Support Vector Machines

Halldor Isak Gylfason

Department of EECS
University of California, Berkeley
Berkeley, CA 94720-1770
halldor at eecs.berkeley.edu

December 11, 2005

Abstract

In this report we present an overview of Support Vector Machines and their implementations. Support Vector Machines (SVMs) are an effective mechanism for binary classification that have good generalization properties. SVMs find the optimal linear separator, the maximal margin hyperplane, in a high dimensional feature space – a different approach from mechanisms like Neural Networks that build a highly non-linear separator in the original space.

The objective of this work was to give the author broad overview of Machine Learning tools not directly covered in CS281A,– a graduate course in probabilistic graphical models at UC Berkeley. This included topics such as computational learning theory, non-parametric methods, support vector machines, optimization and, finally, practical implementations of these concepts .

Contents

1 Introduction

Classification is an important statistical problem, where the objective is to learn a discrete-valued function based on a limited number of examples. Consider the task of learning species of fish, based on various measurements, such as height, weight and color. The ichthyologist may have done various measurements on some set of fish, observed their species, and now he wishes to find a model that will correctly classify *unseen* examples. A number of books on Machine Learning give good introductions to various methods of classification [RN02, Mit97, HTF01, Jor03]

Early methods for classification include decision trees, and neural networks. Decision trees are a simple mechanism, with relatively good generalization properties, but lack expressiveness to represent certain classes of functions. As one example, it is difficult for a decision tree to represent the **majority function** without an exponential increase in the size of the decision tree. Neural networks, on the other hand, are a very expressive mechanism that can represent arbitrary non-linear functions. Thus the risk of overfitting is high, and the training of the neural network has to be conducted carefully to limit the complexity of the resulting hypothesis.

The quality of a particular classification method depends on how well it generalizes to unseen examples. If the mechanism is too *expressive* we run the risk of *overfitting* the data, essentially memorizing the training set. Computational Learning Theory aims to characterize under what conditions a particular learning algorithm is assured of learning successfully. Within the probably-approximately-correct framework (PAC) the objective is to ascertain with some probability $(1 - \delta)$ that the error of the hypothesis learned will be within ϵ. The definition of an error of a hypothesis h with respect to the true function f, given a distribution D, from which the examples are drawn is defined as:

$$\text{error}(h) = P(h(x) \neq f(x) |\ x \text{ drawn from} D) \tag{1}$$

It can be shown [RN02] that in order to reduce the probability of learning a bad hypothesis (one where the error is greater than ϵ) below a small number δ, the number of required samples, N, from the distribution D has to satisfy the inequality

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right) \tag{2}$$

where $|H|$ is the size of the hypothesis space. This number is called the **sample complexity** of the hypothesis space. We see that the larger the hypothesis space, the more examples we need to use for training, or equivalently, the greater the risk is to learn a bad hypothesis. Thus an expressive class of model can induce a greater error than a simple one!

Vapnik and Chervonenkis [VC71] developed an independent approach with their work on **uniform convergence theory**. Their results extends to the analysis of sets of hypotheses that are infinite (e.g. real valued network weights in a neural network), by use of the **VC dimension** [Vap99] – a measure roughly analogous to $|H|$. The VC dimension of a function class H, is the largest size of a set of points that can be *shattered* by H. A set is said to be shattered by H, if for every assignment of (boolean) classes to the points in the set, there exists a $h \in H$ which correctly classifies the set. Thus the VC dimension measures the expressiveness of the class. In [Vap99] it is shown that if a hypothesis is consistent with the training examples , then the error bound is with probability $(1 - \delta)$:

$$\text{error}_D(h) \leq \sqrt{\frac{V(\log(2l/V) + 1) - \log(\delta/4)}{l}} \tag{3}$$

where V is the VC dimension and l is the number of training examples. This bound is called the VC confidence, and we see that decreasing the VC dimension leads to a better error bound. This leads leads to a principle coined *Structural Risk Minimization* [VC71], where the objective is to find a machine that minimizes the sum of empirical risk (the error on the training set) and the VC confidence. Thus, among machines that have a low

