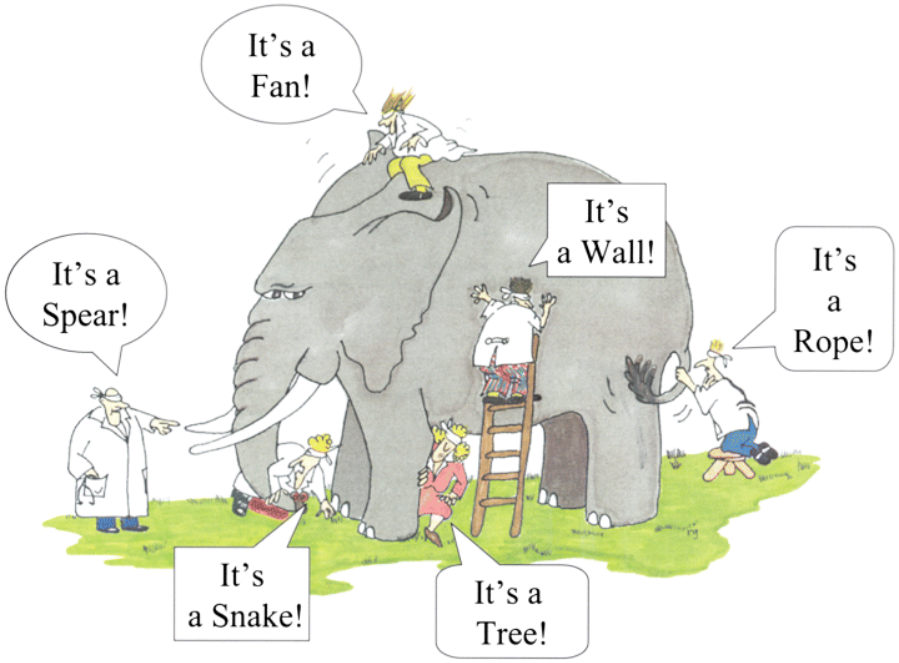
Machine Learning based methods for Invariant Synthesis

PRANAV GARG

P. MADHUSUDAN

UNIV. OF ILLINOIS AT URBANA-CHAMPAIGN



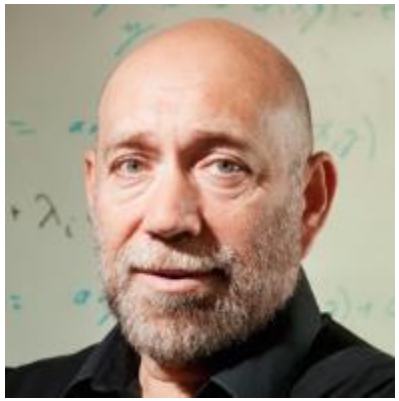
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

ExCAPE
Expeditions in Computer Augmented Program Engineering

NSF

ACKNOWLEDGEMENTS

Daniel Neider (UIUC+UCLA)



Dan Roth (UIUC)

OVERVIEW

1. **Invariant Synthesis and Machine Learning** **(1:15hr)**

   Inductive synthesis of invariants using learning: *iterative ICE model*
   The ``gaps'' we need to bridge in adapting ML algorithms to invariant synthesis

2. **Machine learning: Classification algorithms** **(1:00hr)**
   Learning conjunctions, Linear Threshold Functions, Winnow, Perceptron, SVM, Decision trees with Bool and continuous attribs
   Tool: Weka, C5.0

3. **Building bridges from ML to Inv Synthesis** **(50min)**
   Tool: ICE-DT (with Demo)

4. **Open problems**
   Broader perspective: Inv Synthesis to Program Synthesis **(15m)**

BRIDGING THE GAPS

Gap#1: Ensuring consistent learning

Gap#2: Convergence under iteration (online learning)

Gap#3: ICE

Machine learning is not necessarily a fuzzy theorem-less heuristic-filled world of ad-hoc algorithms!

At one end, there is a very strong theory (computational learning theory) with deep strong results.

But (just like in our world), heuristics are needed to scale and perform well.

← Building bridges to Inv Syn →

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE	Iterative ICE convergence
Conjunctions **Elimination Alg**	1	1	1	2 Houdini	2 **Houdini**
Conjunctions(LTF) **Winnow**	1	1	1	**Open**	--
Conjunctions(LTF) **SVM**	1	1	1	**Open**	--
Boolean functions **Decision trees**	1	1	1	2	2
Bool Threshold fns **Decision trees w/ cont attribs**	1	1	2	2	2

