. Classifying large data sets using svms with hierarchical clusters, 2003.
4. D. Boley and D. Cao. Training support vector machine using adaptive clustering. Proceedings of the SIAM International Conference on Data Mining, pages 126–137, April 2004.
5. T. Kohonen. Learning vector quantization. *Neural Networks*, 1, Supplement 1:3–16, 1988.
6. B. Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.
7. N. A. Syed, H. Liu, and K. K. Sung. A study of support vectors on model independent example selection. In *KDD*, pages 272–276, 1999.
8. A. Gersho and R. M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, 1992.
9. B. Fritzke. A growing neural gas network learns topologies. In *Neural Information Processing Systems*, pages 625–632, 1994.
10. H. Shin and S. Cho. Invariance of neighborhood relation under input space to feature space mapping. *Pattern Recognition Letters*, 26(6):707–718, 2005.
11. C. C. Chang and C. J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.