e-PGPathshala
Subject : Computer Science
Paper: Data Analytics
Module No 14: CS/DA/14 - Data Analysis Foundations – Support Vector Machines
Quadrant 1 – e-text

1.1 Introduction

Support Virtual Machine is a new promising non-linear, non-parametric classification technique, which already showed good results in the medical diagnostics, optical character recognition, electric load forecasting and other fields. This chapter give an overview on SVM concepts, implementation and applications of SVM.

1.2 Learning Outcomes

- **Learn the basics of linearly separable problems**
- **Understand the fundamentals of support vector machine concepts**
- **Learn about implementations and applications of SVMs**

1.3 Support Virtual Machine [1]

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in figure 1 below. In this example, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labelled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).

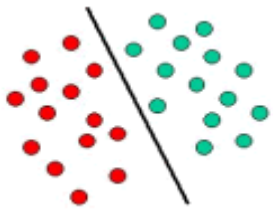

Fig 1. Linear Plane as decision boundary

The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train

cases). This situation is depicted in the illustrated in figure 2 given below. Compared to the previous schematic, it is clear that a full separation of the GREEN and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.

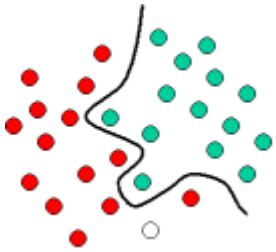

Fig 2. Non-Linear Plane as decision boundary

The illustration in figure 3 shows the basic idea behind Support Vector Machines. Here we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the GREEN and the RED objects.

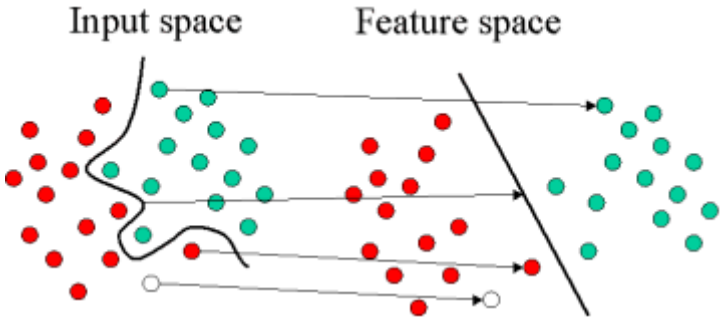

Fig 3. Basic Idea Behind Support Vector Machines

1.4.General input/output for SVM

The input to SVM algorithm is a set of training pair samples comprising of the following:

- The input sample features x_1, x_2…x_n, and the output result y.
- Typically, there can be lots of input features x_i

The output is a set of weights w (or w_i), one for each feature, whose linear combination predicts the value of y.

The important difference between SVM and other classification techniques is that it optimizes the width of the margin to reduce the number of weights that are nonzero to just a few that correspond to the important features that ‘matter’ in

deciding the separating line(hyperplane). These nonzero weights correspond to the support vectors (because they 'support' the separating hyperplane)

1.5 Defining the separating hyper plane

There are many linear classifiers (hyper planes) that separate the data. However only one of these achieves maximum separation. The reason we need it is because if we use a hyper plane to classify, it might end up closer to one set of datasets compared to others and we do not want this to happen and thus we see that the concept of maximum margin classifier or hyper plane as an apparent solution. The equation defining the decision surface separating the classes is a hyperplane (in figure 4) is of the form:

$w^T x + b = 0$

where, w is a weight vector
x is input vector
b is bias

H1 and H2 are the planes in figure 4:

H1: w•xi+b = +1
H2: w•xi+b = –1

The points on the planes H1 and H2 are the tips of the Support Vectors. The plane H0 is the median in between, where

w•xi+b =0
d+ = the shortest distance to the closest positive point
d- = the shortest distance to the closest negative point
The margin (gutter) of a separating hyperplane is (d+) + (d–)

The above equation can be rewritten as shown below:

$w^T x + b \geq 0$ for $d_i = +1$
$w^T x + b < 0$ for $d_i = -1$

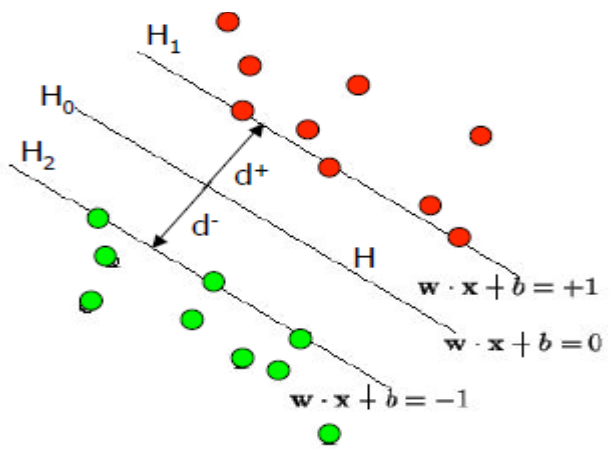

Figure 4. Margin of Separation

Margin of Separation (d): *the separation between the* hyperplane and the closest data point for a given **weight vector w** and **bias b.**

Optimal Hyperplane (maximal margin): the particular hyperplane for which the margin of separation d is maximized.

Maximizing the Margin

Expression for Maximum margin is given as shown below:

$$\text{margin} \equiv \arg\min_{\mathbf{x}\in D} d(\mathbf{x}) = \arg\min_{\mathbf{x}\in D} \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

Distance of closest point on hyperplane to origin can be found by maximizing the x as x is on the hyper plane. Similarly for the other side points we have a similar scenario. Thus solving and subtracting the two distances we get the summed distance from the separating hyperplane to nearest points. Maximum Margin = M = 2 / ||w||

Now we have a quadratic optimization problem and we need to solve for w and b. To solve this we need to optimize the quadratic function with linear constraints. The solution involves constructing a dual problem and where a Langlier's multiplier αi is associated. We need to find w and b such that Φ (w) =½ |w'||w| is minimized;
And for all {(x_i, y_i)}: y_i (w * x_i + b) ≥ 1.

Now solving: we get that w =Σα_i * x_i; b= y_k- w *x_k for any x_k such that α_k[1] 0
Now the classifying function will have the following form: f(x) = Σα_i y_i x_i * x + b

1.6 Slack Variables

In real world problem it is not likely to get an exactly separate line dividing the data within the space as shown in figure 5. And we might have a curved decision boundary. We might have a hyperplane which might exactly separate the data but this may not be desirable if the data has noise in it. It is better for the smooth boundary to ignore few data points than be curved or go in loops, around the outliers.

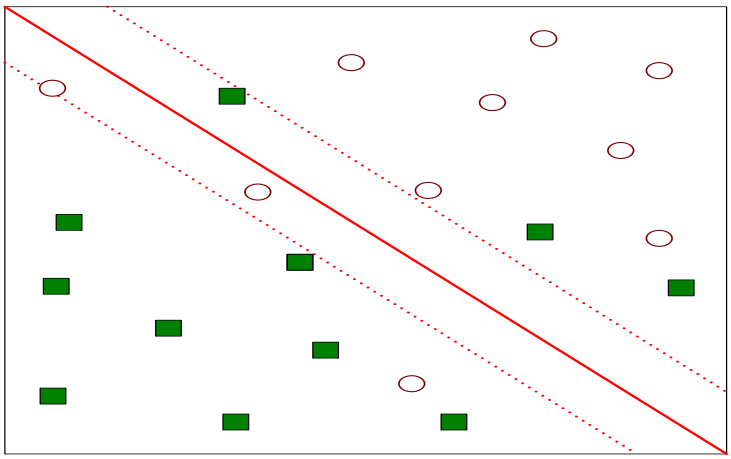

Figure 5. Non-linear problem

This is handled in a different way; here we hear the term slack variables being introduced. Now we have, $y_i(w'x + b) \geq 1 - S_k$ [4] [12]. This allows a point to be a small distance S_k on the wrong side of the hyper plane without violating the constraint. Now we might end up having huge slack variables which allow any line to separate the data, thus in such scenarios we have the Lagrangian variable introduced which penalizes the large slacks.

$$\min L = \tfrac{1}{2} w'w - \sum \lambda_k (y_k (w'x_k + b) + s_k - 1) + \alpha \sum s_k$$

Where reducing α allows more data to lie on the wrong side of hyper plane and would be treated as outliers which give smoother decision boundary.