INTRODUCTION: INVARIANT SYNTHESIS AND MACHINE LEARNING

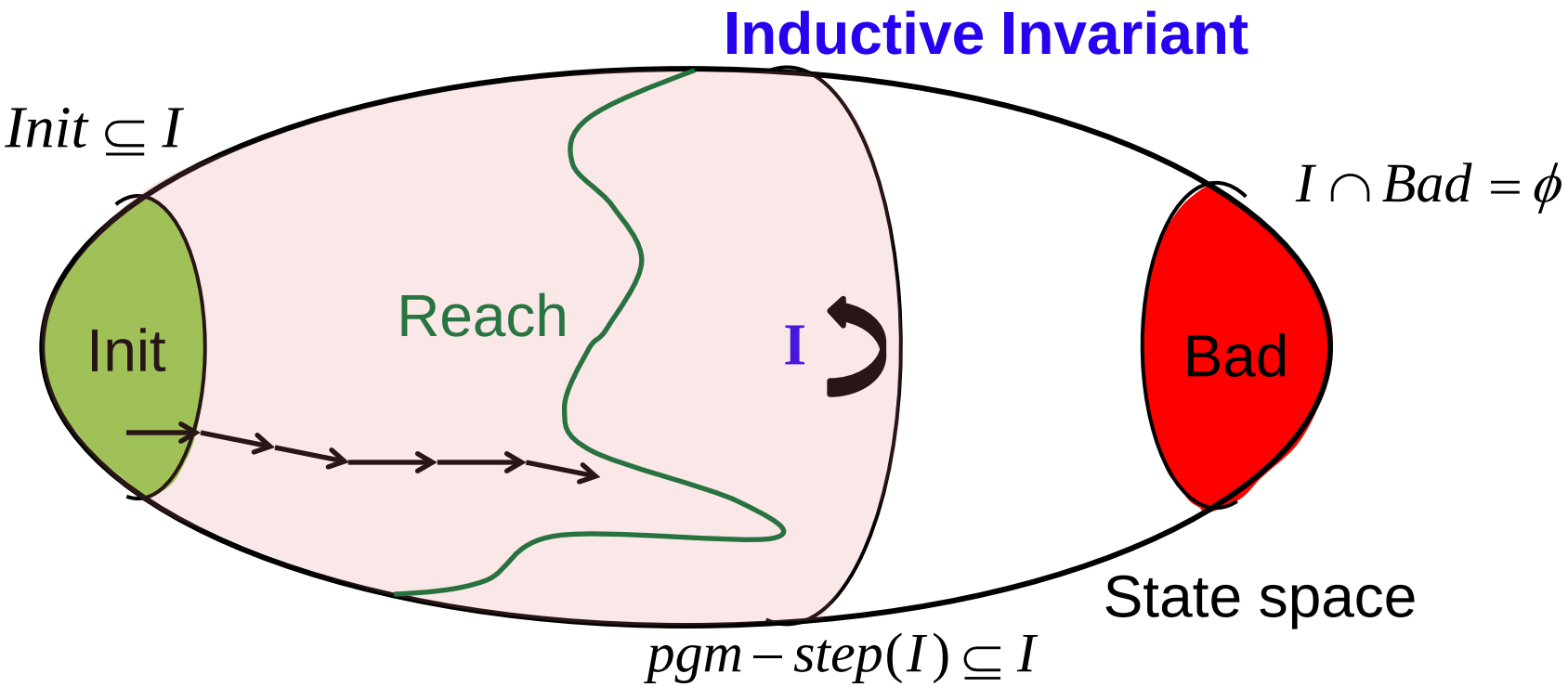
-

- Invariants: why they are important to synthesize
- Inductive synthesis approach to invariant synthesis: Motivation
- Learning from (+,-) samples

- Machine learning: What are they, classifiers, nomenclature and aims
- The *Iterative learning model* for synthesizing invariants
- Why (+,-) examples do not suffice
- The *ICE learning model*

- The ``gaps'' between bridging ML algorithms to invariant synthesis
  - + Consistency (removing fuzziness)
  - + Convergence under iteration
  - + Handling ICE
  - + Convergence for ICE

How does one verify software?



Steps to verify software:

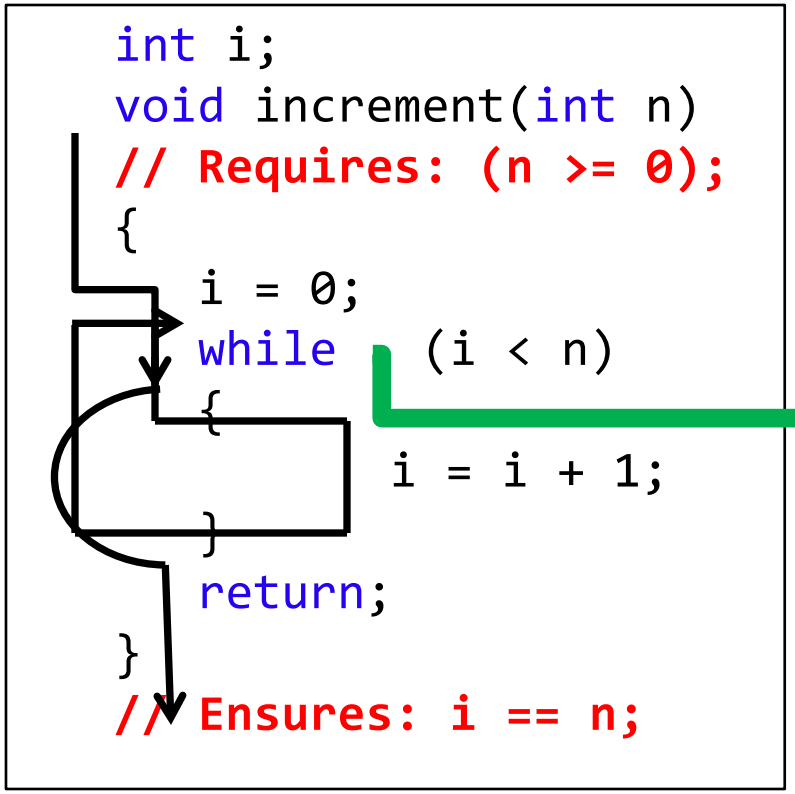
1. Specify an invariant.
2. Validate the invariant:
   - includes Init
   - excludes Bad
   - is inductive