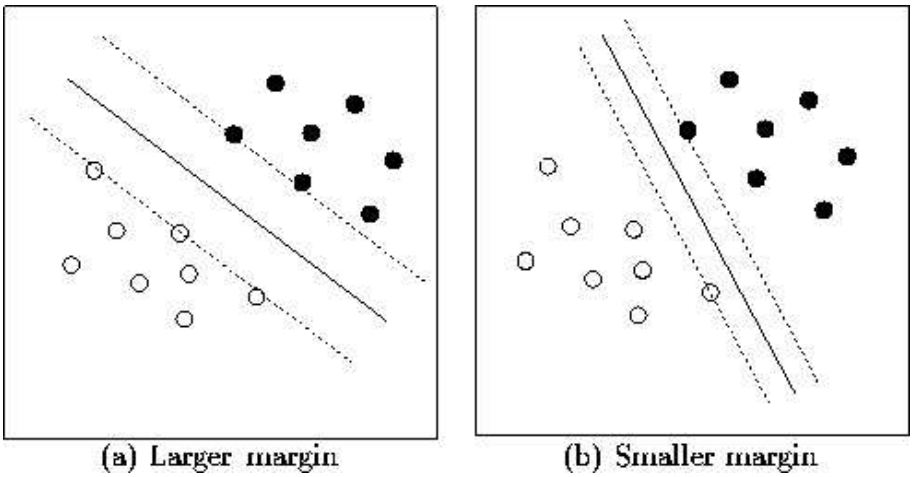

Figure 1: The classifier in (a) has wider margin than the one in (b)

error on the training set we should choose the one with the lowest VC dimension. Support Vector Machines are particularly well suited to this purpose, since their VC dimension scales linearly with the dimension of the input space.

Further results from Computational Learning theory show that when the margin between classes is high the error bound is improved. This makes intuitively sense, as a learning machine that maximizes this margin (see figure 1) should generally have a lower error rate on unseen examples near the margin. This is precisely what Support Vector Machines accomplish and thus owe their existence to results from Computational Learning Theory on how to build classifier with good generalization properties.

2 The basics of Support Vector Machines

2.1 Maximal Margin Classifier

In this section we review the basics of Support Vector Machines. There are several introductory books [CST00], and articles [CV95, Bur98, K. 01] available for the interested reader – here we merely focus on the main ideas.

The simplest case of a Support Vector Machine arises when the data is linearly separable in the input space. As previously mentioned the SVM finds the maximal margin hyperplane, which turns out to be a convex optimization problem [CST00, BV04]. Given a linearly separable training set $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)$, the hyperplane $(\mathbf{w}, b)$ that solves the optimization problem

$$\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{4}$$

$$\text{subject to} \quad y_i\Big(\mathbf{w}^T\mathbf{x}_i + b\Big) \geq 1 \quad \forall i = 1, \ldots, M \tag{5}$$

realizes the maximal margin hyperplane with geometric margin $\gamma = \frac{1}{||\mathbf{w}||}$.

This optimization problem is solved by transforming it into the equivalent Lagrangian dual problem [BV04] . The primal form of the Lagrangian is:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{M} \alpha_i[y_i(\Big(\mathbf{w}^T\mathbf{x}_i + b\Big) - 1] \tag{6}$$

where we have added the Lagrange constraint multipliers. Since this is a convex optimization problem we can differentiate to find the stationary point, a necessary and sufficient condition for the solution:

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=i}^{M} y_i\alpha_i\mathbf{x}_i = 0 \tag{7}$$

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = \sum_{i=i}^{M} y_i\alpha_i = 0 \tag{8}$$

