Computational Learning Theory

Slides by Carla P. Gomes and Nathalie Japkowicz

(Reading: R&N AIMA 3rd ed., Chapter 18.5)

Computational Learning Theory

Inductive learning:

given the training set, a **learning algorithm** generates a hypothesis.

Run hypothesis on the test set. The results say *something* about how good our hypothesis is.

But how much do the results really tell you? Can we be certain about how the learning algorithm generalizes?

We would have to see all the examples.

Insight: introduce probabilities to measure degree of certainty and correctness (Valiant 1984).

Computational Learning Theory

Example:

We want to use height to distinguish men and women drawing people from the same distribution for training and testing.

We can never be absolutely certain that we have learned correctly our target (hidden) concept function. (E.g., there is a non-zero chance that, **so far**, we have only seen a sequence of bad examples)

E.g., relatively tall women and relatively short men…

We'll see that it's generally highly unlikely to see a long series of bad examples!

Aside: flipping a coin

Experimental data

C program – simulation of flips of a fair coin:

Runs of 100 flips (expect 50 "tails"):
On 1,000 tries reached 66
On 10,000 tries reached 69
On 100,000 tries reached 70
On 1,000,000 tries reached 74 (48% over 50)

Runs of 1000 flips (expect 500 "tails"):
On 1,000 tries reached 564
On 10,000 tries reached 564
On 100,000 tries reached 569
On 1,000,000 tries reached 579 (16% over 500)

Experimental Data Contd.

Runs of 10,000 flips (expect 5000 "tails"):

On 1,000 tries reached 5150

On 10,000 tries reached 5183

On 100,000 tries reached 5231

On 1,000,000 tries reached 5239 (5% over 5000)

With a sufficient number of flips
(set of flips=example of coin bias),
large outliers become quite rare.

Coin example is the key to computational learning theory!

Computational Learning Theory

Intersection of AI, statistics, and theory of computation.

Introduce Probably Approximately Correct Learning concerning
efficient learning

For our learning procedures we would like to prove that:

With high probability an (efficient) learning algorithm will find a hypothesis that is approximately identical to the hidden target concept.

Note the double "hedging" – probably and approximately.

Why do we need both levels of uncertainty (in general)?

Probably Approximately Correct Learning

Underlying principle:

Seriously wrong hypotheses can be found out almost certainly (with high probability) using a "small" number of examples

- Any hypothesis that is consistent with a significantly large set of training examples is unlikely to be seriously wrong: it must be probably approximately correct.

- Any (efficient) algorithm that returns hypotheses that are PAC is called a PAC-learning algorithm

Probably Approximately Correct Learning

How many examples are needed to guarantee correctness?

- Sample complexity (# of examples to "guarantee" correctness) grows with the size of the Hypothesis space

- Stationarity assumption: Training set and test sets are drawn from the same distribution

Notations

Notations:

- X: set of all possible examples
- D: distribution from which examples are drawn
- H: set of all possible hypotheses
- N: the number of examples in the training set
- *f:* the true function to be learned

Assume: the true function f is in H.

Error of a hypothesis *h* wrt *f* :

Probability that *h* differs from *f* on a randomly picked example:

$$error(h) = P(h(x) \neq f(x) | \ x \text{ drawn from } D)$$

Exactly what we are trying to measure with our test set.

Approximately Correct

A hypothesis h is approximately correct if:

$$error(h) \leq \varepsilon,$$

where ε is a given threshold, a small constant

Goal:

