VC-Dimension and its Applications in Machine Learning

Xu Miao, Lin Liao
xm@u.washington.edu, liaolin@cs.washington.edu

Abstract

VC-dimension is one of the basic concepts of learning theory. In this paper, we give an overview of VC theory. We discuss the relations between the VC-dimension and the learnability, and the relations between the VC-dimension and the generalization performance. Especially we investigate the issue of applying the theory of VC-dimension in the practice of machine learning. We study the applications of VC theory in two cases: decision tree pruning and model selection of support vector machine.

1 Introduction

Supervised learning, or learning-from-examples, refers to systems that are trained, instead of programmed, with a set of examples. The key point of supervised learning is that the learning should be *predictive*. That is, it should be able to give lower error rates on *test* data than those machines without learning. The performance of a learning machine on test data is called *generalization* performance of the machine. The investigation of the generalization performance of a learning machine has a long history that can be tracked back to 1970's [12]. It has been found that for a given learning task, with a given finite set of training data, the best generalization performance will be achieved if the right balance is struck between the accuracy attained on that particular training set and the *capacity* (or expressiveness) of the machine. A machine with more capacity could give low training errors but might overfit the data and thus perform badly on test data. A machine with less capacity is not going to overfit, but restricted in what it can model. Therefore, the key question here becomes how we characterize the the capacity of learning machines.

Vapnik and Chervonenkis [12] presented the concept of VC-dimension. VC-dimension provides a quantitative way to measure the capacity of a learning machine and has been used to analyze many learning machines such as neural nets, decision trees, support vector machines and many more. In this paper, we give an overview of the VC-dimension theory. In particular, we are interested in its applications in learning theory and practical learning algorithms.

In the next section, we will give an introduction of VC-dimension theory, including the PAC model, the concepts of VC-dimension and effective VC-dimension and structural risk minimization, a model selection framework based on VC-dimension. Then in section 3, we discuss the applications of VC-dimension theory on two popular machine learning algorithms, decision tree and support vector machine, and demonstrate the results of our experiments. At the end, we state our conclusions.

2 VC-Dimension Theory

In the section, we review the main results of VC-dimension theory. The goal is to give a comprehensive picture of the theory and we will leave the proofs of most theorems to the referenced literatures.

2.1 PAC Model

We first give the definition of a learning machine (aka. concept class, or a family of concepts).

Definition A learning machine is defined by a set of possible mappings $\mathbf{x} \mapsto f(\mathbf{x}, \alpha))$, where $\mathbf{x}$ is a vector in domain X, y is the output which can only take value $+1$ or -1 (i.e. we only two classes throughout the paper). α is the parameter that could be adjusted in during training. A particular choice of α generates a "trained machine".

The model we will now introduce is known under a number of different names depending on the discipline concerned. Within statistics it would be known as the study of rates of uniform convergence, or frequentist inference [11], but within computer science it is generally referred to as the *probably approximately correct* or *PAC* model [7; 5]. The goal of the model is to learn a concept so that with a high degree of confidence the prediction error will be small.

The key assumption on which the model is based is that the data used in training and testing are generated independently and identically according to a distribution D. We assume that D is:

- Fixed throughout the learning process.
- Unknown to the learning machine.
- The instances are chosen independently.

The target concept is specified as a computable function c_t, thus our instances are of the form $< x, c_t(x) >$. Our goal is to find a hypothesis h which approximate c_t with respect to D in the following sense.

Let

$$error(h) = Prob_D\{c_t(x) \neq h(x)\}$$

We would like to ensure that $error(h)$ is below a certain threshold ϵ, which is given as a parameter of the algorithm. This parameter is a measure of the accuracy of the learning process.

As a measure of the confidence in the outcome of the learning process, we add another parameter η. We require the following hold:

$$Prob\{error(h) < \epsilon\} \geq 1 - \eta$$

Besides the parameters ϵ and η, the PAC algorithm also has access to the labeled training data, which are generated using the distribution D and labeled by c_t.

We say that an algorithm A *learns* a family of concepts C if for *any* $c_t \in C$ and *any* distribution D, A outputs a hypothesis h, such that the probability that $error(h) < \epsilon$ is at least $1 - \eta$.

One of the key ingredients of the PAC approach is that the error should not depend on the distribution D. This means that the bounds must hold whatever the distribution generating the examples, property sometimes referred to as *distribution free*. It is not surprising that some distributions make learning harder than others, and so a theory that holds for all distributions must inevitably be pessimistic in many cases. In the following subsection, we introduce the concept of VC-dimension in the sense of distribution free. In the discussion of support vector machine (see Section 3.2, we break this worst case deadlock and investigate the VC-dimensions related to the distributions, called *effective* VC-dimension.

