Investigation of the performance of current Support Vector Machine software

James Billett
18538363

Supervisors: Dr. David Albrecht,
Dr Peter Tischer
# Contents

1 **Introduction** .................................................. 1

2 **Support Vector Machines** ..................................... 2
   2.1 Linear SVMs .................................................. 2
      2.1.1 Linearly Separable data .................................. 2
      2.1.2 Non-Linearly separable data ................................ 2
   2.2 Non-Linear SVMs .............................................. 3
   2.3 SVM Applications ............................................. 3

3 **SVM Software** .................................................. 5
   3.1 SVM\textsuperscript{light} ..................................... 5
      3.1.1 Input and Outputs .......................................... 5
   3.2 LIBSVM .......................................................... 7
      3.2.1 Inputs and Outputs .......................................... 7
   3.3 Other SVM Software ............................................ 8
   3.4 Wrapper Software ............................................... 8

4 **SVM Theory** .................................................... 9
   4.1 Normalisation of input data ................................... 9
   4.2 Trimming of input data ........................................ 9
   4.3 SVM starting conditions ....................................... 10

5 **Conclusion** .................................................... 11
1 Introduction

What is a Support Vector Machine? “Support Vector Machines (SVMs) are a set of related supervised learning methods, applicable to both classification and regression” [Wik05]. In addition, a Support Vector Machine may be defined as “a classification method that determines that maximum-margin hyperplane” [Qui]. In terms of their area of computing, Support Vector Machines are competing with Neural Networks as tools for solving pattern recognition problems.

In the case of basic linear classification a Support Vector Machine creates a maximum-margin hyperplane that lies in a transformed input space. Given binary choice training examples (labeled either ‘yes’ or ‘no’), a maximum-margin hyperplane divides the ‘yes’ and ‘no’ examples, such that the distance from the closest examples, i.e. the margin, to the hyperplane is maximized.

When considering non-linear classification the resulting Support Vector Machine algorithm is formally similar, except that every dot product is replaced by a non-linear kernel function. This causes the linear algorithm to operate in a different space and thus fit the maximum margin hyperplane in that space. This space is of constructed features being a non-linear map from the original input space and usually of much higher dimensionality than the original input space. [Wik05] Each of these cases is explained in more detail later.

It should be noted at this point that in the linear case, we have separable data and there is an accepted view of what constitutes a Support Vector Machine however, in the non-linear case we have non-separable data and the Support Vector Machine is therefore a combination of a maximum margin and a penalty for misclassification. With this in mind, there is no consensus as to what constitutes an SVM in the non-linear case.

There already exist many software implementations of Support Vector Machines. The objective of this project is to utilise these implementations in the creation of an environment for conducting experiments on the software and indeed the Support Vector Machine technique itself. This objective will be accomplished through the development of both ‘pre’ and ‘post’ SVM wrapper software. The intended functionality of this software is to allow comparisons between various Support Vector Machine implementations and to serve as a platform from which to implement ideas for improving their performance. It is envisaged that while the initial support vector machine implementations used will be those currently available, future implementations could be added to the wrapper software and thus compared against existing standards.
Machine Learning is a solution to the task of searching a space of potential hypotheses, usually quite large in size, to determine such a hypotheses that will best fit the data and any prior knowledge. This data may be either labeled or unlabeled, the former leads to the problem of supervised learning in which if the data labels are categorical then the problem is one of classification.[BB01]

Originally developed by Vladimir Vapnik in 1995 [Vap95], Support Vector Machines (SVMs) are a relatively new technique for machine learning that can be used for supervised classification which has gained both popularity and momentum. Support Vector Machines learn their classification via a training data set of the form

\[ \{ \mathbf{x}_i, y_i \}, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, 1\}, i = 1 \ldots l \]

The \( l \) instances of the training data each contain an \( n \)-dimensional vector \( \mathbf{x} \) that describes the features of that instance and a label \( y \) that classifies the instance as belonging to one of two categories, -1 or 1. Following sufficient training examples the Support Vector Machine is then able to classify previously unseen examples (instances), i.e. those with no given label, into one of the two categories.[Cla03]

This can also be viewed from a geometric standpoint. The Support Vector Machine attempts to construct a decision surface that dissects \( \mathbb{R}^n \) such that all instances belonging to the positive class appear on one side of the surface with all instances belonging to the negative class appearing on the other. This approach is similar to a number of others in the classification field, where SVMs differ is in their implementation of such an approach.