Manual → **Main bottleneck to automatic verification**

Simple properties → Automatic

Complex properties → Manual

Software Verification 101



Step 1. Specify an invariant

```
// Loop invariant: i <= n
```

Step2. Validate the invariant:

1. Includes Init: $n \geq 0 \wedge i = 0 \Rightarrow i \leq n$
2. Excludes Bad: $\neg (i \leq n \wedge i \geq n \wedge i \neq n)$
3. Is inductive: $i \leq n \wedge (i < n \wedge i' = i + 1) \Rightarrow i' \leq n$

FLOYD-HOARE STYLE REASONING

- **User provides specification using modular annotations**

  (pre/post conditions, class invariants, loop invariants)

- **Automatic generation of verification conditions (e.g., Boogie)**

  (pure logic formulas whose validity needs to be verified)

- **Example:** `[|x > y|] x:=x+2; y:=y+1; [|x > y|]`

  gives verification condition

$$\forall x_{old}, x_{new}, y_{old}, y_{new} :$$
$$\left((x_{old} > y_{old} \wedge x_{new} = x_{old} + 2 \wedge y_{new} = y_{old} + 1) \rightarrow x_{new} > y_{new}\right)$$

- **Validity of verification conditions checked mostly automatically**

  (e.g., SMT solvers)

CHALLENGES OF FH REASONING: BURDEN OF ANNOTATIONS

- **Pre/post conditions for logical macro-modules**
  - Natural to specify; absolutely essential for modular verification!
  - This is the specification at the modular level
- **Pre/post conditions for smaller modules**
  - Harder to write/specify

Synthesize

- **Loop Invariants**
  - Much harder to write… awkward as well, with border conditions
  - Programmer **shouldn't** have to write most of them

Synthesize

- **Proof tactics**
  - Exremely hard to write, even by experts
  - Intimate knowledge of underlying platform