2.2 VC-Dimension

[6; 8] For a finite set of concepts C, the probability that all l of the independent training examples are consistent with a hypothesis h for which $error(h) > \epsilon$, is bounded by

$$\begin{aligned} &Prob\{h \text{ is consistent with } l \text{ examples and } error(h) > \epsilon\} \\ &\leq \quad (1-\epsilon)^l \\ &\leq \quad e^{-\epsilon l} \end{aligned}$$

where the second inequality is a simple mathematical bound. Since the number of possible hypotheses is $|C|$, we know the probability that one of them is consistent with training data and has error greater than ϵ is at most

$$|C|e^{-\epsilon l}$$

Since we want this probability less than η, we have

$$\epsilon = \frac{1}{l} ln \frac{|C|}{\eta}$$

This derivation is simple, but it cannot be used to handle the infinite concept classes, perhaps even not enumerable. The major contribution of the VC-dimension theory is to extend such an analysis to infinite sets of concepts. In this section we will introduce the definition of VC-dimension and the measurement of VC-dimension. The concept of VC-dimension will provide us a substitute to to $ln|C|$, for infinite concept classes.

Definition of VC-Dimension
We start with the definition of *shattering*.

Definition Assume C is a concept class defined over instance space X. Let $S = \{\mathbf{x_1}, \cdots, \mathbf{x_m}\} \subseteq X$. A concept class *shatters* S if every possible function on S can be represented by some $c \in C$

. Here a function is a mapping from input $\mathbf{x}$ to output y. Thus the number of different samples generated by applying every possible function on S is 2^m, where $m = |S|$.

Now we are ready to define the notion of VC-dimension.

Definition VC-dimension of concept class C is the maximum size of a set shattered by C. VC-dimension of C is ∞ if such a maximum size does not exist.

Measuring VC-Dimension
Usually we measure the VC-dimension based on its definition. That is, to show the VC-dimension of a concept class is m, we have to demonstrate it can shatter a set of size n and it cannot shatter *any* set of size $n + 1$.

We give a number of examples on how to do that.

Example Suppose that the space in which the data live is R^2 and the concept class C consists of oriented straight lines, so that for a given line, all points on one side are assigned the class 1, and all points on the other side , the class -1. As shown in Figure 1, it is possible to find three points that can be shattered by C. We can also easily verify that it is not possible to find four. Thus the VC-dimension of C is three [2].

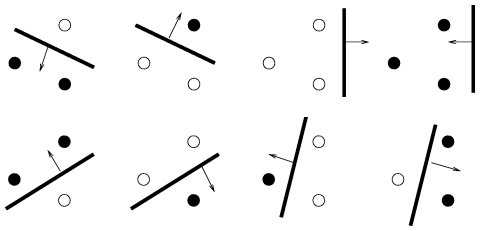


Figure 1: *Three points in* $\mathbf{R}^2$*, shattering by oriented lines*

We can extend this example to high dimensional space and get the following theorem [2]:

Theorem 2.1 *The VC-dimension of the set of oriented hyperplane in* R^n *is* $n + 1$.

One key of shattering is the set of points we choose. In many situations, it is not straightforward as the first example. Following is a more interesting example [6].

Example Parity function. Let $X = \{0,1\}^n$. The concept class C is defined as $\chi_S(x) = \bigoplus_{i \in S} x_i$ where $S \subset \{1, \cdots, n\}$ is the parameter.

We first show that the VC=dimension $\geq n$. We choose the n "points" as the unit vector $e_i = 0 \cdots 010 \cdots 0$ where 1 appears in the i-th place. For any bit assignment $b_1, \cdots, b_n$ for the vectors $e_1, \cdots, e_n$. We choose the set

$$S = \{i : b_i = 1\}$$

Thus we get

$$\chi_S(x) = \begin{cases} 1 & j \in S \\ 0 & j \notin S \end{cases}$$

Thus, we conclude VC-dimension(C) $\geq n$.

Then we show that VC-dimension(C) $\leq n$. Since there are 2^n parity functions. We have VC-dimension $\leq \log 2^n = n$.

Example Convex polygons. In R^2 and points inside the convex polygon are positive and outside are negative.

Suppose we have no bound on the number of edges of the convex polygons C, and we want to show that VC-dimension $= \infty$; i.e. for any positive number d, we can find d points that can be shattering by C.

Given d, we choose S be a set of d points on the unit circle perimeter. We show that for every labeling of the points in S, there exists $c_t \in C$ that is consistent with the labeling. In fact, We can just choose c_t by connecting all the "+1" points. The polygon includes all the positive examples and none of the negative ones. Therefore, VC-dimension(C)= ∞.

