Generalization bounds for incremental search classification algorithms

Yoram Gat
University of California, Berkeley

Technical report No. 575 (March 2000)

Abstract

This paper presents generalization bounds for a certain class of classification algorithms. The bounds presented take advantage of the local nature of the search that these algorithms use in order to obtain bounds that are better than those that can be obtained using VC type bounds. The results are applied to well-known classification algorithms such as classification trees and the perceptron.

1 Introduction

1.1 The classification problem setup

I consider the classical problem of learning a classifier from examples which can be formalized as follows: Let $\mathbf{P}$ be an unknown distribution over $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. Let $Z = (X, Y)$ and $Z_i = (X_i, Y_i)$, $i = 1, 2, \ldots$ be iid random variables with distribution $\mathbf{P}$. The problem is predicting Y given $Z_1, \ldots, Z_l$ and X.

The set of variables $Z_1, \ldots, Z_l$ is referred to as the training set, and will be denoted by $\mathcal{S}$. $\mathcal{S}$ will also be used to denote the field spanned by the training set, and the empirical distribution of the training set.

A solution to the problem of learning from examples is a function mapping the training set into $\mathcal{C}$, which is some set of classifier functions:

$$M : \mathcal{Z}^l \to \mathcal{C}. \tag{1}$$

Each $c \in \mathcal{C}$ is a function $c : \mathcal{X} \to \mathcal{Y}$. Thus the prediction for Y is $c^M(X)$ where $c^M = M(Z_1, \ldots, Z_l)$. The function M is often referred to as a classification algorithm (CA).

There is a one-to-one correspondence between the classifiers in $\mathcal{C}$ and error indicator functions $f_c : \mathcal{Z} \to \{0, 1\}$, where

$$f_c(x, y) = \mathbf{1}\left\{c(x) \neq y\right\}.$$

Denote $\mathcal{F} = \{f_c : c \in \mathcal{C}\}$ and $f^M = f_{c^M}$.

The quality of the classifier c^M, for a given training set, may be measured using the expected error rate (also called expected risk):

$$\mathbf{E}_{\mathbf{P}} f^M = \mathbf{P}\left(c^M(X) \neq Y \,\middle|\, \mathcal{S}\right).$$

AMS 1991 subject classifications. Primary 62H30.
Key words and phrases. Generalization ability, classification trees, perceptron algorithm.

The solution M is usually geared toward finding a classifier which has low empirical error rate (also called empirical risk):

$$\mathbf{E}_{\mathcal{S}} f^M = \frac{1}{l} \sum_{i=1}^{l} \mathbf{1}\left\{ f^M(X_i) \neq Y_i \right\}.$$

Therefore, it is often desirable to be able to obtain bounds for the difference between the empirical and the expected error rates, which is called the overfit of the algorithm and denoted $\mathbf{E}_{\mathbf{P}-\mathcal{S}} f^M$. The behavior of the difference will depend on the underlying, unknown probability measure. The term generalization ability is used to describe the worst-case behavior of the overfit of a specific algorithm. The smaller the probability for a large overfit, the better is the generalization ability of the algorithm.

1.2 Incremental search classification algorithms

One map M commonly used is

$$M(Z_1, \ldots, Z_l) = \arg\min_{c \in \mathcal{C}} \mathbf{E}_{\mathcal{S}} f_c.$$

This is known as the Empirical Risk Minimization (ERM) method. It has been shown that the generalization ability of the algorithm can be determined by using the VC dimension of the set of functions $\mathcal{F}$ ([1] sec. 4.9).

Frequently, it is difficult to carry out the minimization search over the whole range of classifiers considered. In these situations, an incremental search is sometimes carried out. To implement an incremental search, a measure of proximity over the space of classifiers is defined. The incremental search classification algorithm (ISCA) starts by considering some fixed classifier inside the classifier space, and proceeds by considering classifiers that are in proximity to the this starting point, and then classifiers that are in proximity to them, and so on. The algorithm thus proceeds in steps. The more steps that are being taken, and the larger the number of classifiers that are in proximity to any given classifier, the larger is the search.

One type of ISCAs is that of the greedy search classification algorithm. A greedy search classification algorithm searches among the classifiers that are in proximity to the starting point for the classifier with the lowest empirical error. It then searches the classifiers that are close to this classifier for the one that has the lowest empirical error, and so on. Other ISCAs may look a few steps ahead before deciding where to search next.

ISCAs are usually less expensive computationally to implement than exhaustive searches, but they have the drawback that they may not find the global minimum of the empirical error. However, in the presence of uncertainty, such as exists when training classifiers from examples, the fact that incremental searches examine only a small part of the search space, causes a reduction in the overfit that the algorithm may produce. That is, while the empirical error rate achieved may be larger than the global minimum of the empirical error rate, the classifier found may have a lower expected error rate.

This paper lays out a procedure for estimating the generalization ability, i.e., of bounding the overfit, of ISCAs. This procedure can achieve tighter bounds than would be possible for global or exhaustive search algorithms.

Many of the ideas presented in this paper are present in [3] and in [4]. The treatment here is somewhat different, and includes handling the technical difficulty of analyzing continuous classifier spaces.

2 Graph theoretical trees

This section defines a few concepts that will be used in the discussion of ISCAs.

A (graph theoretical) graph is a set of nodes V, and a set of edges, E. Each edge e is a couple of nodes $e = \{u, v\}$, $u, v \in V$.

A path on a graph is a sequence of distinct edges so that any two consecutive edges have a common node:

$$\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}.$$

A (graph theoretical) rooted tree is a graph that has special root node v_0, and which has a unique path from the root node to any other node in the graph.

On a rooted tree any node v can be designated with a level, $\mathrm{lev}(v)$. The root node has level 0, while the level of any other node is the length of, i.e., the number of edges in, the path from the root to the node.

For any node v on a rooted tree, define:

- The set of offspring of the node:

$$\mathrm{offsp}(v) = \{u \in V : \{v, u\} \in E,\ \mathrm{lev}(u) > \mathrm{lev}(v)\}.$$

- The parent of the node (for all nodes except for the root node v_0):

$$\mathrm{parent}(v) = u, \text{ such that } \{v, u\} \in E,\ \mathrm{lev}(v) > \mathrm{lev}(u).$$

- The set of ancestors of the node:

$$\mathrm{ancest}(v) = \{u \in V \setminus \{v\} : \{u, u'\} \in \text{the path from } v_0 \text{ to } v\}.$$

- The set of descendants of the node:

$$\mathrm{descend}(v) = \{u \in V : v \in \mathrm{ancest}(u)\}.$$

A node whose offspring set is empty is called a terminal node, or a leaf.