1.7 Non-linear Nonlinear Support Vector Machines

There may be real problems where the decision boundary is not linear as shown in figure 6, then transform data into higher dimensional space as shown in figure 7, thereby transforming a linearly inseparable problem to a linearly separable one.

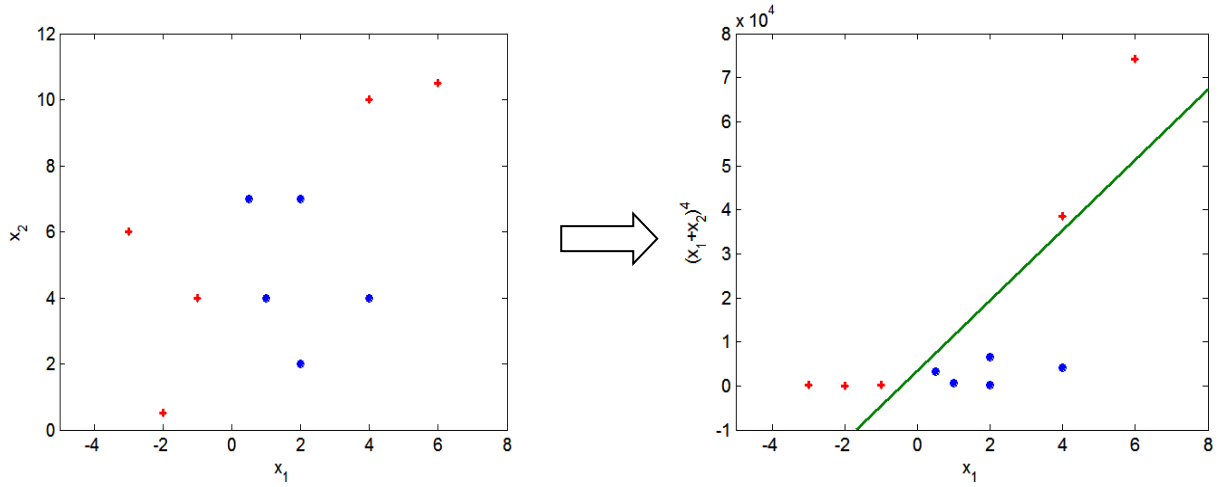

Figure 6. Linearly inseparable **Figure 7.** Linearly separable

To solve problems highlighted in figure 6 we require Kernal functions.

1.7.1 Kernel

Let's first look at few definitions as what is a kernel and what does feature space mean?

Kernel: If data is linear, a separating hyper plane may be used to divide the data. However it is often the case that the data is far from linear and the datasets are inseparable. To allow for this kernels are used to non-linearly map the input data to a high-dimensional space. The new mapping is then linearly separable.

This mapping is defined by the Kernel: $K(x,y) = \Phi(x) \cdot \Phi(y)$

Feature Space: Transforming the data into feature space makes it possible to define a similarity measure on the basis of the dot product. If the feature space is chosen suitably, pattern recognition can be easy.

$$\langle x_1 \cdot x_2 \rangle \leftarrow K(x_1, x_2) = \langle \Phi(x_1) \cdot \Phi(x_2) \rangle$$

Now getting back to the kernel trick, we see that when w,b is obtained the problem is solved for a simple linear scenario in which data is separated by a hyper plane. The Kenral trick allows SVM's to form nonlinear boundaries. Steps involved in kernel trick are given below:

[a] The algorithm is expressed using only the inner products of data sets. This is also called as dual problem.
[b] Original data are passed through non linear maps to form new data with respect to new dimensions by adding a pair wise product of some of the original data dimension to each data vector.
[c] Rather than an inner product on these new, larger vectors, and store in tables and later do a table lookup, we can represent a dot product of the data after doing non linear mapping on them. This function is the kernel function.