### 2.1 Linear SVMs

#### 2.1.1 Linearly Separable data

In the case where the two classes are linearly separable the task is one of finding a hyperplane that maximises the distance between the instance or training point in the positive class that is closest in distance to the negative class and conversely the instance or training point in the negative class that is closest in distance to the positive class. The formula being [OFG97b]

\[ \min_{\mathbf{w}, b} \Phi(\mathbf{w}) = \frac{1}{2} \| \mathbf{w} \|^2 \]

This leads a hyperplane that defines the decision surface for data instances (\( \mathbf{x} \)), given by:

\[ \mathbf{w} \cdot \mathbf{x} + b = 0, \quad \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R} \]

where \( \mathbf{w} \) is the normal of the hyperplane and thus gives its orientation and \( b \) gives the distance to the origin (\( b/\| \mathbf{w} \| \)).

The distance between the respective classes is termed the “margin” of the separating hyperplane and hence the procedure is maximising the margin. This procedure leads to the greatest ability to generalise, where the instances chosen from both the positive and negative class are deemed the “support vectors”.

#### 2.1.2 Non-Linearily separable data

When applying the Support Vector Machine to any real data it is rare that the entire set will be linearly separable. It is inevitable that some data points will end up on the wrong side of the hyperplane’s decision surface. With this in mind, the Support Vector Machine
technique seeks to simultaneously minimise the misclassification’s while still maximising the margin.\cite{BC03} To this end a penalty is introduced. The formula being \cite{OFG97b}:

$$\min_{w, b, \alpha} \quad \Phi(w, \alpha) = \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{l} \xi_i$$

s.t. \quad y_i(w \cdot \vec{x}_i + b) \geq 1 - \xi_i \quad \xi_i \geq 0 \quad i = 1, \ldots, l \tag{1}$$

In the above formula, $\xi_i$ is the overall penalty for individual misclassification, whereas many SVM implementations consider a penalty value $C$, being the relative importance of maximising the margin to minimising misclassification.

Regardless of whether the data is separable or inseparable the data points are classified according to the side of the hyperplane on which they fall. It should also be noted that there does not exist a consensus within the Support Vector Machine community as to the optimal penalty to impose.

### 2.2 Non-Linear SVMs

Support Vector Machines are not limited to linear applications. One of the main advantages to the SVMs approach is the ability to generalise in order to accommodate a decision surface that is a non-linear function of the training data. Such classification is achieved through mapping the data points in the input space to a higher dimensional feature space,

$$F = \{\phi(\vec{x}) : \vec{x} \in \mathbb{R}^n\}$$

This mapping simply adds additional attributes to the data that are non-linear functions of the original data. Classification can then be performed using existing linear algorithms on the expanded dataset in the feature space which in turn produces nonlinear functions in the original input space. This non-linear calculation is achieved through the use of kernel functions and it is in this ability that the power of the Support Vector Machine technique lies. \cite{BC03} Kernel functions can be applied due to the fact that the input data to the Support Vector Machine is only used in a vector dot product calculation, $\vec{x}_i \cdot \vec{x}_j$, or in the case where the data has been mapped into a feature space, $\phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$. Therefore, the Support Vector Machine utilises a kernel function to calculate the required vector dot product, saving computational effort over performing the same calculation in the relevant feature space. This is especially useful in that it saves the calculation of $\phi(\vec{x})$ directly which could prove difficult given the high dimension of the feature space, which may even be infinite. \cite{Bur98} Some commonly used kernels are; Linear, Polynomial, Radial Basis Function and Sigmoid.

Again it is important to recognise at this point that while there is general consensus within the Support Vector Machine community as to what constitutes a Support Vector Machine in the case of clearly separable data, there is no such agreement on how best to define Support Vector Machines for non-separable data.

### 2.3 SVM Applications

Support Vector Machines, while initially used only for binary classification problems have additional applications. These include Regression, novelty detection and multiple class classification.
A regression SVM estimates the functional dependence of the dependent variable $y$ on a set of independent variables $x$. It is assumed, as in other regression problems, that the relationship between the independent and dependent variables is given by a deterministic function $f$ plus the addition of some additive noise: $y = f(x) + \text{noise}$. The additive noise follows a Gaussian Distribution [PMG00]. The goal is then similar to the classification problem, in this case to find a functional form $f$ that can correctly predict new cases that the SVM has not been presented with before.