We need to resubstitute these relations:

$$\mathbf{w} = \sum_{i=i}^{M} y_i\alpha_i\mathbf{x}_i \tag{9}$$

$$0 = \sum_{i=i}^{M} y_i\alpha_i \tag{10}$$

into the primal t to obtain the dual:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\Big(\sum_{i=i}^{M} y_i\alpha_i\mathbf{x}_i\Big)^T \sum_{j=i}^{M} y_j\alpha_j\mathbf{x}_j - \sum_{i=1}^{M} \alpha_i y_i\Big(\sum_{j=i}^{M} y_j\alpha_j\mathbf{x}_j\Big)^T\mathbf{x}_i + \sum_{i=1}^{M} \alpha_i \tag{11}$$

$$= \frac{1}{2}\sum_{i,j=i}^{M} y_i y_j\alpha_i\alpha_j\mathbf{x}_i^T\mathbf{x}_j - \sum_{i,j=1}^{M} \alpha_i\alpha_j y_i y_j\mathbf{x}_i^T\mathbf{x}_j + \sum_{i=1}^{M} \alpha_i \tag{12}$$

$$= \sum_{i=1}^{M} \alpha_i - \frac{1}{2}\sum_{i,j=i}^{M} y_i y_j\alpha_i\alpha_j\mathbf{x}_i^T\mathbf{x}_j \tag{13}$$

Consequently, we have the equivalent (dual) optimization problem:

$$\text{maximize}_\alpha \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=i}^{M} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j \tag{14}$$

$$\text{subject to} \quad \sum_{i=i}^{M} y_i \alpha_i = 0, \tag{15}$$

$$\text{and} \quad \alpha_i \geq 0 \tag{16}$$

This is the form typically used in Support Vector Machines. To classify a new point we compute the sign of $f(x) = \mathbf{w}\mathbf{x} + b^* = \sum_{i=i}^{M} y_i \alpha_i \mathbf{x}_i \mathbf{x} + b^*$ and if it is positive we return the class 1; otherwise we return -1.

A key observation is that the data only enters in the form of dot products, and thus in fact we do not need to use the original vectors – given all the dot products, the problem is fully specified. This leads to the *kernel trick*, where we map the original vectors into a high-dimensional *feature space*, and compute the dot product in that space. Since the only thing we need is the dot product we do not need to perform the mapping; we only need a formula for the dot product, based on the original vectors. A *kernel* is precisely this formula, but more formally it is defined as a function K such that for all $\mathbf{x}, \mathbf{z} \in X$, $K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$, where ϕ is a mapping from X to an (inner product) feature space F. Using the kernel trick the optimization problem becomes:

$$\text{maximize}_\alpha \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=i}^{M} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{17}$$

$$\text{subject to} \quad \sum_{i=i}^{M} y_i \alpha_i = 0, \tag{18}$$

$$\text{and} \quad \alpha_i \geq 0 \tag{19}$$

with a decision function:

$$f(x) = \sum_{i=i}^{M} y_i \alpha_i K(\mathbf{x}_i \mathbf{x}) + b^* \tag{20}$$

Popular kernels include the polynomial kernel:

$$K(\mathbf{x}, \mathbf{z}) = \Big(\langle \mathbf{x} \cdot \mathbf{z} \rangle + c \Big)^d \tag{21}$$

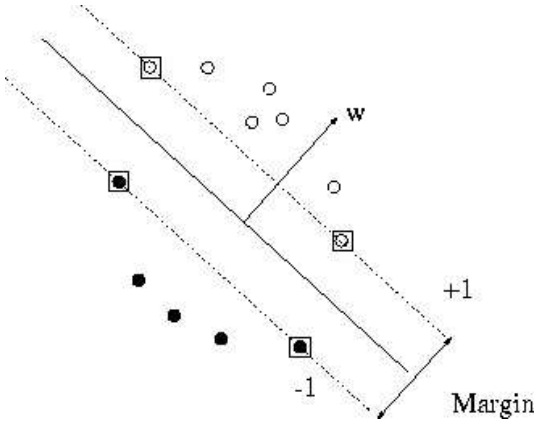

Figure 2: Support Vectors are in the rectangles

whose features are all the monomials of degree d, and the Gaussian kernel:

$$K(\mathbf{x}, \mathbf{z}) = \exp\Big(- \frac{\| \mathbf{x} - \mathbf{z} \|^2}{\sigma^2} \Big) \tag{22}$$

Both of these kernel include parameters that control how powerful the kernel is. In the case of the polynomial kernel, the higher the d, the more powerful the kernel becomes (higher VC dimension), and more complex decision surfaces can be constructed. *Model selection* in SVMs thus involve choosing the form of the kernel, and parameters for the chosen kernel.