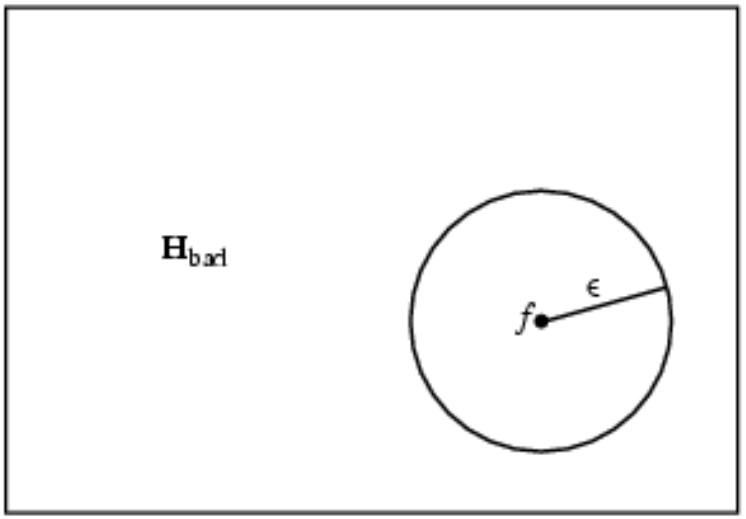
Show that after seeing a small (poly) number of examples N, with high probability, all consistent hypotheses will be approximately correct.

I.e, chance of "bad" hypothesis, (high error but consistent with examples) is small (i.e, less than δ)

Approximately Correct

Approximately correct hypotheses lie inside the ε -ball around f;

Those hypotheses that are seriously wrong ($h_b \in$ HBad) are outside the ε -ball,



Error(h_{bad})= P($h_b(x) \neq f(x)$| x drawn from D) > ε,

Thus the probability that the h_{bad} (a seriously wrong hypothesis) *disagrees* with one example is at least ε (definition of error).

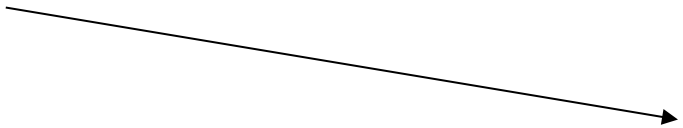
Thus the probability that the h_{bad} (a seriously wrong hypothesis) agrees with one example is no more than (1- ε).

So for N examples, P(h_b agrees with N examples) ≤ $(1- \varepsilon)^N$.

Approximately Correct Hypothesis

The probability that H_{Bad} contains at least one consistent hypothesis is bounded by the sum of the individual probabilities.

$P(H_{bad}$ contains a consistent hypothesis, agreeing with all the examples) $\leq |H_{bad}|(1-\varepsilon)^N \leq |H|(1-\varepsilon)^N$



h_{bad} agrees with one example is no more than $(1-\varepsilon)$.

$P(H_{bad}$ contains a consistent hypothesis) $\leq |H_{bad}|(1-\varepsilon)^N \leq |H|(1-\varepsilon)^N$

Goal –

Bound the probability of learning a bad hypothesis below some small number δ.

Note: $(1-\varepsilon) \leq e^{-\varepsilon}$

The **more accuracy** (smaller ε), and
the **more certainty** (with smaller δ)
one wants, the **more examples** one needs.

$P(H_{bad}$ contains a consistent hypothesis) $\leq |H|(e^{-\varepsilon})^N \leq \delta$

What is the probability $P(H_{good})$ of learning a good hypothesis?

How large should N be?

Derivation: see blackboard

Sample Complexity: Number of examples to guarantee a PAC learnable function class

If the learning algorithm returns a hypothesis that is consistent with this many examples, then with probability at least $(1-\delta)$ the learning algorithm has an error of at most ε.

and the hypothesis is Probably Approximately Correct.

Probably Approximately correct hypothesis h:

- If the probability of a small error (error(h) $\leq \varepsilon$) is greater than or equal to a given threshold 1 - δ
- A bound on the number of examples (sample complexity) needed to guarantee PAC, that is polynomial

$$N \geq \frac{1}{\varepsilon}(\ln \frac{1}{\delta} + \ln |H|)$$

(The more accuracy (with smaller ε), and the more certainty desired (with smaller δ), the more examples one needs.)