More on kernel functions is given below. The idea of the kernel function is to enable operations to be performed in the input space rather than the potentially high dimensional feature space. Hence the inner product does not need to be evaluated in the feature space. We want the function to perform mapping of the attributes of the

input space to the feature space. The kernel function plays a critical role in SVM and its performance. It is based upon reproducing Kernel Hilbert Spaces [8] [14] [15] [18].

$$K(x, x') = \langle \phi(x), \phi(x') \rangle,$$

If K is a symmetric positive definite function, which satisfies Mercer's Conditions,

$$K(x, x') = \sum_{m}^{\infty} a_m \phi_m(x) \phi_m(x'), \quad a_m \geq 0,$$

$$\iint K(x, x') g(x) g(x') dx dx' > 0, \quad g \in L_2$$

Then the kernel represents a legitimate inner product in feature space. The training set is not linearly separable in an input space. The training set is linearly separable in the feature space. This is called the "Kernel trick" [8] [12].

The different kernel functions are listed below [8]: More explanation on kernel functions can be found in the book [8]. The below mentioned ones are extracted from there and just for mentioning purposes are listed below.

1] *Polynomial:* A polynomial mapping is a popular method for non-linear modeling. The second kernel is usually preferable as it avoids problems with the hessian becoming Zero.

$$K(x, x') = \langle x, x' \rangle^d.$$
$$K(x, x') = \left(\langle x, x' \rangle + 1 \right)^d.$$

2] Gaussian Radial Basis Function: Radial basis functions most commonly with a Gaussian form

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2} \right)$$

3] Exponential Radial Basis Function: A radial basis function produces a piecewise linear solution which can be attractive when discontinuities are acceptable.

$$K(x, x') = \exp\left(-\frac{\|x - x'\|}{2\sigma^2} \right)$$

4] Multi-Layer Perceptron: The long established MLP, with a single hidden layer, also has a valid kernel representation.

$$K(x, x') = \tanh\left(\rho \langle x, x' \rangle + \varrho \right)$$

There are many more including Fourier, splines, B-splines, additive kernels and tensor products.

1.8 SVM vs linear regression

Linear regression can only come up with a straight line decision boundary as shown in figure 8 (a). You can never come up with a squiggly curve kind of a decision boundary with logistic regression. **SVM** can do come up with a decision boundary which need not be a straight line as shown in figure 8 (b) if you have a non-linear kernel.