3 A tree node numbering procedure

This section presents a natural way to assign a number, $n(v)$, to each node v on a rooted tree. The numbering which results has the following properties:

- The number $n(v)$ of any node v depends only on the number of offspring of the nodes in the path from the root to the node v.
- $|\{v : n(v) \leq k\}| \leq k$.
- If the tree has no leaves, then
$$\lfloor k/d \rfloor \leq |\{v : n(v) \leq k\}|,$$
where $d = \max_{\{v : n(v) \leq k\}} |\text{offsp}(v)|$.

Definition 1 *For any rooted tree, $T = (V, E, v_0)$, define the a numbering, n_{v_0}, of the node set recursively as follows:*

$$\begin{aligned} n_{v_0}(v_0) &= 1, \\ n_{v_0}(v) &= 1 + |\mathit{offsp}(v_0)| \cdot n_{v_1}(v), \end{aligned}$$

where v_1 is the offspring of v_0 in the path from v_0 to v.

For each node, v, its number, $n_{v_0}(v)$, can be written up explicitly in terms of the number of offspring of the ancestors of v. Let $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v\}$ be the path from v_0 to v. Then:

$$\begin{aligned} n_{v_0}(v) = \; & 1 + |\text{offsp}(v_0)| + \\ & |\text{offsp}(v_0)| \cdot |\text{offsp}(v_1)| + \cdots + \\ & |\text{offsp}(v_0)| \cdot |\text{offsp}(v_1)| \cdots |\text{offsp}(v_{k-1})| \,. \end{aligned}$$

By the definition of n, the first property above holds. Lemma 1 states the second property, and its proof follows:

Lemma 1 *For any $k = 1, 2, \ldots$ and for any $v \in V$, define*

$$C_k^v = \{u \in \mathit{descend}(v) : n_v(u) \leq k\}.$$

The following inequality holds:

$$|C_k^v| \leq k.$$

Proof: By induction on k. The statement holds for $k = 1$. Assume that the statement holds for all $k' < k$.

$$C_k^v = \bigcup_{u \in \text{offsp}(v)} \{u' \in \text{descend}(u) : n_u(u') \leq (k-1)/\,|\text{offsp}(v)|\} \cup \{v\},$$

Using the induction hypothesis,

$$\left|\left\{u' \in \mathrm{descend}(u) : n_u(u') \le \frac{k-1}{|\mathrm{offsp}(v)|}\right\}\right| \le \frac{k-1}{|\mathrm{offsp}(v)|}.$$

The lemma follows. □

The third property is not used here. It is, however, of interest, since the bounds in the paper cannot be tight if $|\{v : n(v) \le k\}| \ll k$. The third property of the numbering, therefore, guarantees that for leafless trees, such as those discussed in this paper, this is not the case.

4 Sieve structures and simultaneous confidence regions

4.1 General sieve structures and bounds

When a classification algorithm has a rich enough range $\mathcal{C}$, it may not be possible to obtain bounds of the form

$$\mathbf{P}\left(\sup_{f\in\mathcal{F}} \mathbf{E}_{\mathbf{P}-\mathcal{S}} f > \epsilon\right) \le \delta(\epsilon),$$

where $\delta(\epsilon)$ is small for small ϵ.

It is, however, often useful to obtain a somewhat weaker, but still interesting bound by defining a sieve structure over $\mathcal{C}$:

$$\mathcal{C} = \bigcup_{i\ge 1} \mathcal{C}_i.$$

The spaces $\mathcal{C}_i$ are formed to be small enough so that, for each i, a bound of the type

$$\mathbf{P}(\sup_{c\in\mathcal{C}_i} \mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > \epsilon) \le \delta_i(\epsilon),$$

exists, where $\delta_i(\epsilon)$ are small for small ϵ, but not uniformly in i.

These separate bounds imply a simultaneous bound:

$$\mathbf{P}(\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > \epsilon_{i(c)}(\delta_{i(c)}) \text{ for some } c \in \mathcal{C}) \le \sum_{i\ge 1} \delta_i,$$

where δ_i, $i = 1, 2, \ldots$ is a sequence of constants, ϵ_i, $i = 1, 2, \ldots$ are the inverse functions of δ_i, and for every $c \in \mathcal{C}$, $i(c)$ is some member of $\{i : c \in \mathcal{C}_i\}$.

4.2 A sieve structure for a countable set of classifiers

One way in which the sieve structure can be used is to split an infinite set of classifiers into a sequence of finite sets. Let $\mathcal{C} = \bigcup_{i\ge 1} \mathcal{C}_i$ be a countable set of classifiers, where $|\mathcal{C}_i| < N_i$. Further assume that it is known that for each given c, $\mathbf{P}(\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > \epsilon) \leq p(\epsilon)$. Then for each i,

$$\mathbf{P}\left(\sup_{c \in \mathcal{C}_i} \mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > \epsilon\right) \leq N_i p(\epsilon).$$

For this case, $\epsilon_i(\delta) = p^{-1}(\delta/N_i)$.

4.3 A sieve structure for a tree of classifiers

By using the numbering procedure of section 3, the bound of the previous sub-section can be directly applied to a set of classifiers that has a rooted tree structure associated with it, i.e., a set of classifiers which are the node set of a tree.

Let N_i, $i = 1, 2, \ldots$ be a sequence of constants and let $n(c)$ be the numbering of the classifiers induced by a tree structure. Then,

$$\mathbf{P}\left(\sup_{c : n(c) \leq N_i} \mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > \epsilon\right) \leq N_i p(\epsilon).$$

Setting $i(n) = \min\{i : N_i \geq n\}$ a simultaneous bound is obtained:

$$\mathbf{P}\left(\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > p^{-1}\left(\frac{\delta_{i(n(c))}}{N_{i(n(c))}}\right) \text{ for some } c \in \mathcal{C}\right) \leq \delta. \tag{2}$$

The results of this paper are all bounds of the type given by expression (2). To obtain such a bound for a particular CA, M, it is sufficient to arrange the classifiers in the set $\mathcal{C}$ in a tree structure. The bound is then implied by the numbering of the classifiers on the tree.