- An efficient learning algorithm

Theoretical results apply to fairly simple learning models (e.g., decision list learning)

PAC Learning

Two steps:

Sample complexity – a polynomial number of examples suffices to specify a good consistent hypothesis (error(h) ≤ ε) with high probability (≥ 1 – δ).

$$N \geq \frac{1}{\varepsilon}(\ln \frac{1}{\delta} + \ln |H|)$$

Computational complexity – there is an efficient algorithm for learning a consistent hypothesis from the small sample.

Let's be more specific with examples.

Example: Boolean Functions

Consider H the set of all Boolean function on n attributes → $| H | = 2^{2^n}$

$$N \geq \frac{1}{\varepsilon}(\ln \frac{1}{\delta} + \ln | \mathrm{H} |) = O(2^n)$$

So the sample complexity grows as 2^n ☹!
(same as the number of all possible examples)
Not PAC-Learnable!

So, any learning algorithm will do not better than a lookup table if it merely returns a hypothesis that is consistent with all known examples!

Intuitively what does it say about H?
Finite H required!

Coping With Learning Complexity

1. Force learning algorithm to look for smallest consistent hypothesis.

   We considered that for Decision Tree Learning, often worst case intractable though.

   .

2. Restrict size of hypothesis space.

   e.g., Decision Lists → restricted form of Boolean Functions:

   Hypotheses correspond to a series of tests, each of which a conjunction of literals

   Good news: only a poly size number of examples

   is required for guaranteeing PAC learning K-DL functions

   and there are efficient algorithms for learning K-DL

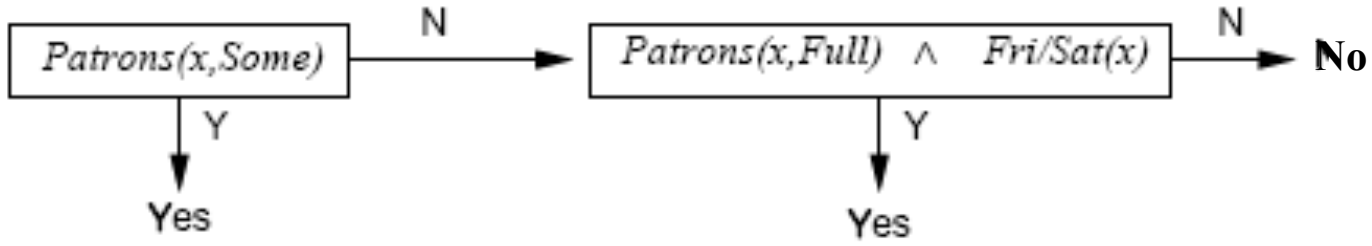
Decision Lists

Resemble Decision Trees, but with simpler structure:

Series of tests, each test a conjunction of literals;

If a test succeeds, decision list specifies value to return;

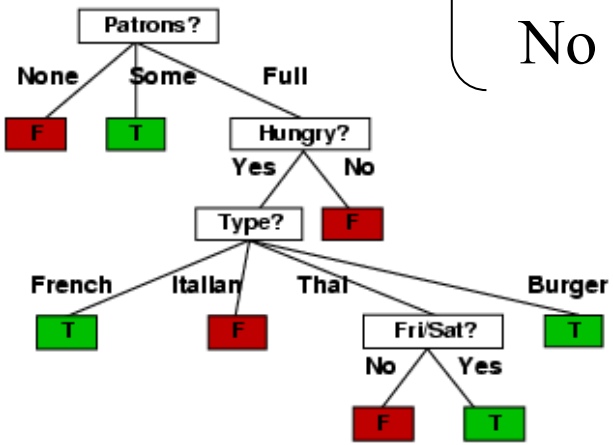
If test fails, processing continues with the next test in the list.



(a)	(b∧c)	
Y	Y	N

a=Patrons(x,Some) b=patrons(x,Full) c=Fri/Sat(x)

Note: if we allow arbitrarily many literals per test , decision list can express all Boolean functions.