Synthesize

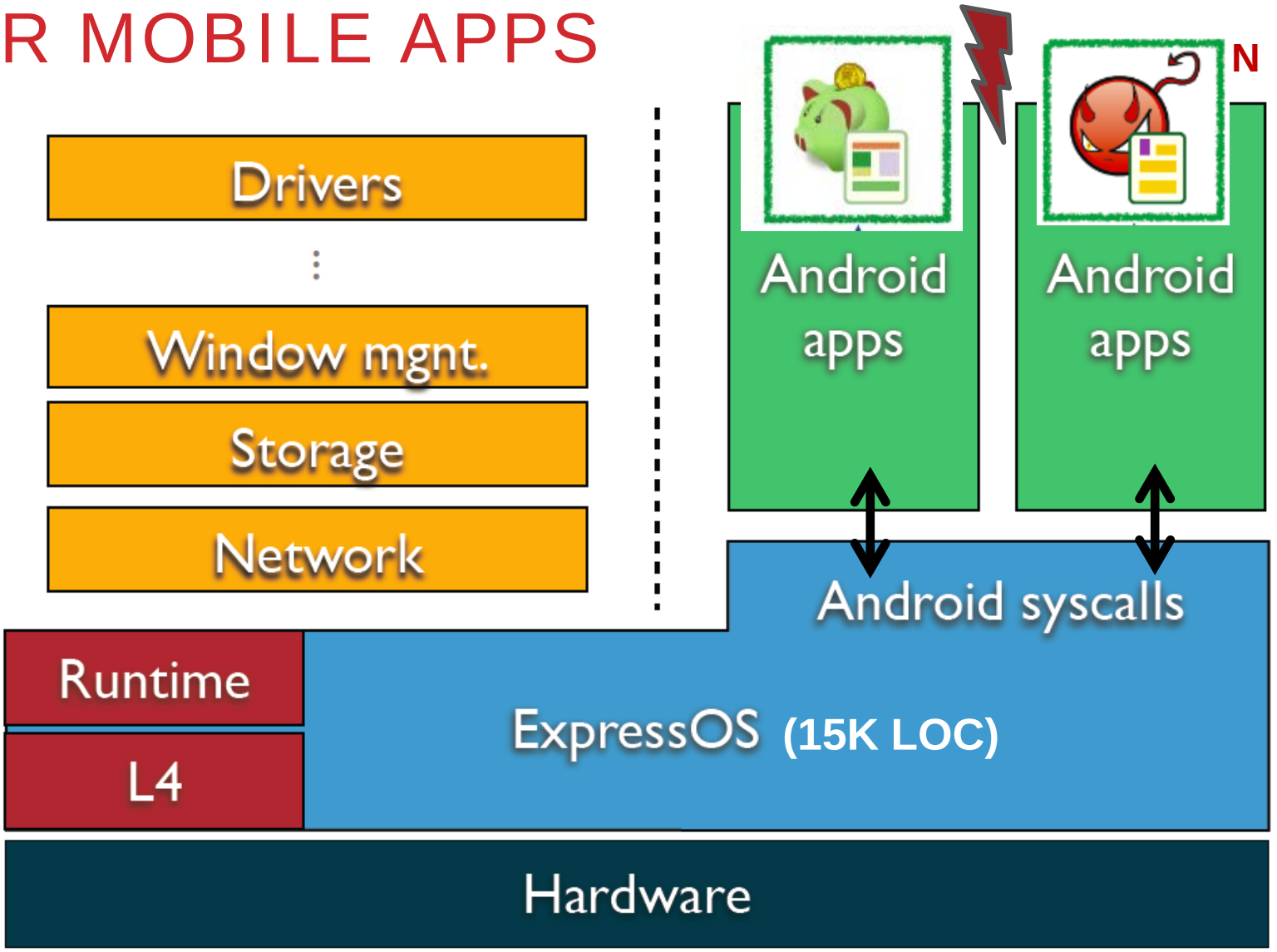
VISION

With *automatic synthesis of annotations and proof tactics* we can provide programmers with tools adequate to build reliable/secure systems with reasonable overhead.

This will enable a new class of reliable software to be built that hasn‘t been possible so far.

You shouldn’t need a PhD in formal methods to use these tools. We can, say over a semester course, be able to enable programmers to write reliable code with proven properties.

EXPRESSOS: A SECURE FOUNDATION FOR MOBILE APPS



ExpressOS fragment: Cache of DiscPages

```
class CachePageHolder {  // this.Head is a sorted list of CachePages
    CachePage Head;  // Class invariant: sorted(this.Head);
    . . .
    CachePage LookupPrev(int k) {
       // Ensures: (ret != NULL => ret.Key <= k);
       // Ensures: (ret != NULL && ret.Next != NULL) =>
                                              ret.Next.Key > k;
       CachePage current = Head, prev = null;
       while (current != null && current.Key <= k) {
              prev = current; current = current.Next;
       } return prev;
    }
    . . .
}
```

ExpressOS fragment: Cache of DiscPages

```
class CachePageHolder {  // this.Head is a sorted list of CachePages
    CachePage Head;  // Class invariant: sorted(this.Head);
    . . .
    CachePage LookupPrev(int k) {
       // Ensures: (ret != NULL => ret.Key <= k);
       // Ensures: (ret != NULL && ret.Next != NULL) =>
                                              ret.Next.Key > k;
       CachePage current = Head, prev = null;
       while (current != null && current.Key <= k) {
              prev = current; current = current.Next;
       } return prev;
    }
    . . .
}
```

ExpressOS fragment: Cache of DiscPages

```
class CachePageHolder {  // this.Head is a sorted list of CachePages
    CachePage Head;  // Class invariant: sorted(this.Head);
    . . .
    CachePage LookupPrev(int k) {
        // Ensures: (ret != NULL => ret.Key <= k);
        // Ensures: (ret != NULL && ret.Next != NULL) =>
                                              ret.Next.Key > k;
        CachePage current = Head, prev = null;
        while (current != null && current.Key <= k) {
                prev = current; current = current.Next;
        } return prev;
    }
    . . .
}
```

```
// Loop invariant:
sorted(this.Head) && ((prev == null && Head ->* current) ||
  (Head ->* prev && current == prev.Next && prev.Key <= k));
```

