

Support Vector Machines

Alexander Vegas Fairley

November 10, 2007

Abstract

I briefly motivate and explain the use of Support Vector Machines in the solution of discriminative classification problems. I follow and summarize the excellent tutorial of Christopher C. Burgess of Bell Labs, which is available at

<http://research.microsoft.com/~cburgess/papers/SVMtutorial.pdf>

1 Introduction

The name **Support Vector Machines** refers to a category of Discriminative Classifiers which have been shown to have quite nice behavior in empirical studies. They are based on work begun by Vladimir Vapnik in the late 70s, and have recently seen a great surge in popularity. Initially developed to solve **linear classification** problems, they have since been generalized to non-linear classification by use of what is called **the kernel trick**, due to the application of work M. Aizerman carried out in the 60s.

1.1 Learning Machines

We will consider a **Learning Machine \mathbf{L}** to be a parametrized set of functions from some input space into an output space consisting of two categories. Formally,:

$$L = \{f_\alpha : X \rightarrow Y | \alpha \in A\}$$

where A is some abstract parameter space. For example, we could consider the class of Feedforward Neural Nets with some particular topology and weights given by α as learning machine under this definition. A simpler example would be the set of lines in \mathbb{R}^2 . In this case, we could consider A to be the space of normal vectors and distances from the origin as we did when constructing our Hough transform implementation. We will call each function $\in \mathbf{L}$ a **Trained Machine**. For example, a particular assignment of weights to the connections of a neural network or the choice of a particular classification line would be a **Trained Machine**. Note that this definition allows us to distinguish between the capacity of the learning machine to learn, and the particular method in which we proceed from one **Trained Machine** to a hopefully superior one. This allows us to analyze rigorously what sorts of classifications can be learned by a particular **Learning Machine**, without having to concern ourselves with the particular method of learning.

1.2 Vapnik-Chervonenkis Dimension

The Vapnik Chervonenkis Dimension, or **VC Dimension** of a learning machine \mathbf{L} is the maximum h for which there exists a set X of cardinality h which is **shatterable** into all possible labelings by some **Trained Machine**. This is a fairly subtle concept, and is not without its shortcomings as a way of categorizing Learning Machines.

1.3 Risk

When we talk about the **Risk** of a trained machine, we are talking about the likelihood that it will misclassify datapoints. Abstractly we can represent this by the integral:

$$R(\alpha) = \int_{\mathbf{x}, y} \frac{1}{2} |y - f(\mathbf{x}, y)| dP(\mathbf{x}, y)$$

where $P(\mathbf{x}, y)$ is a cumulative probability distribution over inputs and categories. Of course, this integral is generally impossible to calculate in practice, because if we knew $P(\mathbf{x}, y)$ we would have no need of machine learning. Instead, we will concern ourselves with the notion of **Empirical Risk**, which we will define as:

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^l |y - f_{\alpha}(\mathbf{x}_i)|.$$

Empirical Risk turns out to be a very useful, as it allows us place a bound on the actual risk of a particular **Trained Machine**. This bound is in terms of the **VC Dimension** of the learning machine which the trained machine is a member of. We will call the VC Dimension of our Learning Machine h . Vapnik 1995 establishes this bound in terms of a parameter η giving the probability that the bound holds. For each particular $\eta \in [0, 1]$ we have a bound

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{h \left(\log \left(\frac{2l}{h} \right) + 1 \right) - \log \left(\frac{\eta}{4} \right)}{l}}$$

which will hold with probability $1 - \eta$. This bound suggests that in general, lower VC dimension will give us better generalizability, as of course will larger training sets. However, it turns out that SVMs actually have infinite VC dimension yet are claimed to S very well. The argument explaining this originally advanced by Vapnik is flawed, and as of the time of Burges' writing(1998) no proof of the generalizability of SVMs has been found. At the end of Burges tutorial is a rather complex argument about a very odd family of functions called "Gap Tolerant Classifiers" which gives some insight into the performance of SVMs, but is not a proof of their virtue.

2 What is a Support Vector Machine?

Support Vector Machines are also called **Maximum Margin Classifiers**, a name which is arguably more descriptive of them. The classical Support Vector Machine is a linear classifier, which, given a labeled data set $(\{\mathbf{x}_i, \omega_i\} | \mathbf{x}_i \in \mathbb{R}^n, \omega_i \in \{-1, +1\})$, will attempt to find a hyperplane (linear subspace of dimension $n - 1$) which separates the x_i based on the ω_i , with maximum **margin**. The **margin** of a datapoint is the distance between it and the separating hyperplane. The **margin** of a dataset is the minimum margin of any point contained in the set. The hyperplane can be described by its **normal**, \mathbf{w} which is a vector perpendicular to the plane, and its **bias**, b , which is the perpendicular distance between the origin of the input space and the hyperplane.

It's important to distinguish between the idea of a Support Vector Machine, and algorithms for finding the optimal SVM for a particular dataset. An SVM is essentially just a classification function based on the maximum margin hyperplane separating two labeled data sets. We learn this function via some algorithm on a training set, and then we use the function to classify future data points. There are multiple ways of construing this learning problem. The text formulates the problem in the language of Lagrange Multipliers as the minimization of the function:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i \omega_i (\mathbf{w} \cdot \mathbf{x}_i + b) + \sum_{i=1}^N \alpha_i$$

This is then a **quadratic programming** problem, which can be solved by numerical packages. However, note that Thorsten Joachims published a new technique at KDD 2006, which gives a cutting plane algorithm for training Support Vector Machines in time **linear** in the size of the data. You can download an implementation of his algorithm from his website, and it is free for scientific use (though you must cite him twice!) and available for licensing for commercial use.

However, not all data sets can be separated by hyperplanes. SVMs have been generalized to handle non-linearly separable data sets in two ways. The first is the notion of a **soft-margin classifier**, which uses **slack variables** to penalize, but allow some data points to be misclassified. The more successful generalization uses the idea of **the kernel trick** which is a somewhat mathematically advanced business. The essential idea is that the data points can be mapped into some higher dimensional feature space by some function Φ where they **are** linearly separable. If the SVM learning problem is formulated differently (in what is known as the **Wolfe Dual**) it can be defined solely in terms of inner products of pairs of vectors. Utilizing the Wolfe Dual leads to a nice optimization, since rather than computing $\Phi(\mathbf{x}_i)$, for all training vectors we can instead compute $K(\mathbf{x}_i, \mathbf{x}_j)$ where K is what we call a **kernel function**. A kernel function gives an **inner product** in some space, that is to say that:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

We can then use the resulting kernel matrix K_{ij} to compute the separating hyperplane in feature space, which will map back into some non-linear hyper-surface in our original space. There are many different kernels that can be used, and in fact, a criteria from functional analysis known as **Mercer's theorem** guarantees us an infinity of them.

3 How can I use SVMs, and why should I

As mentioned previously, there is no rigorous proof that SVMs are good tools for machine learning. However they have many proponents, and there are a great number of free implementations. I will now briefly demo libsvm, a free c++ library which compiles painlessly on modern operating systems, and allegedly on windows as well.

4 More Resources

1. Excellent Tutorial:
<http://research.microsoft.com/~cburgess/papers/SVMtutorial.pdf>
2. Empirical Comparison
<http://www.meraka.org.za/pubs/CvdWalt.pdf>
3. LibSVM
<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
4. Cutting Plane Linear Training Algorithm
http://www.cs.cornell.edu/People/tj/publications/joachims_06a.pdf