Novelty detection is a special case of the classification problem wherein the majority of the data points are in one class with only a small percentage in the opposing class, yet it is correctly classifying these few point that we are most interested in. i.e. given a knowledge of one class, when can we say that a new instance of data is sufficiently unlikely to belong to that class that we should put it into a new class? This leads to an algorithm for carrying out unsupervised classification.

Multiple class classification builds on the classification SVM discussed previously but extends it to handle more than two classes. Several methods have been proposed to achieve this, the main being that we construct a multi-class classifier by combining several binary classifiers. Methods have also been proposed that consider all classes at once although these are computationally expensive [HL01].

The Support Vector Machine technique has successfully be applied to a number of practical domains, these include; face detection in images [OFG97c], text categorisation [Joa98], [TJ01], analysis of micro-array gene expression data [BGL+99] and in scientific visualisation [SSB05].

Given the growing applications of Support Vector Machines, there is increasing interest in improving the generalisation performance and for improving the speed in the training phase. To this end some initial research has been completed into improving the accuracy and classification speed of an SVM [BS97], though the biggest problem remains being how to improve SVMs in the cases of non-separable data.
3 SVM Software

There are numerous software packages available that implement Support Vector Machines in their various forms, from which two have been chosen for this project.

3.1 SVM\textsuperscript{light}

SVM\textsuperscript{light} is an implementation of Support Vector Machines written in C by Thorsten Joachims and is available free (including source code) for scientific use [Joa99]. The software attempts to solve the optimisation problem [Joa99]:

$$\begin{align*}
\text{minimise:} & \quad W(\alpha) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \\
\text{subject to:} & \quad \sum_{i=1}^{l} y_i \alpha_i = 0 \\
& \quad \forall i : 0 \leq \alpha_i \leq C
\end{align*}$$

(2)

This problem (2) can be shown to be equivalent to that previously defined in (1) using Wolfe dual convex form, and is known as a Lagrangian dual problem.

It should be noted that we are only interested in the vectors for which the $\alpha$ value is non-zero. These will be the support vectors.

In order to manage large data sets, the software uses the a General Decomposition Algorithm, which is based on the decomposition strategy proposed by Osuan et al [OFG97a].

The algorithm works by splitting variables ($\alpha_i$) into two categories:

- the set $B$ of free variables
- the set $N$ of fixed variables

The algorithm then iterates through the variables, at each stage the free variables are those which can be updated in the current iteration, whereas the fixed variables are temporarily fixed at a particular value. The set of free variables is also referred to as the working set and has a constant size $q$ which is much smaller than $l$. The algorithm is therefore:

- While the optimality conditions are not violated
  - Select $q$ variables for the working set $B$. The remaining $l-q$ variables are fixed at their current value.
  - Decompose problem and solve QP-subproblem: optimise $W(\alpha)$ on $B$.
- Terminate and return $\alpha$.

where QP stands for Quadratic Programming.

3.1.1 Input and Outputs

In order to use the SVM\textsuperscript{light} software, one must (at least) provide an input file in the format specified below.

For an input file of:

```
1 1 : -5 2 : 5
-1 1 : 4 2 : -1
```
The first value is the class, 1 of -1, and following that are feature value pairs. In such pairs, the first element is an integer giving the number of the attribute. The second element is the value for that attribute. If an attribute is not explicitly assigned a value it will be given the value zero (0) by default.

The (default) software output is:

Scanning examples...done
Reading examples into memory...OK. (5 examples read)
Setting default regularization parameter C=0.0413
Optimizing..............done. (15 iterations)
Optimization finished (0 misclassified, maxdiff=0.00078).
Runtime in cpu-seconds: 0.00
Number of SV: 4 (including 1 at upper bound)
L1 loss: loss=0.83379
Norm of weight vector: |w|=0.23561
Norm of longest example vector: |x|=7.07107
Estimated VCdim of classifier: VCDim<=3.77562
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=40.00% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>66.67% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>66.67% (rho=1.00,depth=0)
Number of kernel evaluations: 116
Writing model file...done

with the produced model file being:

SVM-light Version V6.01
0 # kernel type
3 # kernel parameter -d
1 # kernel parameter -g
1 # kernel parameter -s
1 # kernel parameter -r
empty# kernel parameter -u
2 # highest feature index
5 # number of training documents
5 # number of support vectors plus 1
-0.66679635 # threshold b, each following line is a SV (starting with alpha*y)
-0.003646657702784701990896885502561 1:5 2:-5 #
0.006992538267347114712446654039013 1:-3 2:-1 #
-0.041345679802738878605428141099765 1:4 2:-1 #
0.0380017986788882568380103066374 1:1 2:3 #

Through interpreting this output we can see the resulting Support Vector Machine was made up of four support vectors with a normalised $w$ value of 0.23561 and a $b$ value of -0.66679635. The $w$ and $b$ values define the decision surface of the Support Vector Machine. The value chosen for C (misclassification penalty) was 0.0413 and there was zero
misclassification when the training data was tested on the final Support Vector Machine.

3.2 LIBSVM

LIBSVM is an integrated software package for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM). It supports multi-class classification. The library package includes the source code in both C++ and Java and is written by Chih-Chung Chang and Chih-Jen Lin [CL01]. For classification, the software attempts to solve the problem:

\[
\min_{w, b, \xi} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \\
\text{subject to: } y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i \\
\xi_i \geq 0, i = 1, \ldots, l
\]

In order to manage large data sets, since version 2.8 of the software it implements an Sequential Minimal Optimisation (SMO) type algorithm, originally proposed by Platt [Pla98]. SMO breaks the large QP problem into small subsets (a series of the smallest possible QP sub-problems including only two \(a\)’s at a time), which can be solved analytically. The algorithm itself is quite long and will not be presented here, although it can be found in [Pla98].

3.2.1 Inputs and Outputs

In order to use the LIBSVM software, one must (at least) provide an input file in the format specified below.

For an input file of:

1 1:-5 2:5
-1 1:4 2:-1
-1 1:5 2:-5
1 1:-3 2:-1
1 1:1 2:3

this is in the same format as that for SVM\text{\textit{light}}.

The (default) software output is:

```
optimization finished, #iter = 13
nu = 0.800000
obj = -2.333132, rho = -0.333335
nSV = 5, nBSV = 2
Total nSV = 5
```

with the produced model file being:

```
svm_type c_svc
kernel_type rbf
gamma 0.5
nr_class 2
total_sv 5
rho -0.333335
label 1 -1
```
Through interpreting this output we can see the resulting Support Vector Machine used a Radial Basis Function kernel and was made up of five support vectors of which three were positive class examples and two were negative. The value of $w$ is not specifically shown, while the value for $b$ is $-0.333335$ (rho).

### 3.3 Other SVM Software

Other Support Vector Machine software considered was Gist [NP], Gini [Cha], software produced by the Royal Holloway University of London and AT&T Laboratories [oLL] and SvmFu 3 [Rif].

These software packages were not considered further mainly due to the scope and time constraints of the project.

### 3.4 Wrapper Software

The idea of being able to interact with a number of Support Vector Machine packages through a common interface can be considered new as no reference to such a wrapper could be found in the literature. The benefits of having such software would include time taken to learn to use each SVM package as instead one would simply learn to use the wrapper. For example, when considering the simple example given for both SVMlight and LIBSVM previously, the same input data resulted in different Support Vector Machines, due in part to the default setting of initial parameters meaning different kernels were used. Other advantages would include the ability to run the same input data through multiple implementations and compare the resultant SVMs. In order to achieve this, each Support Vector Machine would need to be represented in a canonical form.
4 SVM Theory

There are many aspects of Support Vector Machine theory which, when explored, can and have led to advances in the SVM algorithm. This has mainly been in the form of speed both in training and later use of the SVM, however advances in efficiency and accuracy have also been obtained.

4.1 Normalisation of input data

The normalization of the vectors of the input space can be considered one of the most basic types of data preprocessing. Assume $x \in \mathbb{R}^N$ is an input vector, the corresponding normalized vector $\tilde{x}$ may be expressed as:

$$\tilde{x} = \frac{x}{\|x\|} \in \mathbb{R}^N$$

where $\|x\|^2 = \sum_{i=1}^{N} x_i^2$

This vector $\tilde{x}$ lies on a unit hypersphere of $\mathbb{R}^N$.