4.4 A sieve structure for a tree of sets of classifiers

A slight generalization of the situation in the previous sub-section results when each node in the tree, v, corresponds to a finite set of classifiers $\mathcal{C}_v$. This situation arises during the construction of the trees of classifiers.

The simultaneous bound in such a situation is

$$\mathbf{P}\left(\sup_{c \in \mathcal{C}_v} \mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > p^{-1}\left(\frac{\delta_{i(n(v))}}{N_{i(n(v))}\,|\mathcal{C}_v|}\right) \text{ for some } v \in V\right) \leq \delta. \tag{3}$$

Since the expression for the bound in (3) is unwieldy, I introduce the abbreviation

$$\epsilon_p(n, k) = p^{-1}\left(\frac{\delta_{i(n)}}{N_{i(n)} k}\right).$$

Thus, in this notation, the bound in (3) is

$$\mathbf{P}\left(\sup_{c \in \mathcal{C}_v} \mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > \epsilon_p\left(n(v), |\mathcal{C}_v|\right) \text{ for some } v \in V\right) \leq \delta. \tag{4}$$

Below, I also use the further abbreviation

$$\epsilon_p(n) = \epsilon_p(n, 1) = p^{-1}\left(\frac{\delta_{i(n)}}{N_{i(n)}}\right),$$

giving the following form for inequality (2):

$$\mathbf{P}\left(\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c > \epsilon_p(n(c)) \text{ for some } c \in \mathcal{C}\right) \leq \delta. \tag{5}$$

4.5 The Hoeffding bound

As a more concrete example of a bound, consider the case where the function p is the Hoeffding bound

$$\mathbf{P}(\mathbf{E}_{\mathbf{P}-\mathcal{S}} f) \leq \exp -2l\epsilon^2 = p(\epsilon),$$

and

$$\begin{aligned} N_i &= N_0 r^i, \\ \delta_i &= \begin{cases} \delta/d & i = 0, 1, \ldots, d-1 \\ 0 & i \geq d \end{cases} . \end{aligned}$$

For this particular example, the simultaneous bound (2) (or equivalently (5)) implies the UCB:

$$\mathrm{UCB}(\mathbf{E}_{\mathbf{P}} f^M) = \begin{cases} \mathbf{E}_{\mathcal{S}} f^M + \sqrt{\frac{1}{2l}\log(n(c^M) r d/\delta)} & \text{when } n(c^M) \leq N_0 r^{d-1} \\ 1 & \text{otherwise} \end{cases} .$$

5 ISCAs as tree search algorithms

5.1 The full and partial search trees

It is convenient to describe ISCAs as algorithms that search trees of classifiers. The root node corresponds to the starting point classifier. The level 1 nodes correspond to the classifiers in proximity to the starting point classifier, the level 2 nodes are those classifiers that are in proximity to them, and so on. I call this tree the full search tree.

While the same classifier may appear on the full search tree several times, it may be assumed, without loss of generality, that each classifier is unique. This may be achieved, if necessary by attaching unique labels to the nodes of the tree. This approach is used here, in order to avoid ambiguities.

The full search tree, however, is often a tree in which each node has a large number of offspring. Furthermore, while some of the offspring have a relatively high probability of being used - in the sense that there is a high probability that the selected classifier will be a descendant of these offspring, most of the offspring have have a very low probability of being used. The offspring which have a low probability of being used have a low impact on the amount of overfit of the algorithm, but unless eliminated from the tree, will increase the overfit bound.

It is, therefore, necessary to define another tree, the partial search tree, in such a way that its coverage probability, i.e., the probability of the selected classifier being on the partial search tree is high, but not 1, and that the number of offspring of every node in the partial search tree is as small as possible.

Since the node and edge sets of the partial search tree are subsets of those of the full search tree, the subscripts full and ∂ will be used to distinguish between properties of the different trees.

Let $T_{\text{full}} = (V_{\text{full}}, E_{\text{full}}, c_0)$ be a full search tree. $T_{\text{part}} = (V_{\text{part}}, E_{\text{part}}, c_0)$ - a partial search tree - is determined by its node set, V_{part}, which is a subset of V_{full}, and which always contains c_0. The edge set of the partial search is

$$E_{\text{part}} = \left\{ \{u, v\} \in E_{\text{full}} : u, v \in V_{\text{part}} \right\}.$$

5.2 A definition template of partial trees

This sub-section describes a template for the definition of partial trees, for various CAs.

The template assumes that there exists a family of events

$$\mathcal{A} = \{A(c, n) : c \in \mathcal{C},\ n \in \mathcal{N}\},$$

with the property that for any sequence of classifiers $c_1, c_2, \ldots$, and for any sequence of natural numbers $n_1, n_2, \ldots$, such that $|\{n_i : n_i \leq N\}| \leq N$, it holds that

$$\mathbf{P}\left(\bigcap_{i=1}^{\infty} A(c_i, n_i)\right) \geq 1 - \delta. \tag{6}$$

This family is referred to as a tree building event family (TBEF).

Using the TBEF $\mathcal{A}$, the node set of the partial tree is defined recursively:

- $c_0 \in V_{\text{part}}$.
- $c \in V_{\text{part}}$ if
  a. $c' = \text{parent}_{\text{full}}(c) \in V_{\text{part}}$, and,
  b. $\{c', c\}$ is in the path of M, for some sample point in the event $A(c', n_{\text{part}}(c'))$.

Note that since $n_{\text{part}}(c)$ depends only on the number of offspring of the ancestors of c in the partial tree, the numbering n_{part} is well defined.

The partial tree will contain c^M in the event

$$A = \bigcap_{c \in V_{\text{part}}} A(c, n_{\text{part}}(c)).$$

Therefore, by lemma 1 and by inequality (6), $P(c^M \in V_{\text{part}}) \geq 1 - \delta$.

6 Application to a projection histogram algorithm

This section presents a somewhat artificial example of a CA whose overfit can be analyzed using the setup described above.