VC-Dimension and the Number of Parameters

The VC-dimension thus give concreteness of the notion of the capacity of a given concept class. Intuitively, one might expect that learning machines with many parameters would have high VC-dimensions, while learning machines with few parameters would have low VC-dimensions. Although this is true in most cases, some counterexamples exist.

One striking counterexample demonstrates a learning machine with just one parameter, but with infinite VC-dimension. Define the step function $\theta(x), x \in R$: $\{\theta(x) = 1 \ \forall x > 0; \theta(x) = -1 \ \forall x \leq 0\}$. Consider the one-parameter family of functions, defined by

$$f(x, \alpha) \equiv \theta(sin(\alpha x)), x, \alpha \in R$$

You choose some number l, and present me with the task of finding l points that can be shattered. I choose them to be:

$$x_i = 10^{-i}, i = 1, \cdots, l$$

You specify any labels you like:

$$y_1, \cdots, y_l, y_i \in \{-1, 1\}$$

Then $f(\alpha)$ gives this labeling if I choose α to be

$$\alpha = \pi(1 + \sum_{i=1}^{l} \frac{(1 - y_i)10^i}{2})$$

Thus the VC-dimension of this machine is infinite.

2.3 VC-Dimension and Learnability

[1; 5] VC-dimension is one of the foundations of learning theory. It helps to answer some basic questions about learning, such as

1. Is a concept class learnable?
2. Can a concept class be learned efficiently?
3. How many training samples do we need?

Here we give some research results about these questions without proof. They are closely related to the PAC model in Section 2.1.

Definition The static learning algorithm has the following characteristics:

- The number of samples it ask for is $l(\epsilon, \eta)$
- It choses its hypothesis based on the sample it gets.

The main limitation of static learning algorithm is that the algorithm cannot update the number of examples necessary depending on the examples it receives. That is, it is not adaptive. This is closely related to the concept of distribution free that we mentioned for the PAC model. For static learning, we have the following theorem.

Theorem 2.2 *If a concept class C has VC-dimension ∞, then C is not learnable by any static learning algorithm.*

For static learning, as to the number of samples we need, we have the following theorem.

Theorem 2.3 *If C is a class with VC-dimension h, then*

$$l(\epsilon, \eta) = \Omega(\frac{h}{\epsilon})$$

Another important question about learning is about the efficiency of a learning algorithm. In [1], the polynomial learnability is classified into two groups: those with respect to the domain dimensions and those with respect to the complexity of target concept. For example, for the polynomial learnability with respect to domain dimension, we have the following theorem.

Theorem 2.4 *The concept class $C(n)$ is not polynomially learnable if the VC-dimension of $C(n)$ grows more than polynomially in n, where n is the dimension of the domain space.*

2.4 VC-Dimension and Generalization Performance

The generalization performance concerns the error rate of a learning machine on test data. Given a learning machine defined in Section 2.1, The expectation of the test error $R(\alpha)$ for a trained machine is defined as

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

where $D(\mathbf{x}, y)$ is the cumulative probability distribution that generating training and test samplings. Here the fact $\frac{1}{2}$ is because for each misclassification, $y - f(\mathbf{x}, \alpha) = 2$.

The quality of $R(\alpha)$ is called the *actual risk*. The *empirical risk* $R_{emp}(\alpha)$ is defined to be just the measured mean error rate on the training set $\{(\mathbf{x_1}, y_1), \cdots \mathbf{x_l}, y_l)\}$:

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(\mathbf{x_i}, \alpha)|$$

Now choose some η such that $0 < \eta < 1$. Then with probability $1 - \eta$, the following bound holds [2]:

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

where h is the VC-dimension of the learning machine and l is the number of training samples. This inequality defines a upper bound of actual risk and we call this upper bound "VC bound". The second term on the right hand side is called "VC confidence".

We note three key points about this bound. First, remarkably, it is independent of $D(\mathbf{x}, y)$. Second, it is usually not possible to compute the left hand side. Third, if we know h, usually we can easily compute the right hand side.

This bound gives a principled method for choosing a learning machine for a given task, and is the essential idea of the structural risk minimization (see Section 2.5). Given a fixed family of learning machine to choose from, to the extent that the bound is tight for at least one of the machines, one will not be able to do better than this. To the extent that the bound is not tight for any, the hope is that the right hand side still gives useful information as to which learning machine minimizes the actual risk. At present, for this case, we must rely on experiment to be the judge.

2.5 Structural Risk Minimization

We can now summarize the principle of structural risk minimization (SRM) [2]. This approach suggests when we do model selection, it's not enough to consider only empirical risk, since some machines may overfit the training data. Instead, we should consider the bound of the actual risk, e.g. VC bound in Eq. (1) or similar bounds.