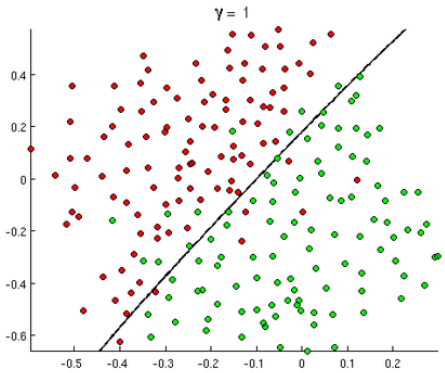

Figure 8 (a) Linear Regression

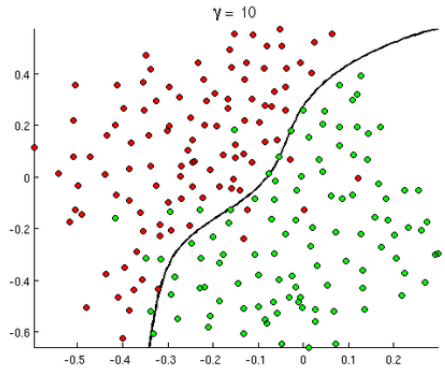

Figure 8 (b) SVM

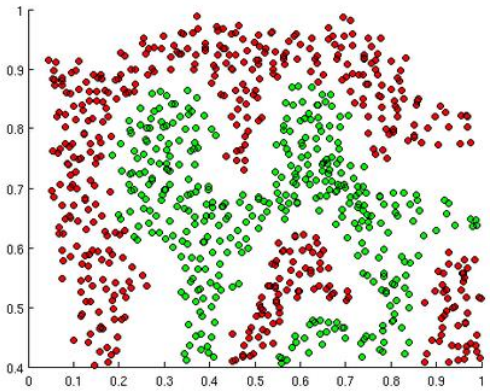

Figure 9 (a) Messy Data

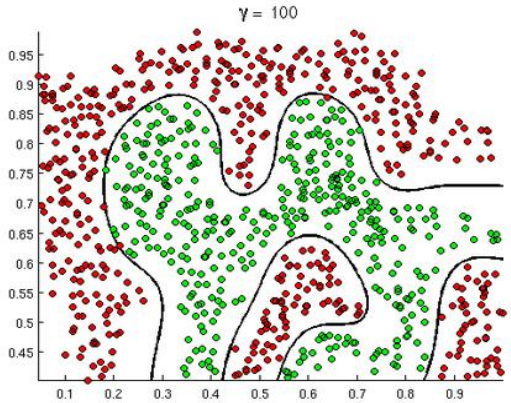

Figure 9 (b) Correctly trained SVM output

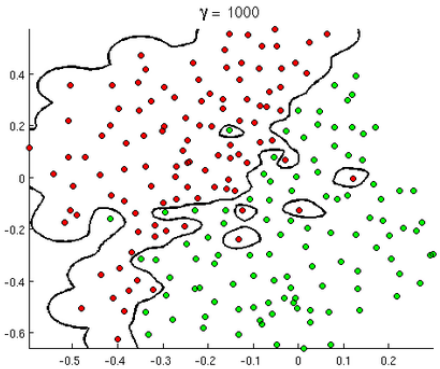

Figure 9 (c) Incorrectly trained SVM output

But the real power of SVM is seen when you have data like the one shown in figure 9 (a). You'll be in a mess if you use logistic regression to separate out the data with a straight line. In fact, you cannot do it correctly with logistic regression. But if you correctly train your SVM you can achieve something like the one in figure 9 (b) else if you don't train it you could also end up with something like figure 9 (c):

1.9 SVM Implementations

i) SVM^{light}

SVM^{light} is an implementation of Support Vector Machines (SVMs) in C. The main features of the program are the following:

- fast optimization algorithm
    - working set selection based on steepest feasible descent
    - "shrinking" heuristic
    - caching of kernel evaluations
    - use of folding in the linear case
- solves both classification and regression problems
- computes XiAlpha-estimates of the error rate, the precision, and the recall
- efficiently computes Leave-One-Out estimates of the error rate, the precision, and the recall
- includes algorithm for approximately training large transductive SVMs (TSVMs)
- can train SVMs with cost models
- handles many thousands of support vectors
- handles several ten-thousands of training examples
- supports standard kernel functions and lets you define your own
- uses sparse vector representation

ii) SVM^{struct}

SVM^{struct} is a Support Vector Machine (SVM) algorithm for predicting multivariate or structured outputs. It performs supervised learning by approximating a mapping
h: X --> Y
using labeled training examples (x_1,y_1), ..., (x_n,y_n). Unlike regular SVMs, however, which consider only univariate predictions like in classification and regression, *SVM^{struct}* can predict complex objectsy like trees, sequences, or sets. Examples of problems with complex outputs are natural language parsing, sequence alignment in protein homology detection, and markov models for part-of-speech tagging. The *SVM^{struct}* algorithm can also be used for linear-time training of binary and multi-class SVMs under the linear kernel.
The *SVM^{struct}* implementation is based on the *SVM^{light}* quadratic optimizer .
SVM^{struct} can be thought of as an API for implementing different kinds of complex prediction algorithms. Currently, we have implemented the following learning tasks:

- ***SVM^{struct} Python***: A python interface to the *SVM^{struct}* API for implementing your own structured prediction method. The Python interface makes prototyping much easier and faster than working in C.

- ***SVM^{struct} Matlab***: A matlab interface to the *SVM^{struct}* API for implementing your own structured prediction method. Again, prototyping should be much easier and faster than working in C.

- ***Latent SVM^{struct}***: Training of structural SVM predictions rules when the training labels are not fully observed (e.g. unobserved dependency structure in NP-coref, motif finding, ranking with weak orderings).

- ***SVM^{multiclass}***: Multi-class classification. Learns to predict one of k mutually exclusive classes. This is probably the simplest possible instance of *SVM^{struct}* and serves as a tutorial example of how to use the programming interface.