(a)	(b)	(¬d)	(e)	(f)	(h)	(i)	
No	Yes	No	Yes	No	Yes	Yes	No

a=Patrons(x,None) b=Patrons(x,Some)

d=Hungry(x)

e=Type(x,French) f=Type(x,Italian) g=Type(x,Thai) h=Type(x,Burger)

i=Fri/Sat(x)

K Decision Lists

Decision Lists with limited expressiveness (K-DL) – at most k literals per test

2-DL

(a)	(b∧c)	
Y	Y	N

K-DL is PAC learnable!!!

For fixed k literals, the number of examples needed for PAC learning a K-DL function is polynomial in the number of attributes n.

There are efficient algorithms for learning K-DL functions.

So how do we show K-DL is PAC-learnable?

2-DL

(x)	(y)	(w∧¬v)	(u∧¬b)	
No	Yes	No	Yes	No

K Decision Lists: Sample Complexity

$$N \geq \frac{1}{\varepsilon}(\ln\frac{1}{\delta} + \ln|H|)$$

What's the size of the hypothesis space H, i.e, |K-DL(n)|?

K-Decision Lists → set of tests: each test is a conjunct of at most k literals

How many possible tests (conjuncts) of length at most k, given n literals, *conj(n,k)?*

$$|Conj(n,k)| \leq 2n + \binom{2n}{2} + \binom{2n}{3} \cdots + \binom{2n}{k} = O(n^k)$$

A conjunct (or test) can appear in the list as: Yes, No, absent from list

So we have at most $3^{|Conj(n,k)|}$ different K-DL lists (ignoring order)

But the order of the tests (or conjuncts) in a list matters.

$$|\text{k-DL(n)}| \leq 3^{|Conj(n,k)|}\,|Conj(n,k)|!$$

After some work, we get (useful exercise!; try mathematica or maple)

$$| K - DL(n) | = 2^{O(n^k \log_2 (n^k))}$$

1 - Sample Complexity of K-DL is:

$$N \geq \frac{1}{\varepsilon}(\ln \frac{1}{\delta} + O(n^k \log_2 (n^k)))$$

Recall sample complexity formula

$$N \geq \frac{1}{\varepsilon}(\ln \frac{1}{\delta} + \ln | H |)$$

For fixed k literals, the number of examples needed for PAC learning a K-DL function is polynomial in the number of attributes n, ☺!

:

2 – Efficient learning algorithm – a decision list of length k can be learned in polynomial time.

So K-DL is PAC learnable!!!

Decision-List-Learning Algorithm

Greedy algorithm for learning decisions lists:

→repeatedly finds a test that agrees with some subset of the training set;

→adds test to the decision list under construction and removes the corresponding examples.

→uses the remaining examples, until there are no examples left, for constructing the rest of the decision list.

(see R&N, page 672. for details on algorithm).