Note that the VC confidence term in Eq. (1) depends on the chosen class of functions, whereas the empirical risk and actual risk depends on one particular function chosen by the training procedure. We would like to find that subset of the chosen set of functions, such that the risk bound for that subset is minimized. To do that we can divide the family of functions into a set of subsets where each subset has the same value of VC dimension h. We can do that since h only takes integer values. For each subset, we must be able to compute h, or to get a bound on h itself. Since all the functions with a subset have the same VC-confidence, it is enough to compare just the empirical risk of each machine. For each subset, we choose the one with minimum empirical risk. Then we get a series of machines, one for each subset. One then takes that trained machine in the series whose sum of empirical risk and VC confidence is minimal.

3 Case Study

In this section, we investigate two popular algorithms in machine learning community: decision tree and support vector machine. Through these examples, we want to demonstrate how VC-dimension can be used in practice.

3.1 Decision Tree

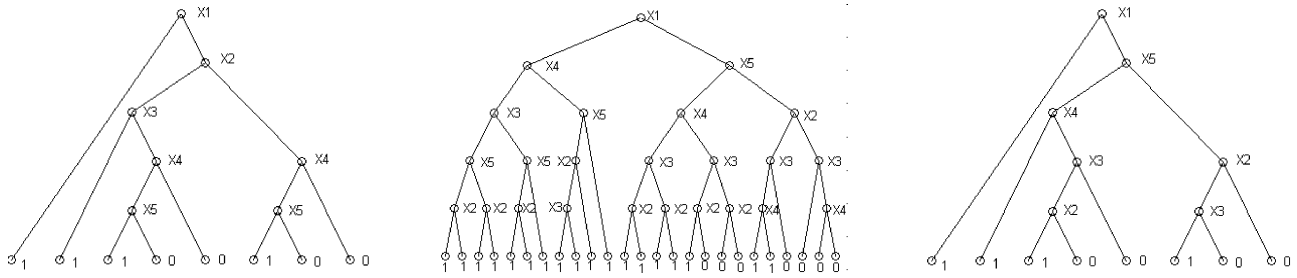
A decision tree takes as input a set of attributes and return a "decision"–the predicted output value for the input. Decision trees are expressive enough to represent any boolean function. Figure 2(a) gives an example. We will concentrate on boolean function induction although decision trees have been successfully used in many other domains including continuous function regression. The problem of decision tree induction on boolean functions is to induce an appropriate decision tree from a set of examples, where each example consists of a vector of input boolean attributes and a single boolean output value. In this section, we first explain how to build a decision tree based on the entropies of examples and then demonstrate using structural risk minimization to avoid overfitting. At the end, we state our conclusions.

Inducing Decision Tree from Examples

Our task is to build a decision tree that satisfies two conditions:

- First, we want the induced decision tree to be *consistent* with all the training examples (if there are no conflicts in the training data).
- Second, applying the idea of Occam's razor, we want to find the *smallest* tree.

In general, finding a *smallest* decision tree is an intractable problem. Fortunately, with some simple heuristics, we can do a good job in practice. The most widely-used heuristic is to test the most important attribute first. By "most important", we mean the one



(a) **(b)** **(c)**

Figure 2: *Decision trees in our experiments (a) Decision tree used for generating samples; (b) Decision tree before pruning; (c) Decision tree after pruning.*

that makes the most difference to the classification of an example. This way, we hope to get to the correct classification with a small number of tests, meaning that all paths in the tree will be small and the tree as a whole will be small.

One implementation of such a heuristic is called *ID3* algorithm, which is based on the concept of entropy from the information theory. Suppose we have a sample set S and each sample belongs to one of the two classes, $\oplus$ and $\ominus$. Let $p_\oplus$ and $p_\ominus$ the proportions of positive and negative examples in S, respectively. Then the entropy of S is

$$p_\oplus(-\log p_\oplus) + p_\ominus(-\log p_\ominus)$$

Then when we pick a attribute A and divide S based on A, the information gain is computed as

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

At each step of ID3 algorithm, we greedily pick the attribute that has the biggest information gain. Both theoretical analysis and empirical results have shown that the decision trees generated by ID3 are approximately "shortest trees".

Decision Tree Pruning

If there is no noise in the training data, the heuristic discussed above works well. But in practice most training data are not noise-free and ID3 algorithm may overfit the training data. Thus we want to prune unnecessary nodes of the resulting trees. This problem is actually a model selection problem in that we have to make trade-off between the accuracy on the training data and the sizes of the trees. Here we develop an algorithm to prune decision trees based on the idea of structural risk minimization (see section 2.5). Note that there are some other ways to do pruning, e.g. cross-validation and C4.5, and a few of them may be much more efficient than structural risk minimization. Our goal here is just to show how to do that using the idea of VC-dimension.