Normalising for Support Vector Machines often refers to the distance of a given vector from the origin. In such cases, normalisation would adjust the size of all vectors such that the longest was at a distance of 1.

Normalising the input vectors before passing them to the Support Vector Machine could, and perhaps should, be done using Mahalanobis Distance. This is the distance between two $N$ dimensional points scaled by the statistical variation in each component of the point [Fis02] and is particularly useful when comparing feature vectors whose elements are quantities having different ranges and amounts of variation.

Normalisation of the input data presents a problem for the Support Vector Machine algorithm since the latter requires input vectors in the feature space which also suitably scaled. Graf and Borer suggested that normalization in the feature space would present a solution to this problem [GB01]. Graf and Borer also noted that an SVM normalised in feature space outperforms an SVM normalised in the input space.

4.2 Trimming of input data

It can be assumed that in any given situation, if the amount of work to be done is reduced, it should be completed in a shorter period of time. In the case of training Support Vector Machines, if the same machine can be produced via a subset of the original data, trained in a smaller amount of time, but subsequently applied with the same accuracy, then any technique that would facilitate this is worth while.

A naive way by which to reduce the size of the training data set is to simply train the Support Vector Machine on a random sample of the total training examples. Bazzani et al. discovered that using a random selection of between 13% and 50% of the total training set produced only minor degradations in the generalisation performance of the resulting SVM [BBB+01]. An alternative approach is to remove examples from the training set in an intelligent fashion so that the remaining examples are the best representation of the overall training data. This idea lead El-Naqa et al. to develop the successive enhancement-learning (SEL) algorithm.

The SEL algorithm uses iteration in selecting the optimum subset of the overall dataset. Let $Z^{(n)}$ be the current training subset on the $n$-th iteration of the algorithm, where $Z = \{(\tilde{x}_1, y_1), (\tilde{x}_2, y_2), ..., (\tilde{x}_l, y_l)\}$. Initially, $Z^{(0)}$ is simply a randomly selected subset of
the training dataset. For each iteration, the current training subset $Z^{(n)}$ is used to train a Support Vector Machine. The resulting machine is tested on the entire training dataset. All the misclassified examples are recorded and the subset used for the next iteration $Z^{(n+1)}$ consists of $Z^{(n)}$ in which $N$ of the correctly classified training examples from $Z^{(n)}$ have been replaced with $N$ misclassified examples as previously recorded. The algorithm is then repeated until convergence is achieved. Proof of convergence of the successive enhancement-learning algorithm can be found in El-Naqa et al [ENYW+02].

As has been previously mentioned the trained Support Vector Machine is dependant solely on the training vectors for which the $\alpha$ value was non-zero. These are the support vectors, and the remaining examples in the training data could be discarded without any ill effect. It can therefore be concluded that if it were possible to identify the likely support vectors in the training examples and thus form a subset containing these vectors alone the training time for the Support Vector Machine would be considerably reduced. This is the idea behind the approach taken by Shin and Cho in their selective k-nearest neighbour spanning algorithm [SC03].

The algorithm developed by Shin and Cho centers around two key observations;

- a training point near the decision boundary (and hence possible support vector) is likely to be surrounded by points that belong to a different class.
- the neighbours of training points near the decision boundary are themselves likely to be near the decision boundary.

In the most basic sense, the algorithm iterates over the set of training data points, adding them to the set of possible support vectors if the set of k-nearest neighbours of a given point contains points belonging to at least two different classes.

4.3 SVM starting conditions

While there has been some research into finding and providing optimal parameters (such as the penalty parameter $C$) for Support Vector Machines [CVBM02], there is no mention in the literature of experimentation with the ordering of the vectors in the training data set.
5 Conclusion

Use of Support Vector Machines is becoming more widespread and as such, the average level of user knowledge and understanding of the SVM algorithm is decreasing. With this in mind, it would become increasingly valuable to have a standard platform from which to implement and test available SVM software. In addition, the amount of data available for initial training can be large however the amount of data required is significantly less. Ways in which to reduce the size of the training data set are of great value and importance. This and any other training speed increase that can be gained would enhance the SVM approach possibly leading to solutions to previously unsolved problems.
References