Let $\mathcal{X} = \{0,1\}^d$ and $\mathcal{Y} = 0, 1$. Let the space of classifiers considered be

$$\mathcal{C} = \left\{ c_{i(1),\ldots,i(r),\mathbf{y}} : r = 1, \ldots, d,\, 1 \le i(1), \ldots, i(r) \le d,\, \mathbf{y} \in \mathcal{Y}^{2^r} \right\},$$

where

$$c_{i(1),\ldots,i(r),\mathbf{y}}(x) = y_{x_{i(1)},\ldots,x_{i(r)}}.$$

That is, each classifier in the set classifies a point x according to its projection onto an r dimensional space. The classifiers are indexed according to the dimensions of projection and according to the classification labels assigned to the various values in the projected space.

Consider the projection histogram CA, M, which works in the following way:

- Given the dimensions of projection, $i(1), \ldots, i(r)$, M picks the vector $\mathbf{y}$ that minimizes the empirical risk (with arbitrary choice if there is more than one such vector).

- The projection dimensions are picked one by one and in a greedy fashion. That is, having picked dimensions $i(1), \ldots, i(k)$, the next dimension to be picked would be the dimension that will give the largest reduction in the empirical error.

Define the full search tree with the nodes corresponding to the classifiers in $\mathcal{C}$, and a dummy root node c_0 which classifies all $x \in \mathcal{X}$ as 0. Let edges join any two nodes such that the projection space of one node is equal to the that of the other node with one additional dimension. That is, two nodes $c_{i(1),\ldots,i(r),\mathbf{y}}$ and $c_{j(1),\ldots,j(r),j(r+1),\mathbf{y}'}$ are joined by an edge if

$$\{i(1), \ldots, i(r)\} \subset \{j(1), \ldots, j(r+1)\}.$$

In addition, let c_0 be joined with all the classifiers with a one dimensional projection space. Thus, a node of level r has $(d-r)2^{2^{r+1}}$ offspring in the full tree.

A partial search tree which contains only those offspring that are likely to produce minimal empirical error rate will usually have fewer offspring per node.

Following the partial tree definition template, a TBEF $\mathcal{A}$ is defined:

$$A(c,n) = \left\{ \omega : \sup_{c' \in \mathrm{offsp}_{\mathrm{full}}(c)} |\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_{c'}| \le \epsilon_p \left(n, |\mathrm{offsp}_{\mathrm{full}}(c)|\right) \right\}.$$

The set of offspring of c in the partial tree, assuming that c is in the partial tree, and that its number on that tree is $n_{\mathrm{part}}(c) = n$, is

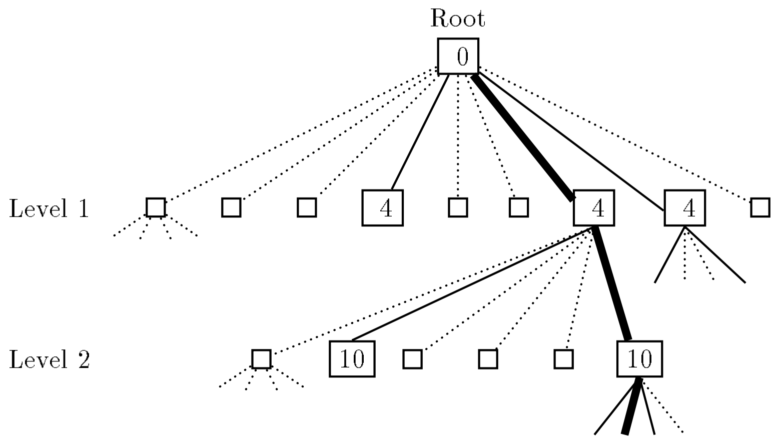

Figure 1: Parts of a full and partial tree for a greedy algorithm

$$\mathrm{offsp}_{\mathrm{part}}(c) = \Bigg\{ c' \in \mathrm{offsp}_{\mathrm{full}}(c) : \\ \mathbf{E}_{\mathbf{P}}(f'_c) \leq \inf_{c'' \in \mathrm{offsp}_{\mathrm{full}}(c)} \mathbf{E}_{\mathbf{P}} f_{c''} + 2\epsilon_p \left(n, \left|\mathrm{offsp}_{\mathrm{full}}(c)\right|\right) \Bigg\}.$$

The family of events just defined, and the corresponding partial tree give a recipe that can be used for producing partial trees for greedy algorithms in a variety of settings. This definition and the following analysis are going to be used twice more in the context of classification trees (see below).

The family of events $\mathcal{A}$ is indeed a TBEF, by the definition of ϵ_p, and using the bound of equation (3). Therefore, the partial tree has coverage probability of no less than $1 - \delta$.

A schematic diagram of the resulting tree appears in Figure 1. A part of a full tree is shown (dotted arcs), with the partial tree overlaid (solid arcs). The path of the greedy algorithm is shown in a bold line. The numbering of the partial tree nodes is shown.

Applying the bound of equation (2), it was thus proved that for the greedy algorithm

$$\mathbf{P}\left(\mathbf{E}_{\mathbf{P}-\mathcal{S}} f^M > \epsilon_p\left(n\left(c^M\right)\right)\right) \leq 2\delta,$$

where the number $n(c^M)$ is calculated for the partial search tree defined above.

Of course, the structure of the partial search tree is unknown, since the expected error rates of the classifiers are unknown. It is, however, possible to build a tree which contains the partial search tree for all sample points outside an event with bounded probability.

In the event

$$A = \bigcap_{c \in V_{\text{part}}} A(c, n_{\text{part}}(c)),$$

the difference $|\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c|$ is bounded by $\epsilon_p \left(n(c), \left|\text{offsp}_{\text{full}}(c)\right|\right)$ for each of the classifiers on the partial search tree and for their offspring. Therefore, in the event A, the set $\text{offsp}_{\text{part}}(c)$ is contained in set

$$\overline{\text{offsp}_{\text{part}}(c)} = \Bigg\{ c' \in \text{offsp}_{\text{full}}(c) :$$

$$\mathbf{E}_{\mathcal{S}}(f_{c'}) < \inf_{c'' \in \text{offsp}_{\text{full}}(c)} \mathbf{E}_{\mathcal{S}} f_{c''} + 4\epsilon_p \left(n(c), \left|\text{offsp}_{\text{full}}(c)\right|\right) \Bigg\}.$$

The considerations above lead to the following procedure for a $1 - 2\delta$ UCB for the expected error rate of the greedy algorithm:

- **Greedy algorithm**: Run the greedy algorithm, producing a path on the tree

  $$\{c_0, c_1\}, \{c_1, c_2\}, \ldots, \{c_{k-1}, c_k\},$$

  where c_0 is the root node and $c_k = c^M$ is the selected classifier.