To use structural risk minimization, we need to figure out a way to compute the VC-dimension h of a given decision tree. It has been shown that h of a decision tree is roughly the number of nodes (not including the leaves). [1]

For example, the VC-dimension of the tree in Figure 2 is 7. Thus we also call the VC-dimension h the size of the tree.

Although computing VC-dimension and VC confidence is simple in this case, the main problem is finding for every h the best decision tree T_h of size h. That is, for any given size h, we want to find the decision tree with size h and minimum empirical risk. An naive way to do that is for each size h, enumerating all the possible trees and finding the best one. This method obviously has exponential complexity.

In our work, we developed an algorithm based on the idea of dynamic programming that can do it much more efficiently.

The algorithm of is shown in Table 3.1. Now let us analyze its complexity. Let n and d the size and depth of the tree before pruning, respectively. Let l the number of training samples.

Experiments and Results

We implement the ID3 algorithm for inducing a decision tree and the dynamic programming algorithm for pruning the tree. The decision tree we use for generating the samples is shown in Figure 2(a). We generate 1000 training samples, 5% of them are noisy data (i.e. their classes are incorrect). The decision tree generated by ID3 is displayed in Figure 2(b).

Then we use dynamic programming to prune trees. For each possible size of tree, we find the tree with minimum training errors. Thus we get a series of trees with different sizes. We compute the VC bound for each size based on Eq. (1). The resulting curve is shown in Figure 3. For comparison, we generate another 1000 test samples in the same way. We measure the actual risk (test error) using the series of trees with different sizes. The result is also shown in Figure 3. Note that the two curves have minima at the

[1] We have seen a proof that shows the VC-dimension of a decision tree is at least the number of nodes [6]. Some literature mentioned that in practice, we usually just use the node number as its VC-dimension [8].

```
function R = prune-tree(T,S)
T: input, decision tree to prune
S: input, training samples
R: output, 3-column table, for each row i
  R(i,1) is the number of nodes to prune
  R(i,2) is the minimum num of misclassified
    samples among all the trees with the
    given size
  R(i,3) the tree achieving the value R(i,2)

  if size(T) = 1
    set R with 2 rows, corresponding
    to prune the only node or not;
    return;
  else
    root=get-root(T);
    T1=left-subtree(T);
    S1=filter-sample(S,root=true);
    R1=prune-tree(T1,S1);
    T2=right-subtree(T);
    S2=filter-sample(S,root=false);
    R2=prune-tree(T2,S2);
    for nodesToPrune=0 : size(T)
      B(nodesToPrune,1)=nodesToPrune;
      B(nodesToPrune,2)=∞;
      for lNodesToPrune=0 : nodesToPrune
        rNodesToPrune=nodesToPrune
-           lNodesToPrune
        if B1(lNodesToPrune,2)+
          B2(rNodesToPrune,2) <
          B(nodesToPrune,2)
          B(nodesToPrune,2)=
          B1(lNodesToPrune,2)+
          B2(rNodesToPrune,2);
          B(nodesToPrune,3)=merge − subtree();
        end if;
      end for;
    end for;
  end if;
end function
```

***Table 1:** Algorithm of dynamic programming for pruning*

same place of size 7. Thus, from the idea of structural risk minimization, we will select the size with lowest VC bound and this size is exactly where actual risk becomes minimum.

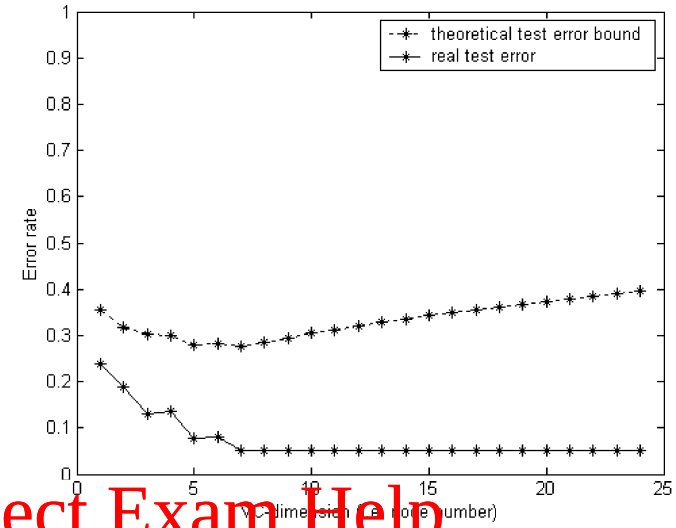


***Figure 3:** The predictivity of VC bound*

One thing to point out is that in the Figure 3. The curve of actual risk stays flat after reaching the minimum. That's because the decision tree we used in our experiment is very simple. Therefore, although the noisy data will make the tree induced over-complex, it will not make wrong decision since at the leaves the classification is determined by majorities.