INDUCTIVE INVARIANT SYNTHESIS

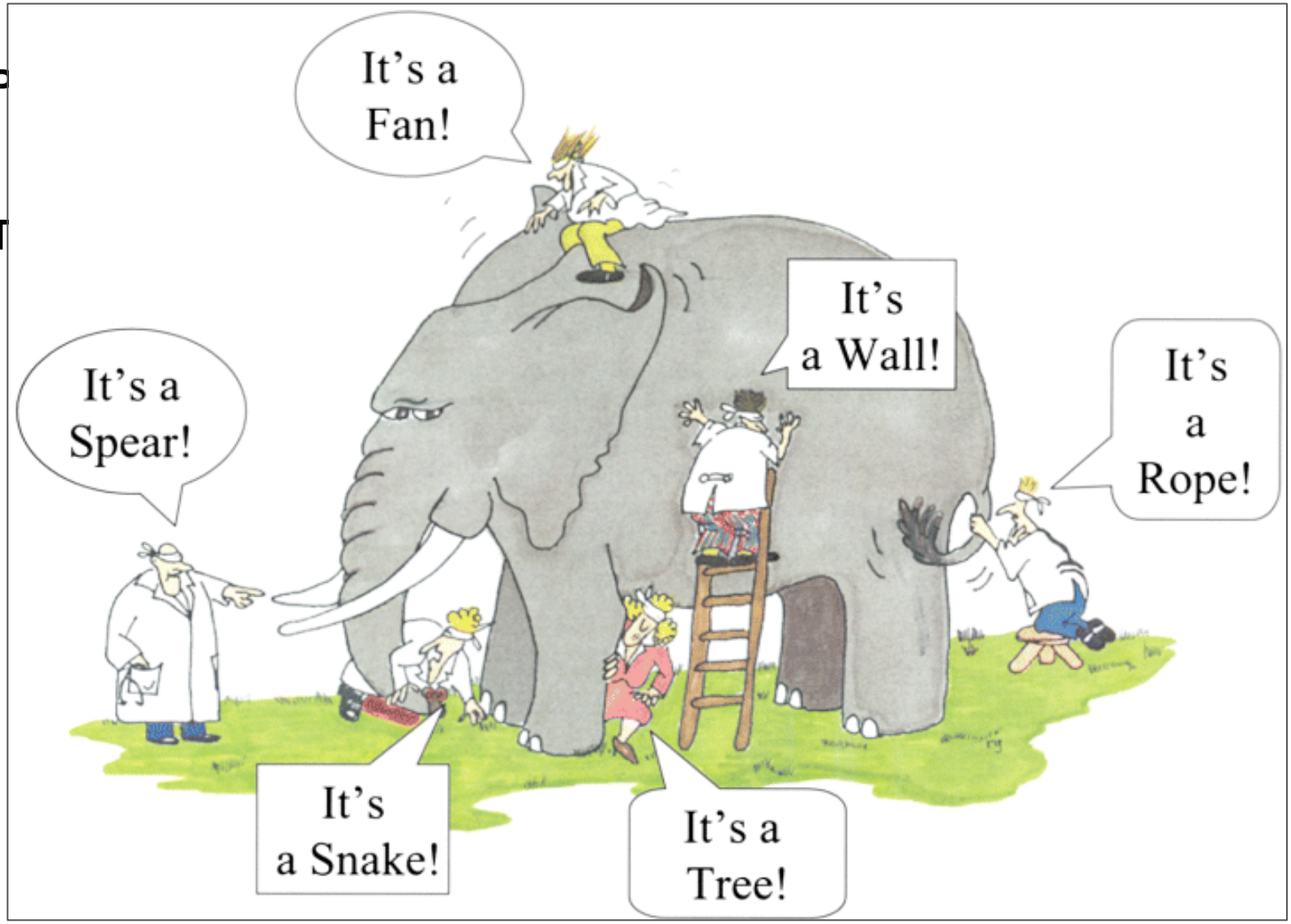
White-box approaches:

- Fixed-point in an abstract domain
- Farkas lemma
- Interpolation
- IC3

White-box approaches (more recently)

- Idea: Learn invariant from sample configurations

SYNTHESIS USING LEARNING



G. Renee Guzlas

WHACKY IDEA?

Black-box invariant synthesis:

Certainly an unusual idea, to hide the program from synthesizer.

Motivation:

- program can be complex:

  complex memory model, complex semantics,

  pointers, heaps, etc.

  user-written annotations can also be complex

- inductive learning ignores this complexity, learning

  only over a restricted observation of
  the state of the program

INDUCTIVE SYNTHESIS

Invariant synthesis:

- Houdini, ICE-learning (stochastic search [Sharma et al 2014], constraint solving, automata [Garg et al 2014] machine learning [Garg et al 2015])

More general synthesis (program synthesis, completing Sketches, etc.)

- Black-box synthesis is the norm!
- CEGIS (Counter-example guided inductive synthesis)
- SyGuS synthesis competition: all solvers are based on CEGIS:
  - Enumerative, stochastic, symbolic, machine learning-based (Alchemist)

    enumerative search using abstract recursive data-types (CVC4)

- …. more on this at the end of this tutorial

MACHINE LEARNING

Traditional Algorithms:

Task that we need the program to do is clear, precise.

We build algorithms that do the task, efficiently.