- **Initialization**: Set $t \leftarrow 1, n \leftarrow 1$ and $i \leftarrow 0$.

- **Stopping condition**: If $i = k$, stop. The UCB is

  $$\mathbf{E}_{\mathcal{S}} f^M + \epsilon_p(n).$$

  Otherwise, continue to the **Offspring** step.

- **Offspring**: Set

  $$K \leftarrow \left\{ c \in \text{offsp}_{\text{full}}(c_i) : \mathbf{E}_{\mathcal{S}} f_c < \mathbf{E}_{\mathcal{S}} f_{c_{i+1}} + 4\epsilon_p \left(n, \left|\text{offsp}_{\text{full}}(c_i)\right|\right) \right\},$$

  where $\left|\text{offsp}_{\text{full}}(c_i)\right| = (d - i)2^{2^{i+1}}$.

- **Loop**: Set $t \leftarrow |K|t, n \leftarrow n + t$, and $i \leftarrow i + 1$. Go back to the **Stopping condition** step.

7 Classification trees

Two of the classification algorithms considered below construct classification trees (CTs) ([6]). This type of classifiers can be naturally described as rooted trees.

The terminal nodes of the CT are elements of $\mathcal{Y}$. Any non-terminal node v, is a simple classifier, with feature space $\mathcal{X}$ and label space $\mathcal{Y}_v$. The edges that connect v to its offspring correspond to the elements of $\mathcal{Y}_v$. The value of a CT, T, for a certain point x in feature space, is calculated as follows:

- **Initialization**: Set $v = v_0$, v_0 being the root node of T.
- **Evaluate**: If v is a terminal node, $T(x) = v$. Otherwise, set $y = v(x)$.
- **Loop**: Find, u, the offspring of v so that the edge $\{v, u\}$ corresponds to the value y. Set $v \leftarrow u$ and go back to step **Evaluate**.

In the cases considered here, $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_d$, where the $\mathcal{X}_i$ are some ordered sets. Furthermore, for all non-terminal v, $\mathcal{Y}_v = \{0, 1\}$, and the non-terminal nodes classifiers are axis orthogonal splits, i.e., they have the form

$$v(x) = \mathbf{1}\left\{x(k_v) \leq a_v\right\},$$

where $x(k_v)$ is the k_v-th coordinate of x, $k_v \in \{1, 2, \ldots, d\}$, and $a_v \in \mathcal{X}_{k_v}$.

Note that rooted trees are used in this paper to describe both CTs and search trees. However, there is no essential connection between these two devices, and, as demonstrated by the previous application and another application below, the search tree construction is relevant for classifiers which are not CTs.

8 Application to a classification tree with binary features

Let $\mathcal{X} = \{0, 1\}^d, \mathcal{Y} = \{0, 1\}$, and let $\mathcal{C}$ be the space of CTs with axis orthogonal splits.

Consider a greedy algorithm, that builds a classification tree by iteratively adding splits which result in the minimal empirical errors achievable by a single split.

Since each split adds one more terminal node to the classification tree, after the n-th split, the algorithm makes a choice among about $(n + 1)d$ possible splits, and for each split it has to decide which side of the split is designated with 0 and which with 1. Thus, in the full search tree each n-th level node has about $2(n + 1)d$ offspring.

Again, the partial search tree may contain substantially fewer offspring per node and will, in such a case, provide much tighter bounds than those implied by the full tree.

The TBEF and the partial tree are defined in exactly the same way as in Section 6. The resulting analysis and UCB for the expected risk are therefore the same, with the sole difference being that the number of offspring in the full search tree (used in the **Offspring** step) is bounded in this case by $2(i + 1)d$.

9 Application to randomized search-correct classification algorithms

This section defines a type of algorithms, which are referred to below as randomized search-correct classification algorithms (RSCCAs), and proves a generalization bound for this class. The proof involves, in addition to the construction of a partial search tree, a conditioning technique which is identical to the one used in [5].

9.1 Randomized search-correct classification algorithms

Definition 2 *A classification algorithm with the following structure is a randomized search correction classification algorithm (RSCCA).*

- **Initialization***: Set $c \leftarrow c_0$.*
- **Stopping condition***: If all the points in the training set are classified correctly by c, or if some other stopping condition is met, output c and stop. Otherwise, continue to the* **Search** *step.*
- **Search***: Select a point, z, which is misclassified by c, randomly among all the misclassified training points.*
- **Correction***: Set $c \leftarrow T(c, z)$.*
- **Loop***: Go back to the* **Stopping condition** *step.*

One example of an RSCCA is the perceptron algorithm ([2]), where $\mathcal{X} = \mathcal{R}^d$, $\mathcal{Y} = \{-1, 1\}$, and $\mathcal{C}$ is the set of linear classifiers in $\mathcal{R}^d$. That is,

$$\left\{c_{a,b} : a \in \mathcal{R}, b \in \mathcal{R}^d\right\},$$

where

$$c_{a,b}(x) = 2\mathbf{1}\left\{x \cdot b > a\right\} - 1.$$

Let $c_0 = c_{0,0}$, and

$$T(c_{a,b}, (x, y)) = c_{a+y, b+yx}.$$

Another example is a variation of the nearest neighbor algorithm: Let $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{R}$ be a distance function defined on the feature space. Define a set of classifiers as follows: $c_\emptyset(0) = y_0$, for some arbitrary $y_0 \in \mathcal{Y}$, and $c_{(x_1,y_1),\ldots,(x_k,y_k)}(x) = y_{j(x)}$, where

$$j(x) = \arg \min_{j \in \{1,2,\ldots,k\}} d(x_j, x).$$

Let $c_0 = c_\emptyset$, and

$$T(c_{z_1,\ldots,z_k}, z) = c_{z_1,\ldots,z_k,z}.$$

9.2 Randomized classification algorithms

Note that the definition of a classification algorithm given above does not allow for the classification associated with a feature point to be random. While the definition could be modified to allow that, it is also possible to consider classification algorithms that involve randomization as a distribution over deterministic classification algorithms. This is the approach used here.