Another key observation is that the Karush-Kuhn-Tucker conditions for optimality of the convex optimization problem [BV04] state that:

$$\alpha_i(y_i f(\mathbf{x}_i) - 1) = 0 \tag{23}$$

$$\tag{24}$$

We observe that the Lagrange multiplier α_i is non-zero if and only if the the training point $\mathbf{x}_i$ is on the margin. These training points are called the Support Vectors (see figure 2). and in order to classify an unseen instance we only need to store the support vectors. This is generally referred to as the *sparsity property* of the SVMs, and is clearly a desirable property if we want to work with huge datasets.

2.2 Soft margin classifier

The maximum margin classifier always produces a perfectly consistent hypothesis. However, in real world data there may be errors, and we do not want few outliers to affect the resulting hypothesis too much. In fact, a single point might force us to use a very powerful kernel, which in most cases would result in worse generalization properties. Thus, we want to accommodate for errors, and allow a few points to be misclassified. This leads to the soft margin classifier:

$$\text{minimize}_{\mathbf{w},b,\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{M}\xi_i \tag{25}$$

$$\text{subject to} \quad y_i\Big(\mathbf{w}^T\phi(\mathbf{x}_i) + b\Big) \geq 1 - \xi_i \quad \forall i = 1, \ldots, M \tag{26}$$

The ξ are slack variables that relax the hard-margin constraints (see figure 3), and C determines the weight of a non-zero value of ξ and represents the trade-off in misclassifying one points, for the benefits of others. When C approaches ∞, we are back into the familiar terrain of the hard margin classifier.

The corresponding dual form is:

$$\text{maximize}_{\alpha} \sum_{i=1}^{M}\alpha_i - \frac{1}{2}\sum_{i,j=i}^{M} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{27}$$

$$\text{subject to} \quad \sum_{i=i}^{M} y_i\alpha_i = 0, \tag{28}$$

$$\text{and} \qquad C \geq \alpha_i \geq 0 \tag{29}$$

The only difference here are the constraints that the α_i must not exceed C.

2.3 Multiclass classification

The basic SVM performs a binary classification, whereas many important problems involve multiple classes (e.g. handwritten digit recognition). The classical approach is to train k different binary classifiers, where each classifier recognizes one of k classes. This method is often referred to as the *one-versus-rest* method. A training point is assigned to the class which has the largest distance to the hyperplane in the positive direction of the associated classifier. Methods exist to optimize the constructions of these classifiers [WW99]

Alternative method is the tree based method, which builds all pairwise classifiers and uses a tree-based voting system to assign a training point to a class

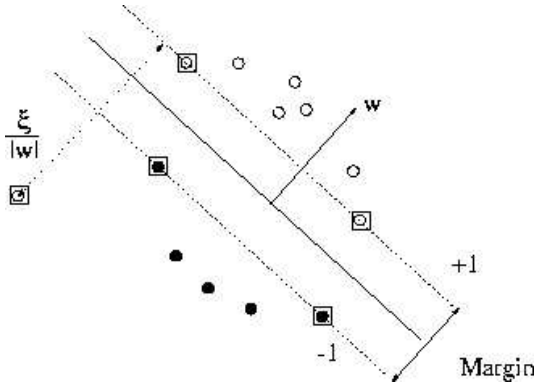

Figure 3: Soft Margin

2.4 Uses of SVMs

SVMs have been used for a wide variety of tasks. Among the first was optical character recognition [CV95, DS02], and the result of that was quite accurate, when compared to state-of-the-art results using neural networks. Another area where SVMs have been performing well is text categorization [Dum98] – a very high dimensional problem that SVMs tend to handle well. Further applications include face recognition [HHP01], and DNA analysis