The optimal tree after pruning is displayed in Figure 3(c), whose size is 7. It can be easily verified that this tree is equivalent with the generating the samples.

3.2 Support Vector Machine

Support Vector Machines (SVM) are learning systems that use a hypothesis space of linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory. This learning strategy introduced by Vapnik and co-workers is a principled and very powerful method that in the few years since its introduction has already outperformed most other systems in a wide variety of applications. In this section, we give a brief introduction of SVM and we will focus on the application of VC-dimension in this approach.

Structure of SVM

[3; 2] If given a classification task, SVM will first map the data from the input space into a feature space by a mapping function vector. $\mathbf{\Phi} : \mathbf{R}^d \mapsto \mathbf{F}$. By this mapping, the data in the feature space are separable by a hyperplane, but in the input space usually are not.

The dimension of the feature space is different from the dimension of input space, usually is much higher. Then, SVM will use a linear learning machine to do the classification in the feature space. A linear learning machine is used to find a hyperplane $y = \mathbf{w} \bullet \Phi(\mathbf{x}) + b$, with the constraints

$$\mathbf{w} \bullet \Phi(\mathbf{x_i}) + b \geq +1, \text{ for } y_i = +1$$
$$\mathbf{w} \bullet \Phi(\mathbf{x_i}) + b \leq -1, \text{ for } y_i = -1$$

Then the classification function we learn is

$$f(\mathbf{x}, \alpha) = sign(\mathbf{w} \bullet \Phi(\mathbf{x}) + b)$$

where $\alpha = [\mathbf{w}, b]$.

The objective of training is to find the learnable parameters according to the given l observations. According to the geometry, the vector $\mathbf{w}$ is the normal of the hyperplane, the b is the bias. We define a margin as the distance between hyperplane $H1$ and $H2$. $H1$ and $H2$ are the tight boundaries for the data and the normal of both of them is $\mathbf{w}$.

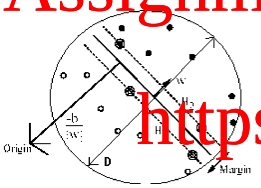


Figure 4: *Linear Learning Machine*

The solution of SVM always maximizes the margin. So this optimization problem could be formulated by Lagrangian representation with introducing a set of positive Lagrange multipliers α_i. The constraints are derived from the Karush-Kuhn-Tucker conditions. Finally, it turns out to be a quadratic programming (QP) problem by using its dual formulation

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\Phi(x_i) \bullet \Phi(x_j)) \quad (2)$$

where $\sum_i \alpha_i y_i = 0$. From (2), we can find the term $(\Phi(\mathbf{x_i}) \bullet \Phi(\mathbf{x_j}))$ actually is a function of $\mathbf{x_i}$ and $\mathbf{x_j}$, we can calculate the inner product without knowing the mapping function Φ. This property can save us a lot of computational time, since, for most of the times, the dimension of the feature space is very high, even infinite, so that to calculate Φ is at least very expensive. We define this function as a kernel function, $K(\mathbf{x_i}, \mathbf{x_j})$. Not all kinds of functions can be a kernel function. The Mercer's theorem is used to verify whether a function is a kernel. The frequently used kernels have, for example:

With the introduced kernel method, the solution of the SVM is:

$$\|\mathbf{w}\|^2 = \alpha^T H \alpha H_{(i,j)} = y_i y_j K(\mathbf{x_i}, \mathbf{x_j})$$

.

In the testing phase, we are using another set of observations to test the generalization performance of our SVM.

$$f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x_i}, \mathbf{x})$$

. An error will be detected if $f(\mathbf{x}) \neq y$

Furthermore, even in the feature space, the data are still non-separable. In this case, QP has no feasible solution.

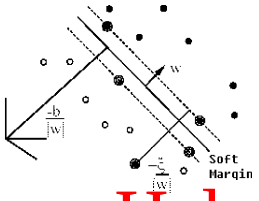


Figure 5: *Soft margin and slack variable*

The concept of Soft Margin is introduced to deal with this problem. A soft margin is a margin that can tolerant some training errors to some extent Figure 5. This is done by introducing positive slack variables ξ_i, i = 1, ..., l. $C \sum_i \xi_i$ reflects the upper bound on the number of training errors. Hence the objective function is now to minimize

$$\frac{\|\mathbf{w}\|^2}{2} + C(\sum_i \xi_i)^2$$

. where C is a parameter giving the penalty to errors. The larger is the C the bigger the penalty and the less the errors that soft margin can tolerant. This change reflected in the QP form is limit the α_i with upper bound of C.