9.3 Conditioning

Since the classifiers considered by the RSCCA are functions of the training points, a conditioning technique has to be employed in order to define a search tree for the algorithm. This sub-section describes the technique, which I call symmetric conditioning.

The following definitions will be used:

Definition 3 *A test set, $\mathcal{S}'$, is a set of iid random variables $Z'_i = (X'_i, Y'_i), i = 1, 2, \dots, l'$ with the same distribution $\mathbf{P}$ as the variables in the training set $\mathcal{S}$, and independent of $\mathcal{S}$.*

The law of large numbers implies that, for large l', the test set error rate

$$\mathbf{E}_{\mathcal{S}'} f^M = \frac{1}{l'} \sum_{i=1}^{l'} \mathbf{1} \left\{ c^M(X'_i) \neq Y'_i \right\}$$

is a good approximation of the expected error rate.

The test set can be identified with the continuation of the sequence Z_i, i.e., $Z'_i = Z_{i+l}$. This notation is useful in defining the symmetric field of events:

Definition 4 *The symmetric field, $\mathcal{F}_\pi$, is the field spanned by events of the form:*

$$\cup_\pi \left\{ Z_{\pi(1)} \in B_1, \dots, Z_{\pi(r+l)} \in B_{l+l'} \right\},$$

where the union is over all permutations π of the numbers $1, \dots, l+l'$, and the B_i's are measurable subsets of $\mathcal{Z}$.

The symmetric field has the property that, conditioned on the symmetric field, the distribution of $Z_1, \dots, Z_{l+l'}$ is uniform over the $(l+l')!$ permutations of some $l+l'$ (not necessarily distinct) values $z_1, \dots, z_{l+l'}$. This is true, no matter what is the underlying probability measure $\mathbf{P}$.

Symmetric conditioning relies on the fact that

$$\mathbf{P}\left(\mathbf{E}_{\mathcal{S}'} f^M - \mathbf{E}_{\mathcal{S}} f^M > \epsilon - \frac{1}{l'} \,\middle|\, \mathbf{E}_{\mathbf{P}-\mathcal{S}} f^M > \epsilon \right) \geq \frac{1}{2}.$$

It is therefore enough to bound $\mathbf{P}(\mathbf{E}_{\mathcal{S}'} f^M - \mathbf{E}_{\mathcal{S}} f^M > \epsilon)$. Since

$$\mathbf{P}(\mathbf{E}_{\mathcal{S}'} f^M - \mathbf{E}_{\mathcal{S}} f^M > \epsilon) = \mathbf{E}\mathbf{P}\left(\mathbf{E}_{\mathcal{S}'} f^M - \mathbf{E}_{\mathcal{S}} f^M > \epsilon \,\middle|\, \mathcal{F}_\pi \right),$$

it is sufficient to show that for all $z_1, \dots, z_{2l}$,

$$\frac{1}{(2l)!} \sum_\pi \mathbf{1} \left\{ \left| \frac{1}{l} \sum_{i=1}^{l} f^M(z_{\pi(i)}) - \frac{1}{l'} \sum_{i=l+1}^{l+l'} f^M(z_{\pi(i)}) \right| > \epsilon \right\}$$

is small, where

$$f^M = M\left(z_{\pi(1)}, \dots, z_{\pi(l)} \right).$$

That is, it is sufficient to show that the conditional probability of a large difference given the symmetric field is small.

Given the symmetric field, the full search tree can be constructed. The root of the tree is c_0, as denoted in the definition of an RSCCA. The offspring of each classifier, c, in the tree, are

$$\{T(c, z_i) : z_i = (x_i, y_i) \in \mathcal{S} \cup \mathcal{S}',\, c(x_i) \neq y_i\}.$$

Note that the tree contains branches that correspond to the elements of both the training and test sets.

9.4 The partial search tree

Throughout the rest of this sub-section, the conditioning on the symmetric field is implicitly assumed. The randomization of the data is thus solely through the partition of a set of points $\{z_1, \ldots, z_{l+l'}\}$ into the training and the test sets. The probability space $(\omega, \mathcal{F}_\omega, \mathbf{P})$ can therefore be treated as the space of uniform probability over such partitions. To further simplify the expressions, I set $l' = l$.

The randomized search order can be implemented by determining an order of priority among the offspring of each classifier. This order will be expressed as a set of functions, one for each classifier:

$$\mathrm{o}_c : \mathrm{offsp}_{\mathrm{full}}(c) \to \{1, \ldots, |\mathrm{offsp}_{\mathrm{full}}(c)|\}.$$

For each classifier c, the priorities, $\mathrm{o}_c(c')$, are distinct integers in the range $1, \ldots, |\mathrm{offsp}_{\mathrm{full}}(c)|$, with lower values corresponding to higher priority. The order of priorities is chosen uniformly over all possible orderings of the offspring of each classifier, and independently for each classifier.

The algorithm then proceeds from a classifier to the offspring in the training set that has the highest priority. Given the priorities of the offspring, the chance that the algorithm will proceed to a particular offspring, given that the parent is in the path, is the chance that all the offspring with higher priorities are not in the training set, but rather in the test set. Therefore, the chance that a low priority offspring is in the path is small.

Specifically, since

- $l\mathbf{E}_{\mathcal{S}} f_c$ is the number of offspring of c in the training set,
- $l\mathbf{E}_{\mathcal{S}'} f_c$ is the number of offspring of c in the test set,
- the order of priorities among the offspring of c is uniform over all orders and independent of the training set and the priorities of offspring of the other classifiers,

then the the probability of a classifier on the path of M, of having no offspring in the training set among the m highest priority offspring is bounded by $\mathbf{E}(R_c)^m$, where

$$R_c = \frac{\mathbf{E}_{\mathcal{S}'} f_c}{\mathbf{E}_{\mathcal{S}} f_c + \mathbf{E}_{\mathcal{S}'} f_c} = \frac{\mathbf{E}_{\mathcal{S}'} f_c}{2\mathbf{E}_{\mathbf{P}} f_c}.$$

In the event $\{\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c \leq \epsilon\}$,

$$R_c \leq r_c(\epsilon) = \frac{\mathbf{E}_{\mathbf{P}} f_c + \epsilon}{2\mathbf{E}_{\mathbf{P}} f_c}.$$

Therefore, given $\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c \leq \epsilon$, in order to make sure that the conditional probability that the algorithm chooses an offspring of c that is not on the partial search tree is less than η, it is enough to retain in the tree only the $\log \eta / \log r_c(\epsilon)$ highest priority offspring of c.