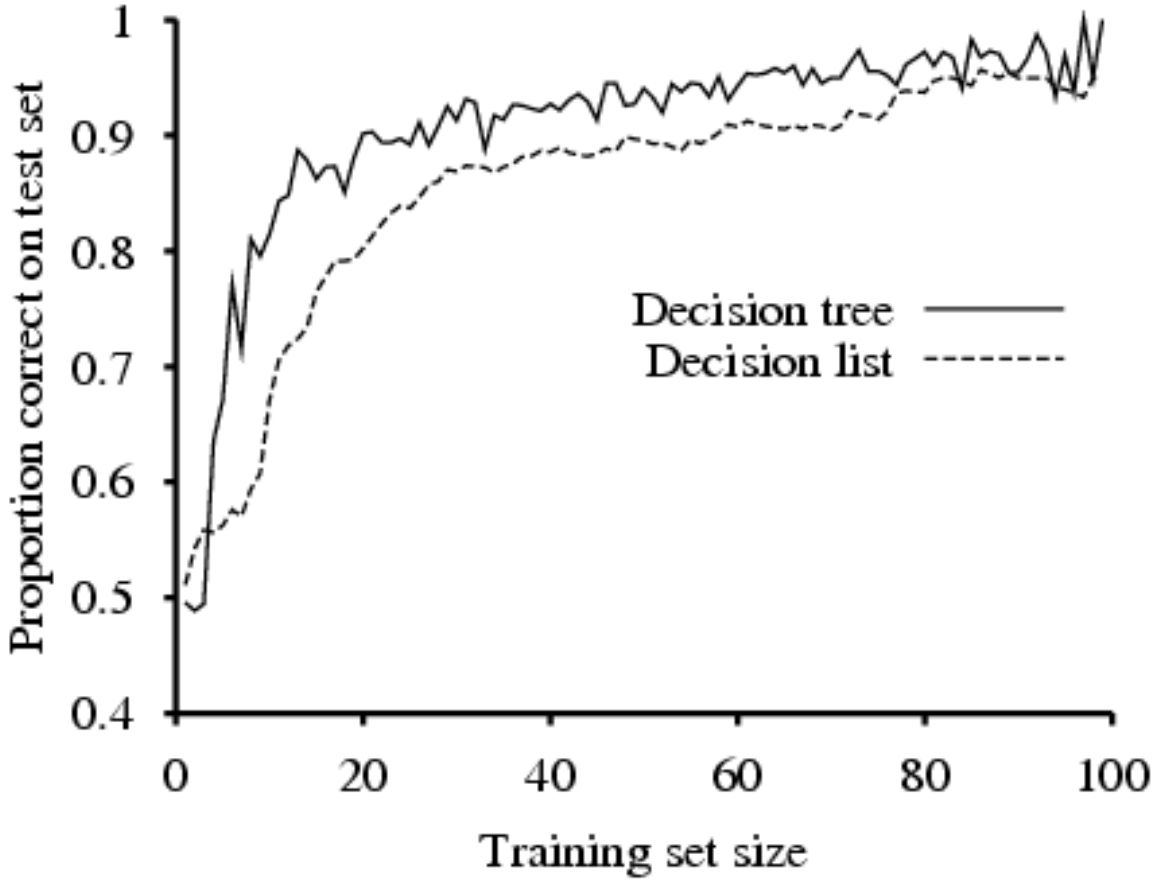
Decision-List-Learning Algorithm

Greedy algorithm for learning decisions lists:

function DECISION-LIST-LEARNING(*examples*) **returns** a decision list, *No* or failure

if *examples* is empty **then return** the value *No*
t ← a test that matches a nonempty subset $examples_t$ of *examples*
such that the members of $examples_t$ are all positive or all negative
if there is no such *t* **then return** failure
if the examples in $examples_t$ are positive **then** *o* ← *Yes*
else *o* ← *No*
return a decision list with initial test *t* and outcome *o*
and remaining elements given by DECISION-LIST-LEARNING(*examples* − $examples_t$)

Decision-List-Learning Algorithm



Restaurant data.

Examples

1. H space of Boolean functions

Not PAC Learnable, hypothesis space too big: need too many examples (sample complexity not polynomial)!

2. K-DL

   PAC learnable

3. Conjunction of literals

   PAC learnable

Probably Approximately Correct Learning (PAC)Learning (summary)

A class of functions is said to be PAC-learnable if there exists an efficient learning algorithm such that for all functions in the class, and for all probability distributions on the function's domain, and for any values of epsilon and delta ($0 < \text{epsilon, delta} < 1$), using a polynomial number of examples, the algorithm will produce a hypothesis whose error is smaller than ε with probability at least δ.

The error of a hypothesis is the probability that it will differ from the target function on a random element from its domain, drawn according to the given probability distribution.

Basically, this means that:

- there is some way to learn efficiently a "pretty good" approximation of the target function.
- the probability is as big as you like that the error is as small as you like.

(Of course, the tighter you make the bounds, the harder the learning algorithm is likely to have to work).