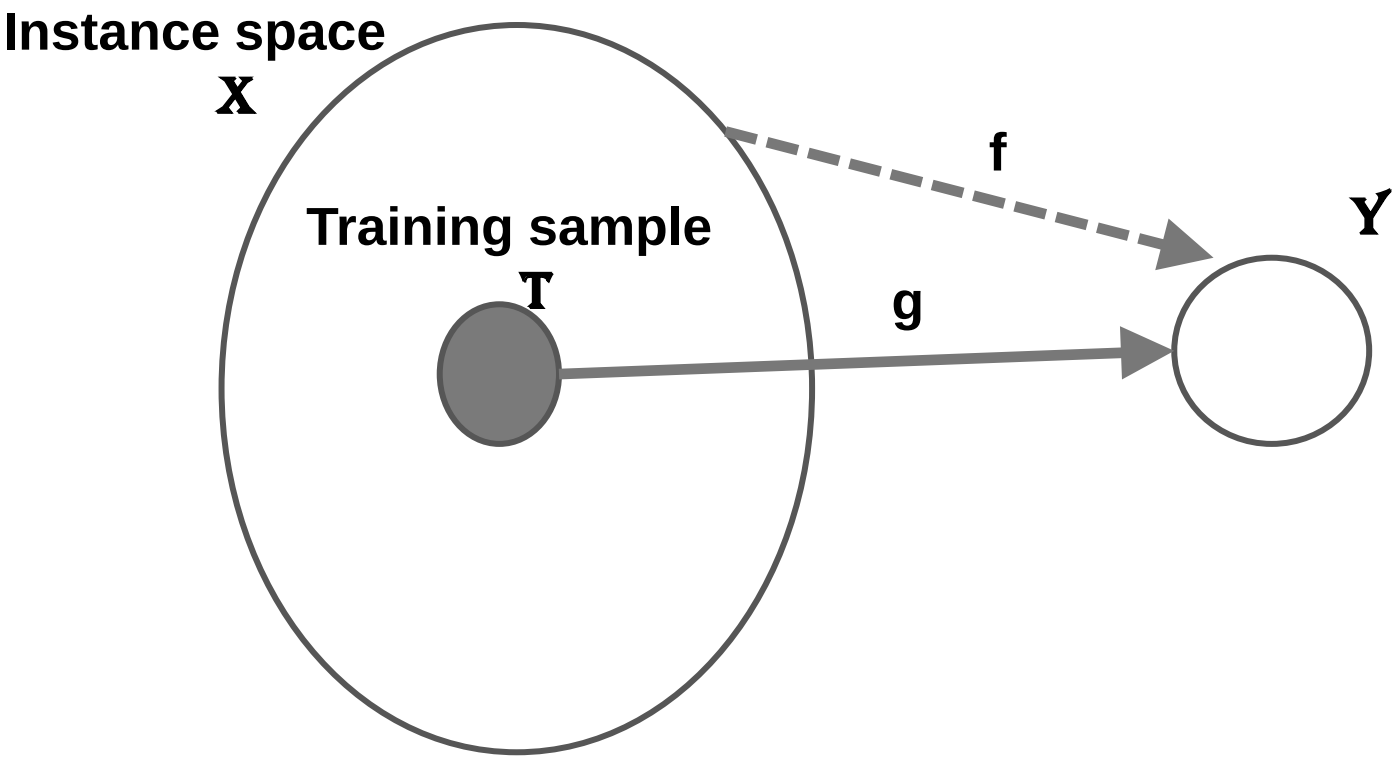
Machine Learning Algorithms:

Algorithms that automatically improve on (possibly ill-defined) tasks through experience.

Improves with experience at some task T
- with respect to some performance measure P
- based on experience E

E.g.: Spam filters, Recognizing faces in a crowd, Credit fraud, Finding the movie to recommend to you from your viewing history

MACHINE LEARNING

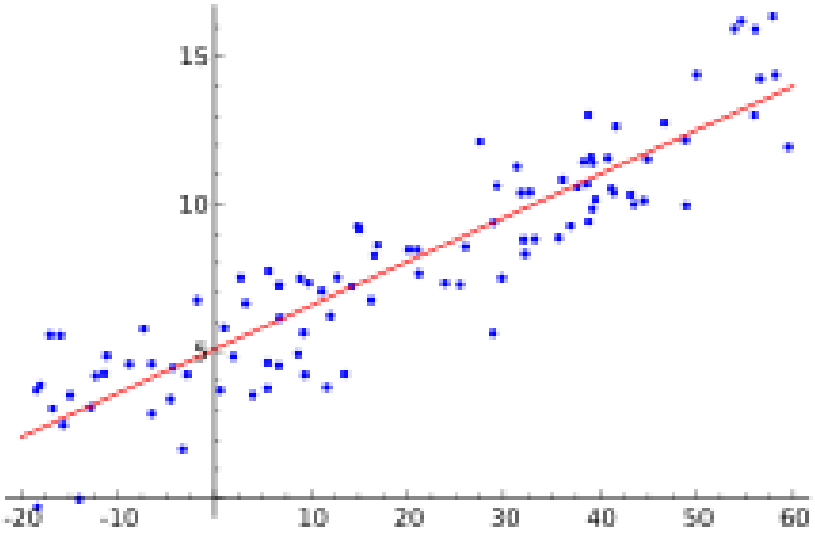


Given $g: T \rightarrow Y$ ``predict'' $f: T \rightarrow Y$

Most important: how well f works on elements not in T or how well f generalizes T

Formalizations: PAC, etc. but not useful for us.