Generalization performance

To get the best generalization performance is one basic goal of a learning machine. There are several theories applicable: Bayesian evidence framework, VC dimension theory and cross validation [4; 9]. In this project, we studied the VC dimension theory. The general idea is still following the structural risk minimization, i.e. we will use Structural Risk Minimization to tune our SVM to get the best generalization performance.

For SVM, the original VC-dimension doesn't work well. Vapnik proved that the VC-dimension of a SVM is

$$h = dim(\mathbf{F}) + 1$$

Kernel name	Definition	Dim. of F	VC-dim.
linear	$\mathbf{x} \bullet \mathbf{v}$	d	d+1
polynomial	$(\mathbf{x} \bullet \mathbf{v} + 1)^p$	$\binom{d+p-1}{p}$	$\binom{d+p-1}{p} + 1$
Gaussian RBF	$e^{\frac{\|\mathbf{x}-\mathbf{v}\|^2}{2\sigma^2}}$	inf	inf

Table 3.2 shows the VC dimension of some kernel.

Mostly, the VC-dimension of the SVM is very high, even infinite. So the value of the VC-bound of the structural risk is also very high and even infinite. However even with an infinite error bound, the actual performance of the SVM is still very good. That is because VC-dimension is the measurement of the capacity of the SVM on *any* distribution of the training sample. It is distribution free. But for a given training task, the distribution is given. So the SVM will not reach its capacity fully when training. And according to PAC model, the distribution of the testing sample is the same as the training sample too, which means the testing will not reach the capacity fully either. Vapnik introduced an effective VC dimension, which is a measurement of the capacity of the SVM and, at the same time, the sample complexity [5].

Definition Let F be a class of real-valued functions defined on a domain X. We say a set of points $\{x_1, x_2, ..., x_i\} \in \mathbf{X}^l$ is γ-shattered by F, if there exist real numbers r_i, such that for every binary classification $y \in \{-1, 1\}^l$, there exists $f_b \in \mathbf{F}$, such that

$$f_b(x_i) = \begin{cases} \geq r_i + \gamma & b_i = 1 \\ \leq r_i - \gamma & b_i = -1 \end{cases}$$

The fat-shattering dimension $fat_F(\gamma)$ is the size of the largest γ-shattered subset of X. This fat-shattering dimension is the effective VC-dimension. It can be bounded as [10]:

$$h^* \leq min((\frac{D}{\gamma})^2 + 1, l)$$

where D is the diameter of the Minimal Enclosing Sphere, which covers all the training data points in the feature space as seen in Figure 4. γ is the margin. And a new VC-bound is generated by Vapnik. We would like to quote without proof.

Theorem 3.1 *For optimal hyperplanes passing through the origin, we have [2]*

$$E[P(error)] \leq \frac{E[D^2/\gamma^2]}{l}$$

where $P(error)$ is the probability of error on the test set, the expectation on the left is over all training sets of size $l-1$, and the expectation on the right is over all training sets of size l.

From this theorem, Burges argued that minimizing D^2/γ^2 could be expected to give better generalization performance.

Experiment1

The objective of this experiment is to implement the idea Burges suggested choosing a kernel with appropriate parameter to minimize D^2/γ^2. Firstly, in this experiment, we will measure the new VC-bound of a series of SVM with polynomial kernel of parameter p, in this experiment, $p = 1...30$. Secondly, we will find the appropriate p by minimal VC-bound so as to obtain the best generalization performance. Finally, the test data are used to measure the test errors. It is used to argue that the method suggested by Burges is valid and reasonable for tuning SVM. The training sample is faked by hand to keep a particular shape. To keep the consistency of the distribution of the training sample and test sample, we add the random noise to the training data to get a larger training data set and the testing data set. The size of the training set is 14,100. The size of the testing set is 141.

We implement the method to calculate the D. We wish to minimize R^2 subject to

$$\|\Phi(\mathbf{x_i} - \mathbf{B}\|^2 \leq \mathbf{R}^2 \; \forall i$$

It turned out to be another quadratic programming problem. Maximize

$$L_D = \sum_i \lambda_i K(\mathbf{x_i}, \mathbf{x_j}) - \sum_{i,j} \lambda_i \lambda_j K(\mathbf{x_i}, \mathbf{x_j}) \sum_i \lambda_i = 1 \lambda_i \geq 0$$

The solution is $\mathbf{B} = \sum_i \lambda_i \Phi(\mathbf{x_i})$ so

$$D = 4R^2 = 4 \max_i (K(\mathbf{x_i}, \mathbf{x_i}) - 2 \sum_j \lambda_j K(\mathbf{x_i}, \mathbf{x_j}) + \lambda^T K \lambda) \quad (3)$$

.