- ***SVM^{perf}***: Learns a binary classification rule that directly optimizes ROC-Area, F1-Score, or the Precision/Recall Break-Even Point. It is also a training algorithm for conventional linear binary classification SVMs that can be orders of magnitude faster than SVM-light for large datasets.
  SVM^{rank}: Learns a rule for predicting rankings as typically used in search engines and other retrieval systems. It is equivalent to SVM-light in '-z p' mode, but it is a much more efficient algorithm for training Ranking SVMs.

- ***SVM^{pairMRF}***: Semantic scene labeling for 3D point cloud data. Basically learns a general Markov Random Field model with pairwise potentials and can be used beyond that specific application.
  SVM^{sle}: Learning algorithm for predicting document-level sentiment polarities with latent explanations.

- ***SVM^{struct} for activity recognition***: Learning algorithm for training activity recognizers for video.

SVM^{struct} for web-page segmentation: Learning algorithm for segmenting web pages based on directed acyclic graph structure.

iii) LIBSVM

LIBSVM is an integrated software for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM). It supports multi-class classification.

LIBSVM provides a simple interface where users can easily link it with their own programs. Main features of **LIBSVM** include

- Different SVM formulations
- Efficient multi-class classification
- Cross validation for model selection
- Probability estimates
- Various kernels (including precomputed kernel matrix)
- Weighted SVM for unbalanced data
- Both C++ and Java sources
- GUI demonstrating SVM classification and regression
- Python, R, MATLAB, Perl, Ruby, Weka, Common LISP, CLISP, Haskell, OCaml, LabVIEW, and PHP interfaces. C# .NET code and CUDA extension is available. It's also included in some data mining environments: RapidMiner, PCP, and LIONsolver.
- Automatic model selection which can generate contour of cross validation accuracy.

iv) mySVM

mySVM is an implementation of the Support Vector Machine introduced by V. Vapnik. It is based on the optimization algorithm of SVM*light* as described in mySVM can be used for pattern recognition, regression and distribution estimation.
The input of mySVM consists of:

- a parameter definition
- a kernel definition
- one or more example sets

Input lines starting with "#" are treated as commentary. The input can be given in one or more files. If no filenames or the filename "-" are given, the input is read from stdin. mysvm trains a SVM on the first given example set. The following example sets are used for testing (if their classification is given) or the functional value of the examples is being computed (if no classification is given).

attern	use SVM for pattern recognition, y has to be in {-1,1}.
regression	use regression SVM *(default)*
nu *float*	use nu-SVM with the given value of nu instead of normal SVM (see [Schoelkopf/etal/2000a] for details on nu-SVMs).
distribution	estimate the support of the distribution of the training examples (see [Schoelkopf/etal/99a]). Nu must be set!
verbosity [1..5]	ranges from 1 (no messages) over 3 (default) to 5 (flood, for debugging only)

scale	scale the training examples to mean 0 and variance 1 (default)
no_scale	do not scale the training examples (may be numerically less stable!)
format	set the default example file format. See the description here.
delimiter	set the default example file format. See the description here.

v) Matlab

MATLAB is a programming language developed by MathWorks. It started out as a matrix programming language where linear algebra programming was simple. It can be run both under interactive sessions and as a batch job.

vi) Huller

Huller is a simple and efficient online SVM approximation method published by Antoine Bordes and Leon Bottou. They propose a novel online kernel classifier algorithm that converges to the Hard Margin SVM solution. The same update rule is used to both add and remove support vectors from the current classifier. Experiments suggest that this algorithm matches the SVM accuracies after a single pass over the training examples. This algorithm is attractive when one seeks a competitive classifier with large datasets and limited computing resources.

1.8 Applications

- SVMs are helpful in text and hypertext categorization
- Classification of images can also be performed using SVMs
- SVMs are also useful in medical science
- Hand-written characters can be recognized using SVM

Summary

- SVMs express learning as a mathematical program taking advantage of the rich theory in optimization
- SVM uses the kernel trick to map indirectly to extremely high dimensional spaces
- SVMs extremely successful, robust, efficient, and versatile while there are good theoretical indications as to why they generalize well