Discussion

Computational Learning Theory studies the tradeoffs between the expressiveness of the hypothesis language and the complexity of learning

Probably Approximately Correct learning concerns efficient learning

Sample complexity --- polynomial number of examples

Efficient Learning Algorithm

Word of caution:

PAC learning results → worst case complexity results.

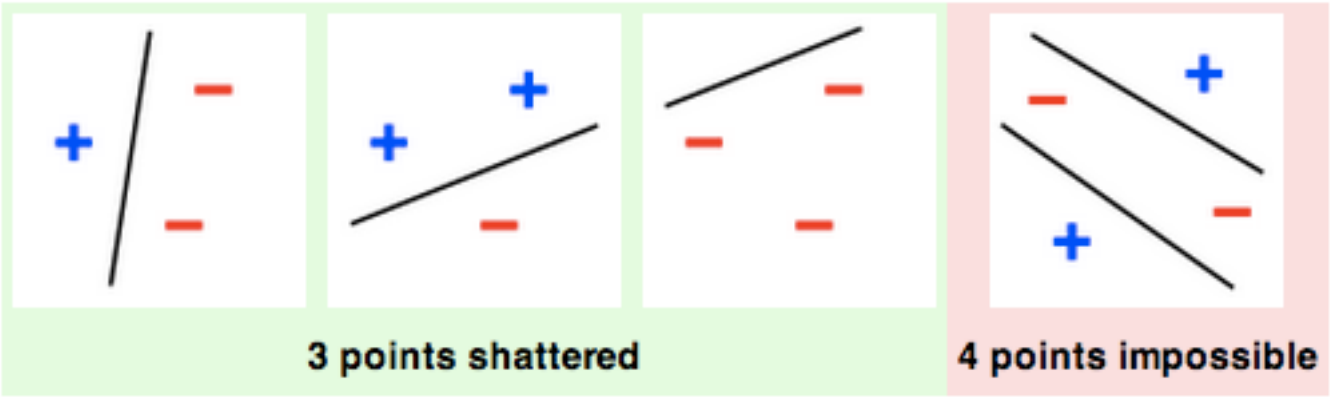
Sample Complexity for Infinite Hypothesis Spaces I: VC-Dimension

- The PAC Learning framework has 2 disadvantages:
  - It can lead to weak bounds
  - Sample Complexity bound cannot be established for infinite hypothesis spaces
- We introduce new ideas for dealing with these problems:
  - A set of instances S is **shattered** by hypothesis space H iff for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.

VC Dimension: Example

The VC dimension of a model f is the maximum number of points that can be arranged so that f shatter them. More formally, it is h' where h' is the maximum h such that some data point set of cardinality h can be shattered by f.

For example, consider a straight line as the classification model: the model used by a perceptron. The line should separate positive data points from negative data points. There exist sets of 3 points that can indeed be shattered using this model (any 3 points that are not collinear can be shattered). However, no set of 4 points can be shattered:



Sample Complexity for Infinite Hypothesis Spaces I: VC-Dimension

The Vapnik-Chervonenkis dimension, **VC(H)**,
of hypothesis space H defined over instance space X
is the **size of the largest finite subset of X** shattered by H.

If arbitrarily large finite sets of X can
be shattered by H, then VC(H)=∞

VC Dimension: Example 2

- H = Axis parallel rectangles in R^2
- What is the VC dimension of H
- Can we PAC learn?

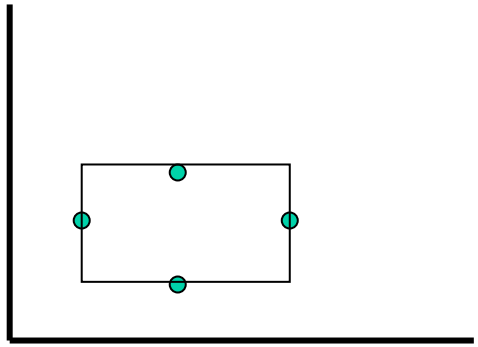
whesse@clarkson.edu

Learning Rectangles

- Consider axis parallel rectangles in the real plane
- Can we PAC learn it ?