In light of the above, the partial search tree can be constructed using the partial tree definition template. Fix a sequence of constants η_i, $i = 0, 1, \ldots$, and denote $\eta = \sum_{i=0}^{\infty} \eta_i$. Set

$$\begin{aligned} A(c,n) \quad = \quad & \Bigg\{ (\omega, \omega') : \mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c \leq \epsilon_p(n) \text{ and} \\ & \min_{c' \in \mathcal{S} \cap \text{offsp}_{\text{full}}(c)} \text{o}_c(c') \leq \frac{\log \eta_{\text{lev}(c)}}{\log R_c} \text{ for all } c' \text{ in } M\text{'s path} \Bigg\}. \end{aligned}$$

The set of offspring of c in the partial tree, assuming that c is in the partial tree, and that its number on that tree is $n_{\text{part}}(c) = n$, is

$$\text{offsp}_{\text{part}}(c) \quad = \quad \left\{ c' \in \text{offsp}_{\text{full}}(c) : \text{o}_c(c') \leq \frac{\log \eta_{\text{lev}(f)}}{\log r_c(\epsilon(n))} \right\}.$$

The probability of the event

$$A = \bigcap_{c \in V_{\text{part}}} A(c, n_{\text{part}}(c)),$$

in which case $c^M \in V_{\text{part}}$, is at least $1 - \delta - \eta$.

As in the case of the CT, the construction of the partial search tree requires the values of the expected error rates of the classifiers in the full search tree, which are not available. Again, as in the case of the CT, a tree which contains the partial search tree can be constructed.

In the event $\{\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_c \leq \epsilon\}$,

$$r_c(\epsilon) \leq \overline{r_c}(\epsilon) = \frac{\mathbf{E}_{\mathcal{S}} f_c}{2\mathbf{E}_{\mathcal{S}} f_c - 2\epsilon}.$$

Therefore, in the event A, $\text{offsp}_{\text{part}}(c)$ is contained in

$$\overline{\text{offsp}_{\text{part}}(c)} = \left\{ c' \in \text{offsp}_{\text{full}}(c) : \text{o}_c(c') \leq \frac{\log \eta_{\text{lev}(c)}}{\log \overline{r_c}(\epsilon_p(n))} \right\}.$$

The following procedure calculates the resulting level $1 - \delta - \eta$ UCB for the expected risk of the classifier generated by the RSCCA.

- **RSCCA algorithm**: Run the RSCCA algorithm, producing a path on the tree
$$\{c_0, c_1\}, \{c_1, c_2\}, \ldots, \{c_{k-1}, c_k\},$$
where c_0 is the root node and $c_k = c^M$ is the selected classifier.

- **Initialization**: Set $t \leftarrow 1, n \leftarrow 1$ and $i \leftarrow 0$.

- **Stopping condition**: If $i = k$, stop. The UCB is
$$\mathbf{E}_{\mathcal{S}} f^M + \epsilon_p(n).$$
Otherwise, continue to the **Count offspring** step.

- **Count offspring**: Set
$$\overline{r} \leftarrow \frac{\mathbf{E}_{\mathcal{S}} f_{c_i}}{2\mathbf{E}_{\mathcal{S}} f_{c_i} - 2\epsilon_p(n)},$$
and
$$m \leftarrow \frac{\log \eta_i}{\log \overline{r}}.$$

- **Loop**: Set $t \leftarrow mt, n \leftarrow n + t$, and $i \leftarrow i + 1$. Go back to the **Stopping condition** step.

Note that the number of offspring of any classifier in the partial tree, as well as in the random tree that covers it, does not grow as the number of training points, l, increases. The number of offspring of a classifier is an increasing function of bounds of the overfit of its ancestors. These bounds actually decrease as l increases, and so the number of offspring drops (but not to 1) as the training set grows large. The resulting simultaneous confidence bound therefore has the form $\mathbf{E}_{\mathcal{S}} f_c + \frac{\alpha_c}{l}$, rather than the form $\mathbf{E}_{\mathcal{S}} f_c + \frac{\alpha_c \log l}{l}$, as results from the bound of [5].

10 Application to a classification tree with continuous features

The application of the search tree bound to CTs with continuous features uses the symmetric conditioning as well.

Let $\mathcal{X} = \mathcal{R}^d, \mathcal{Y} = \{0, 1\}$, and let $\mathcal{C}$ be the space of CTs with axis orthogonal splits.

Again, consider a greedy algorithm, that builds a classification tree by iteratively adding splits which result in the minimal empirical errors achievable by a single split.

As in sub-section 9.4, this section uses symmetric conditioning implicitly, thus reducing the random choice of the training set to the random partitioning of a set of $2l$ points, $z_1, \ldots, z_{2l}$, into the training set $\mathcal{S}$ and test set $\mathcal{S}'$.

In the full search tree, the offspring of a particular CT, c, are all the CTs which result by substituting one of the leaves of c with a non-leaf node, which corresponds to a classifier, v, of the form

$$v(x) = \mathbf{1}\left\{x(k_v) \geq x_v(k_v)\right\},$$

where x_v is the feature vector of one of the points $z_1, \ldots, z_{2l}$. This new non-leaf node, v, has two offspring which are leaves.

As in the case of the projection histogram CA and of the binary tree, the TBEF $\mathcal{A}$ is defined as:

$$A(c,n) \quad = \quad \left\{\omega : \sup_{c' \in \mathrm{offsp}_{\mathrm{full}}(c)} \left|\mathbf{E}_{\mathbf{P}-\mathcal{S}} f_{c'}\right| \leq \epsilon_p\left(n, \left|\mathrm{offsp}_{\mathrm{full}}(c)\right|\right)\right\},$$

and again, the set of offspring of c in the partial tree, assuming that c is in the partial tree, and that its number on that tree is $n_{\mathrm{part}}(c) = n$, is

$$\mathrm{offsp}_{\mathrm{part}}(c) = \Bigg\{c' \in \mathrm{offsp}_{\mathrm{full}}(c) :$$
$$\mathbf{E}_{\mathbf{P}}(f'_c) \leq \inf_{c'' \in \mathrm{offsp}_{\mathrm{full}}(c)} \mathbf{E}_{\mathbf{P}} f_{c''} + 2\epsilon_p\left(n, \left|\mathrm{offsp}_{\mathrm{full}}(f')\right|\right)\Bigg\}.$$

This gives rise to a partial tree that has coverage probability of no less than $1-\delta$, and thus,

$$\mathbf{P}\left(\mathbf{E}_{\mathbf{P}-\mathcal{S}} f^M > \epsilon_p\left(n_{\mathrm{part}}(c^M)\right)\right) \leq 2\delta,$$

where the number $n(c^M)$ is calculated for the partial search tree defined above.