1. From Figure 6(b), we can see the minima of the VC-bound at the same place as the minima of the actual risk, which is $p = 6$. In this case, the minimization of the VC-bound works well to find the appropriate kernel to build the SVM.

2. As a comparison, we measure another risk bound, support vector bound, which is defined by

$$R \leq \frac{\text{number of support vector}}{l}$$

Actually, for this case, the risk grows bigger beyond the $p = 6$, while this bound keeps decreasing. So minimizing this bound will not give good result.

3. When measuring the VC-bound, we divide the sample set into 10 pieces; the VC-bound is the expectation of the VC-bound is done over these 10 pieces. It will cost too much time if to do all sample size. And this also gives a reasonable result. But maybe it is not very correct and accurate. Anyway the best way is to do the expectation over all sample size.

Figure 6: *(a) VC-bound, SV-bound and actual risk (b) VC-bound and scaled actual risk.*

4. In some sense, if the kernel is fixed and the training task is given, then to minimize the VC-bound is equivalent to maximize the margin. This gives an explanation of why optimal solution is of maximal margin.

5. For polynomial kernel with small degree like 1 or 2, this training sample actually is non-separable. So there will be a training error term added to the VC-bound according to the structural risk definition. When the degree is large enough, about 5 in this experiment, the training error vanishes.

Experiment2

For non-separable case, the SVM has another parameter C could be tuned to minimize the VC-bound. This experiment is designed to do the SVM tuning by choosing C according to structural risk minimization. Firstly, a RBF kernel SVM is taken to classify a non-separable training data with a series of different C value. The kernel parameter is fixed, $\sigma = 5$. Secondly, the VC-bound is measured, and the C with the minimum VC-bound will be found. Finally, the testing error is measured, and the minimum place will be found and compared with the C obtained in the previous step.

In this non-separable case, the VC-bound will count on the training errors according to the structural risk definition. Vapnik gives the bound:

$$R \leq \frac{D^2 + (\sum_i \xi_i)^2}{M^2} / l$$

where $(\sum_i \xi_i)^2$ reflects the training errors.

1. The C we tested is 10,1000,10000,10000,infinite. The result shows $C = 1000$ is the best solution.
2. The actual risk shows $C = 1000$ is best too.

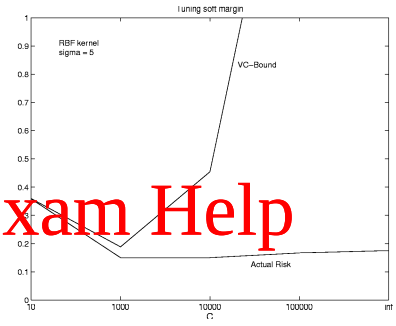


Figure 7: *Soft margin and risk bound*

4 Conclusion

VC-dimension provides a quantitative way for measuring the capacity of a learning machine. Since the capacity is a very important concept in machine learning, VC-dimension thus becomes a very useful tool. Structural risk minimization, which is based on the VC-bound, suggests a feasible framework for model selection. Our experiments on decision tree and support vector machine demonstrate the applications of VC-dimension in machine learning.

5 Acknowledgment

We would like to thank Prof. Richard Ladner for his instructions and encouragements during the project. We are also grateful to Prof. Raj Rao for lending us a book on support vector machine.

References

[1] A.Ehrenfeucht A. Blumer and D.Haussler. Learnablility and the vapnik-chervonenkis dimension. *Journal of the ACM*, 36:929–965, 1989.

[2] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 1998.

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

[4] A.Boni D.Anguita and S.Ridella. Evaluating the generalization ability of support vector machines through the bootstrap. *Neural Processing Letters*, 2000.

[5] M. J. Kearns. *The Computational Complexity of Machine Learning*. Cambridge University Press, 1990.

[6] M. J. Kearns. *An introduction to Computational Learning Theory*. 1994.

[7] L.G.Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[8] Andrew Moore. Tutorial on vc-dimension for characterizing classifiers. `http://www-2.cs.cmu.edu/ awm/tutorials/`.

[9] M. Cheriet N. E. Ayat and C.Y. Suen. Optimization of the svm kernels using an empirical error minimization scheme. *Lecture Notes in Computer Science*, 2388, 2002.

[10] V. N. Vapnik. *The Nature of Statistical Learning Machine*. Springer-Verlag Press, 1995.

[11] V. N. Vapnik. An overview of satistical learning theory. *IEEE Transaction on Neural Networks*, 10:988–999, 1999.

[12] V.Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.