  (1) What is the VC dimension ?

- Some four instances (points on the rectangle) can be shattered



Shows that VC(H)>=4

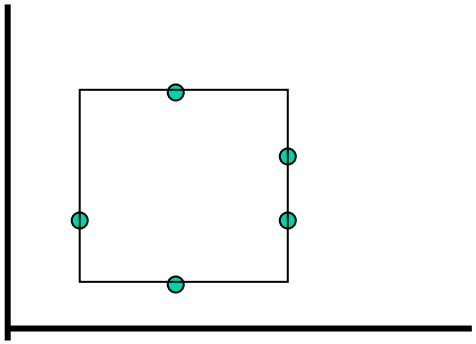
whesse@clarkson.edu

Learning Rectangles

- Consider axis parallel rectangles in the real plane
- Can we PAC learn it ?

  (1) What is the VC dimension ?

- But, no five instances can be shattered



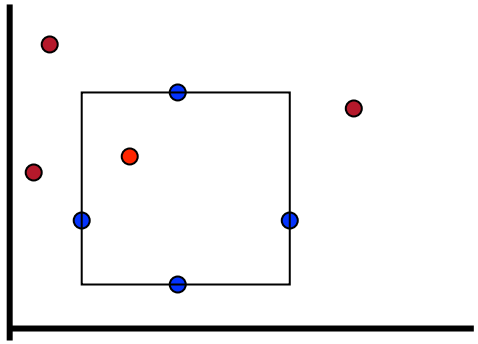
- Two points must share a line, and if we take 4 points from different lines, there is no rectangle that separates the 4 points from the remaining one.

Therefore VC(H) = 4

whesse@clarkson.edu

Learning Rectangles

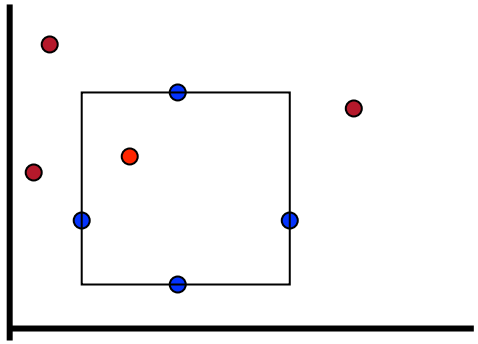
- Consider axis parallel rectangles in the real plane
- Can we PAC learn it ?
  (1) What is the VC dimension ?
  (2) Can we give an efficient algorithm ?



whesse@clarkson.edu

Learning Rectangles

- Consider axis parallel rectangles in the real plane
- Can we PAC learn it ?
  (1) What is the VC dimension ?
  (2) Can we give an efficient algorithm ?



Find the smallest rectangle that contains the positive examples (necessarily, it will not contain any negative example, and the hypothesis is consistent.

Axis parallel rectangles are efficiently PAC learnable.

whesse@clarkson.edu

The *Mistake Bound* Model of Learning

- The ***Mistake Bound*** framework is different from the PAC framework as it considers learners that receive a sequence of training examples and that predict, upon receiving each example, what its target value is.
- The question asked in this setting is: ***"How many mistakes will the learner make in its predictions before it learns the target concept?"***
- This question is significant in practical settings where learning must be done while the system is in actual use.

Optimal Mistake Bounds

- **Definition:** Let C be an arbitrary nonempty concept class. The optimal mistake bound for C, denoted $Opt(C)$, is the minimum over all possible learning algorithms A of $M_A(C)$. $Opt(C)=min_{A \in Learning\_Algorithms} M_A(C)$
- **Proposition:** For any concept class C, the optimal mistake bound is bound as follows:

$$VC(C) \leq Opt(C) \leq log_2(|C|)$$