Calculating the $n(c)$'s, however, involves, in the case of continuous feature space, not only the unknown values, $\mathbf{E}_{\mathbf{P}} f_c$, but also the unknown points in the test set $\mathcal{S}'$.

As in the previous applications, $\left|\mathrm{offsp}_{\mathrm{part}}(c)\right|$ will be bounded from above, giving bounds from above for $n(c)$ for all the classifiers on the partial search tree.

Bounding $\left|\mathrm{offsp}_{\mathrm{part}}(c) \cap \mathcal{S}\right|$ can be done in the same way that it was done for the binary tree. That is, in the event $A = \bigcap_{c \in V_{\mathrm{part}}} A(c, n_{\mathrm{part}}(c))$,

$$\mathrm{offsp}_{\mathrm{part}}(c) \cap \mathcal{S} \subseteq \overline{\mathrm{offsp}^{\mathcal{S}}_{\mathrm{part}}}(c), \tag{7}$$

where

$$\overline{\mathrm{offsp}^{\mathcal{S}}_{\mathrm{part}}}(c) = \Bigg\{c' \in \mathrm{offsp}_{\mathrm{full}}(c) :$$
$$\mathbf{E}_{\mathcal{S}}(f_{c'}) \leq \inf_{c'' \in \mathrm{offsp}_{\mathrm{full}}(c)} \mathbf{E}_{\mathcal{S}} f_{c''} + 4\epsilon_p\left(n(c), \left|\mathrm{offsp}_{\mathrm{full}}(c)\right|\right)\Bigg\}.$$

To bound $\left|\text{offsp}_{\text{part}}(c) \cap \mathcal{S}'\right|$ define the event $A' = \bigcap_{c \in V_{\text{part}}} A'(c, n_{\text{part}}(c))$, where

$$A'(c,n) \quad = \quad \left\{\omega : \left|\text{offsp}_{\text{part}}(c) \cap \mathcal{S}\right| \leq \left|\text{offsp}_{\text{part}}(c) \cap \mathcal{S}'\right| + l\epsilon_p(n)\right\}.$$

By the definition of ϵ_p, $\mathbf{P}(A') \geq 1 - \delta$. Therefore $\mathbf{P}(A \cap A') \geq 1 - 2\delta$.

In the event $A \cap A'$,

$$\left|\text{offsp}_{\text{part}}(c)\right| \leq 2\left|\text{offsp}_{\text{part}}(c) \cap \mathcal{S}\right| + l\epsilon_p(n_{\text{part}}(c)).$$

This, together with the inclusion in (7), implies that

$$2\left|\overline{\text{offsp}^{\mathcal{S}}_{\text{part}}}(c)\right| + l\epsilon_p(n_{\text{part}}(c))$$

is a level $1 - 2\delta$ UCB for $\left|\text{offsp}_{\text{part}}(c)\right|$.

The resulting UCB for the risk of c^M is implemented by the following algorithm. Note that in the full search tree the number of offspring of any classifier is bounded by $2d(l+1)$.

- **Greedy algorithm**: Run the greedy algorithm, producing a sequence of classifiers $c_0, c_1, \ldots, c_{k-1}, c_k = c^M$.
- **Initialization**: Set $t \leftarrow 1, n \leftarrow 1$ and $i \leftarrow 0$.
- **Stopping condition**: If $i = k$, stop. The UCB is

  $$\mathbf{E}_{\mathcal{S}} f^M + \epsilon_p(n).$$

  Otherwise, continue to the **Offspring** step.
- **Offspring**: Set

  $$K \leftarrow \left\{c \in \text{offsp}_{\text{full}}(c_i) : \mathbf{E}_{\mathcal{S}} f_c \leq \mathbf{E}_{\mathcal{S}} f_{c_{i+1}} + 4\epsilon_p\left(n, 2d(l+1)\right)\right\}.$$
- **Update**: Set
  - $m \leftarrow 2|K| + l\epsilon_p(n)$,
  - $t \leftarrow mt$,
  - $n \leftarrow n + t$,
  - $i \leftarrow i + 1$.
- **Loop**: Go back to the **Stopping condition** step.

11 Conclusions

This paper presented a way to describe some classification algorithms as tree search algorithms. It was shown that the more narrow the search is, the smaller is the overfit that the algorithm incurs. In addition to providing insight into the behavior of classification algorithms, the search tree description provides overfit bounds for several classification algorithms, e.g., classification trees, which improve upon other, more general, bounds, such as VC dimension bounds and the bounds of [5].

The question of how sharp the bounds obtained are was not addressed and remains a topic for further investigation.

Acknowledgments I thank Prof. Peter Bickel for reading the drafts of this paper and for his insightful comments that have led to significant improvements in the presentation.

References

[1] Vapnik, V. N. (1998). *Statistical Learning Theory.* John Wiley & Sons, Inc., New York.

[2] Minsky, M. L. and Papert S. A. (1988). *Perceptrons.* The MIT Press, Cambridge.

[3] Freund, Y. (1998). Self bounding classification algorithms. *COLT '98: Proceedings of the eleventh annual conference on computational learning theory*, 247-258. ACM Press, New York, NY, 1998.

[4] Langford J. and Blum A. (1999). Microchoice bounds and self bounding learning algorithms. *COLT '99: Proceedings of the twelfth annual conference on computational learning theory.*

[5] Gat, Y. (1999). A bound concerning the generalization ability of a certain class of learning algorithms. *Technical report No. 548.* Department of Statistics, UC Berkeley.

[6] Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984). *Classification and regression trees.* Wadsworth, Belmont, CA.

Yoram Gat
University of California, Berkeley
367 Evans Hall
Berkeley, CA, 94720
E-mail: yoram@stat.berkeley.edu