3 Implementation of SVMs

3.1 Classical Optimization methods

A training of SVM requires solving the optimization problem in equation (27). An equivalent formula, using matrix notation is:

$$\text{minimize}_\alpha \quad \frac{1}{2}\alpha^T Q\alpha - \alpha^T \mathbf{1} \tag{30}$$

$$\text{subject to} \quad \alpha^T \mathbf{y} = 0, \tag{31}$$

$$\text{and} \quad \mathbf{0} \leq \alpha \leq C\mathbf{1} \tag{32}$$

where $(Q)_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$. This is a convex optimization problem in standard form, for which there exists a large variety of methods in the literature [BV04]. Moreover, a number of standard software optimization packages are available for these problems [Van] . Some of the standard methods for doing convex optimization include the Newton method, conjugate gradient and primal-dual interior-point methods.

The problem is that these standard approaches require the storage (or recomputation) of the full kernel matrix Q, which becomes prohibitively expensive as the number of training

examples grows – the size of the matrix is quadratic in the sample size. Fortunately, there exist several algorithms that exploit the structure of the SVM optimization problem, and have fast convergence and small memory requirements, even on large problems.

3.2 Optimization methods for SVMs

The solution to the SVM optimization problem is sparse, i.e. a considerable number of the α_i is 0, and the associated constraints inactive. If we would know in advance which of the constraints were inactive in the final solution we could drop them, and solve the reduced problem. This would only involve the kernel matrix of the support vectors, which in many cases will be a lot smaller than the full matrix. Of course we do not know the set of support vectors in advance, but we can make an initial guess of the active constraints – the *active set* – and iteratively refine it until we satisfy the KKT conditions. This methodology is called *chunking* and increases the size of the problems that can be solved by SVMs, but is still constrained by the fact that the matrix is quadratic in the number of support vectors.

Another category of methods, *Decomposition methods* [Joa98] are similar to chunking, except for the fact that the size of the subproblems is fixed. This is based on the observation that a sequence of Quadratic Problems that always contains at least one constraint that violates the KKT conditions will converge to the optimal solution. Since the size is fixed, every time a new point is added to the data set, another one has to be removed. Moreover, we can now train arbitrarily large data sets, although the convergence will of course be slower for large data sets. In practice, it is important to have efficient and effective methods for selecting the working set. SVM^{light} [Joa] is an open-source implementation of these concepts

Sequential Minimal Optimization (SMO) [Pla98] takes the idea of decomposition to the extreme, by using an active set of size 2. This is also the minimum size possible, since the constraint $\alpha^T \mathbf{y} = 0$ implies that if we change one α_i we have to modify another to enforce the constraint. This relaxed optimization problem, where only two variables are allowed to change can be solved analytically, and thus no numerical optimization needed. It remains important to select a good pair of variables – one that causes a large decrease in the objective function. [Pla98] contains such heuristics which are based on finding a point that violates the KKT conditions, and choose the second point to maximize the change in the objective function.

4 SvmPy - SVM library for SciPy

Scientific Python [Oli05] (SciPy) is an open source library of scientific tools for the Python programming language. SciPy extends Python with two fundamental objects, an N-dimensional array object (*ndarray*), and a universal function object (*ufunc*) which enables matrix oriented computation such R and Matlab provide. The benefits of using SciPy, when compared to R and Matlab, is the Python programming language and all of its associated tools. R and Matlab are for the most part prototype environments that are excellent when experimenting

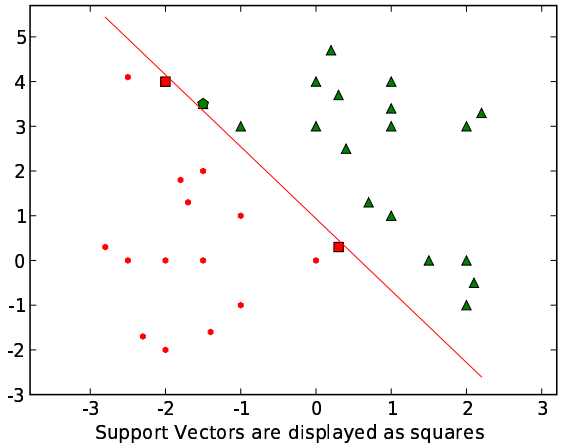

Figure 4: Normal Dot product kernel

with new ideas, but fall short when building sophisticated production systems. Python, on the other hand, provides all facilities of modern programming languages, and the difference is especially noticeable in the realms of data structures. The disadvantage of using SciPy is the lack of available libraries. Indeed, there is no available implementation of Support Vector Machines, so as part of this work we started an implementation of a SVM library for SciPy – SvmPy.

Currently we have implemented two algorithms for solving the QP – one that uses the Cvxopt optimization library for Python [Van], and one that uses the Newton method. However, the Newton method is not completed as we have yet to handle the cases when the KKT matrix is singular. As these two methods can only handle simple problems, it remains as future work to implement more advanced algorithms, such as John's Platt SMO, or to encapsulate an existing package such as SVM^{light} .

To get an intuition for how the decision surface depends on the chosen kernel we tested SvmPy on a simple two dimensional dataset; the results are displayed in figures 4, 5, and 6. We observe that the complexity of the decision surface, and the number of support vectors, grows as the kernel becomes more complex.

5 SVM for handwritten digit recognition

Handwritten digit (or character) recognition is one of the classical pattern recognition problems. There exist several publicly available data sets, which facilitates comparison between different algorithms. We took one of those dataset, MNIST [mni], for the purpose of testing SVMs. MNIST consists of two datasets; a training set of 60.000 examples, and a test set of 10.000 examples. Each example is a 28 by 28 array of greyscale pixel values that range from 0 to 255, where 0 is white and 255 black

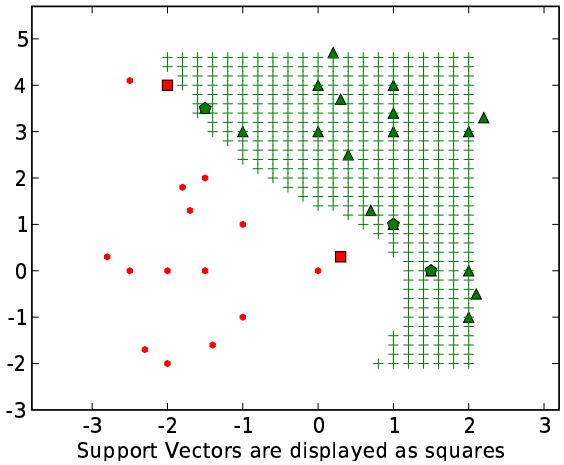

Figure 5: Polynomial kernel with $d = 3$

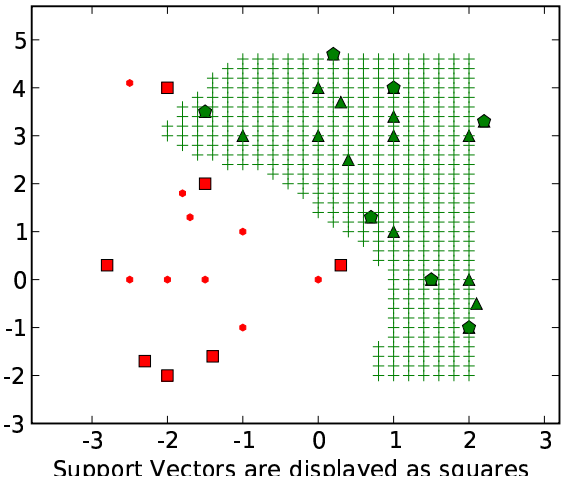

Figure 6: Gaussian kernel with $\sigma = 1.0$

Kernel	parameters	Training Set Error	Test Set Error
Linear (dot product)		7.1%	15.47%
Polynomial	$d = 3$	0%	9.76%
Polynomial	$d = 6$	0 %	13.11 %
Gaussian	$\sigma = 1$	0 %	14.36 %
Gaussian	$\sigma = 3$	0 %	14.11 %

Table 1: SVM performance on the MNIST data set, with $C = 0.01$

As our implementation cannot currently handle the size of this dataset, we ran SVM^{light} on it. One interesting feature of SVM^{light} is that it uses a sparse representation of the input set, and the resulting model (i.e. the support vectors and the intercept). This is very convenient for MNIST since a lot of the pixels have value 0. SVM^{light} is a classical SVM, so it performs only binary classification. Recently Thorsten Joachims has released SVM^{struct} which allows for prediction of structured output, and a specific instantiation of that, SVM^{multiclass} implements multiclass classification.

Because of lack of CPU cycles, we only trained the SVM on a portion of the training set, namely 1.000 examples. It took around 2 minutes to train the SVM with the parameter C set to 0.01. Increasing the value of C to 1 causes the training time of SVM^{multiclass} to increase to up to 1 hour! The training of 30.000 examples with $C = 0.01$ takes around 30 minutes. This, in general, suggests that it would be worthwhile to do research on how to reduce the training time even more than has currently be done.

We experimented with a few different configurations of the kernels as table 1 displays. These results must be taken with a grain of salt, since we are using only a limited portion of the complete training set. Nevertheless, we observe that the best result on the test set is gained with the relatively simple polynomial kernel of degree 3. When using degree 6 polynomial kernel, or one of the Gaussian kernels we have overfitted the data, which leads to worse results on the test set.

6 Conclusions

Support Vector Machines are an effective mechanism for classification, leading to classifiers with good generalization properties, ascertained by results from Computational Learning Theory. SVMs implicitly map the input data into a high-dimensional feature space, where the maximal margin hyperplane is constructed. Thus, complex decision surfaces can be constructed, as the hyperplane in the original space is non-linear, while at the same time lowering the risk of overfitting.

The main challenge in SVMs is to find an effective way of solving the convex optimization problem. Several SVM-specific approaches have been devised; yet there is room for considerable improvement, especially in an on-line setting. We have started implementation of a SVM library for Scientific Python, but have yet to implement one of those effective

algorithms.

References

[Bur98] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimiziation*. Cambridge University Press, 2004.

[CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

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

[DS02] Dennis Decoste and Bernhard Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190, 2002.

[Dum98] S. Dumais. Using svms for text categorization. *IEEE Intelligent Systems*, 13(4), 1998.

[HHP01] B. Heisele, P. Ho, and T. Poggio. Face recognition with support vector machines: Global versus component-based approach. *IEEE International Conference on ICCV*, pages 688–694, 2001.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[Joa] T. Joachims. SVM-Light Support Vector Machine. http://svmlight.joachims.org/.

[Joa98] T. Joachims. Making large-scale support vector machine learning practical. In A. Smola B. Schoelkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[Jor03] Michael I. Jordan. *An Introduction to probabilistic graphical models*. Book in preparation, 2003.

[K. 01] K. Mueller and S. Mika and G Raetsch and K. Tsuda and B. Schlkopf. An Introduction to Kernel-Based Learning Algorithms. *IEEE Transactions on Neural Networks*, 2(12):181–201, 2001.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[mni] MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[Oli05] Travis E. Oliphant. *Guide to SciPy: Core System*. 2005.

[Pla98] J. Platt. Fast training of support vector machines using sequential minimal optimization. In A. Smola B. Schoelkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[RN02] Stuart Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (2nd Edition)*. Prentice Hall, 2002.

[Van] Lieven Vandenberghe. CVXOPT: A Python Package for Convex Optimization. http://www.ee.ucla.edu/ vandenbe/cvxopt/cvxopt.html/.

[Vap99] V. Vapnik. *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science. Springer, 1999.

[VC71] Vapnik and Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications. 16, 264-280*, 1971.

[WW99] J. Weston and C. Watkings. Support vector machines for multi-class pattern recognition. *Proceedings of the 6th European Symposium on Artificial Neural Networks (ESANN)*, 1999.