OVERFITTING AND GENERALIZATION



Given g: $T \rightarrow Y$ ``predict'' f: $T \rightarrow Y$

An f that maps T correctly but maps other points to 0 overfits!

In order to generalize, g may not even map T correctly.

MACHINE LEARNING

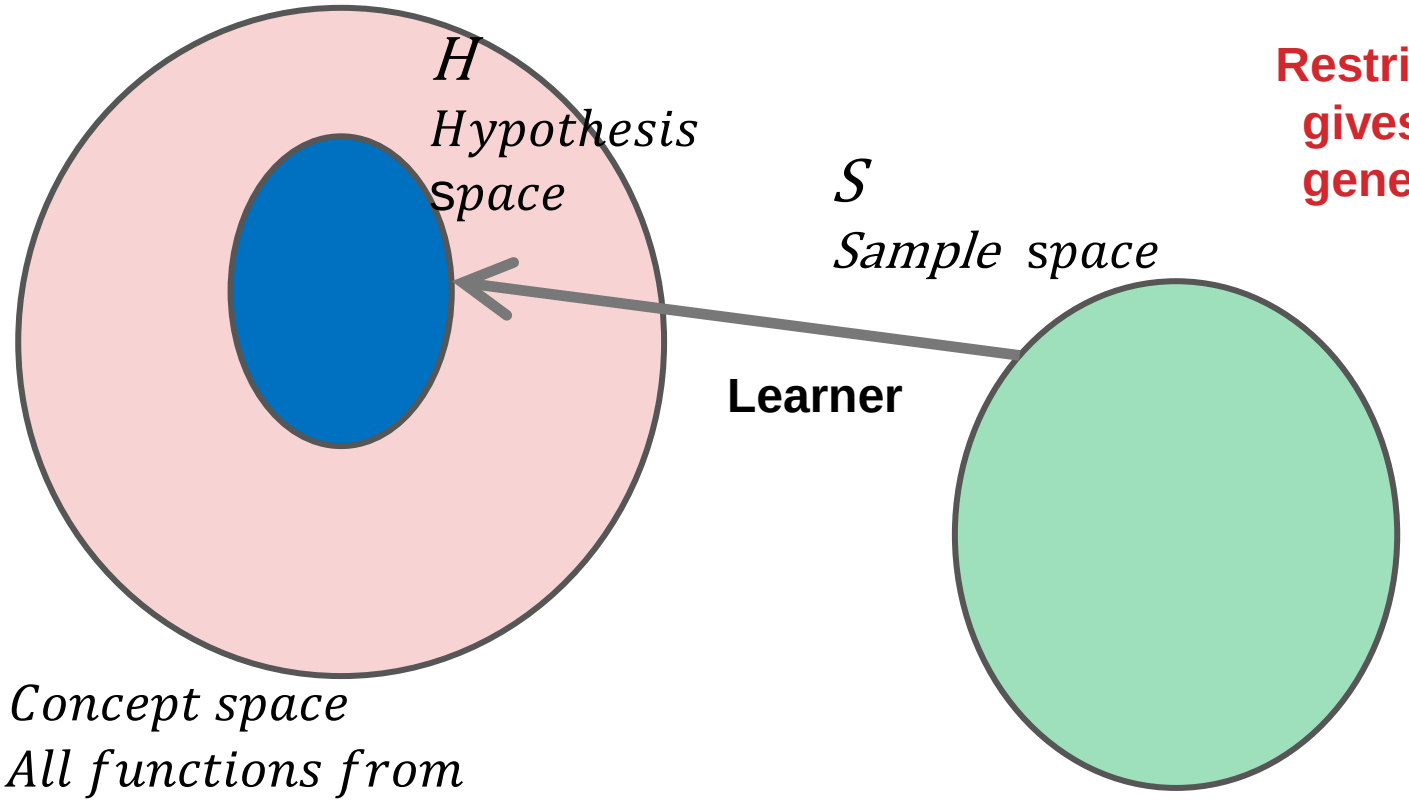
$Set\ \ of\ \ features\ :\mathcal{F}$

$Feature\ extraction\colon\ \mathcal{X}\ \ \rightarrow\ \ \mathcal{F}$

Note:

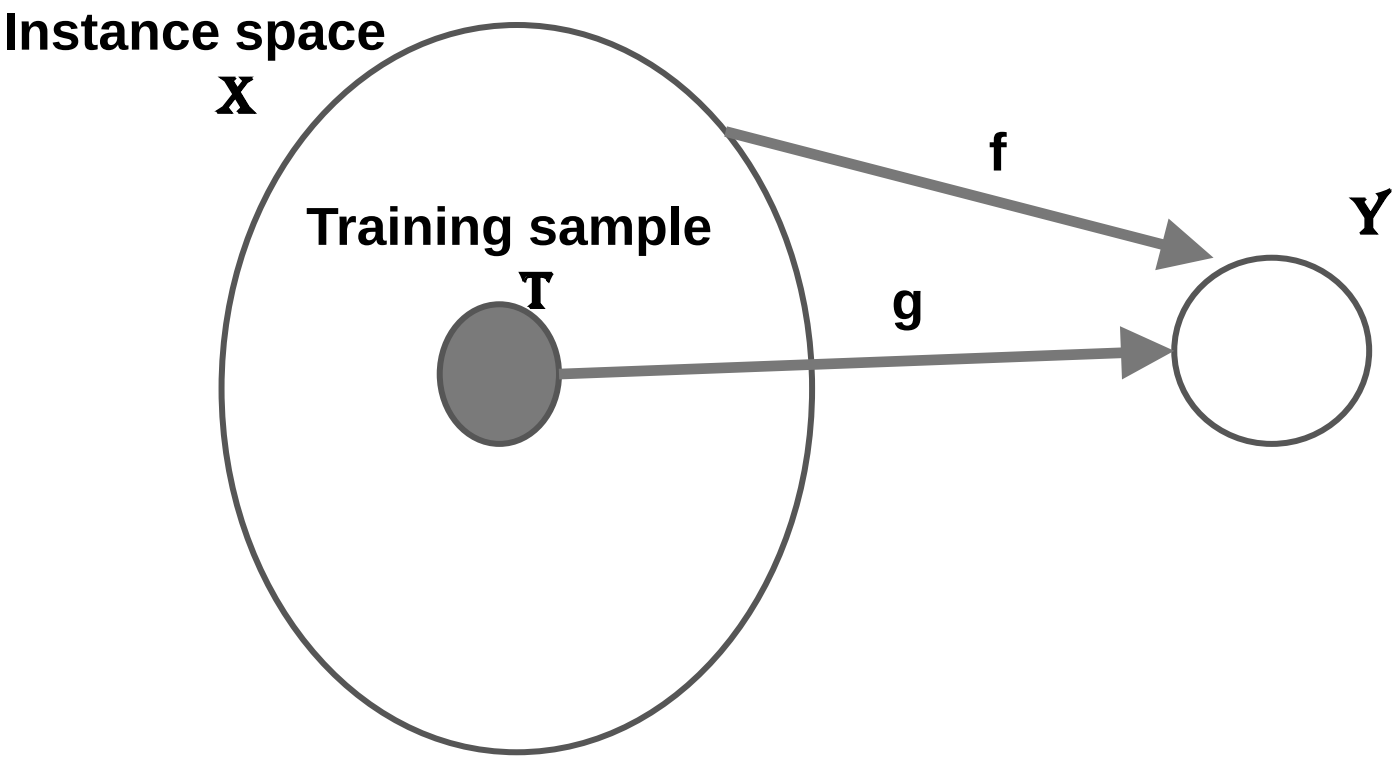
H induces bias

Restricting to H gives some generalization



$C:$ $Concept\ space$
$All\ functions\ from$
$feature\ space\ \mathcal{F}\ to\ \boldsymbol{y}$

LEARNING BOOLEAN FUNCTIONS



$$\mathcal{F} = \{T, F\}^n$$

$$Y = \{T, F\}$$

f: *Boolean functions over* $\{ b_1, \ldots \ b_n\}$

LEARNING BOOLEAN FORMULAS

$$B_n = \{ b_1, \ldots, b_n \}$$

Concept space: All Boolean functions over B_n

Hypothesis space:

- **Conjunctions** over B_n
- **Disjunctions over** B_n
- **k-CNF (or k-DNF) over** B_n
- **Linear Threshold Functions**
  An arithmetization of Boolean functions
- **Arbitrary Boolean formulas over** B_n

OVERFITTING AND GENERALIZATION

Samples : $\{ x_1 \mapsto r_1, x_2 \mapsto r_2, \ \ldots x_n \mapsto r_n \}$

where each x_i is a valuation of B_n and each r_i is T or F

Let hypotheses space be all Boolean functions.

Then the function *f that says*

if $(x = x_1)$ $\boldsymbol{then\ return}$ r_1

$\boldsymbol{elseif}$ $(x = x_2)$ $\boldsymbol{then\ return}$ r_2

...

$\boldsymbol{elseif}(x = x_n)\ \boldsymbol{then\ return}$ r_n

else return True;

overfits the sample.

How do we avoid overfitting?

- Choose a more constrained hypothesis space (e.g., conjuncts)

- Occam's razor: learner must find a "small" function.

SOME MACHINE LEARNING ALGORITHMS FOR LEARNING BOOLEAN FUNCTIONS

HYPOTHESIS SPACE	LEARNING ALGORITHM
Conjuncts	Elimination Algorithm (prefers LARGE conjuncts)
Linear Threshold Functions	Winnow, SVM (margin bias)
Arbitrary Boolean Formulas	Decision tree learning (bias towards small trees)

BRIDGING ML AND INV SYNTHESIS

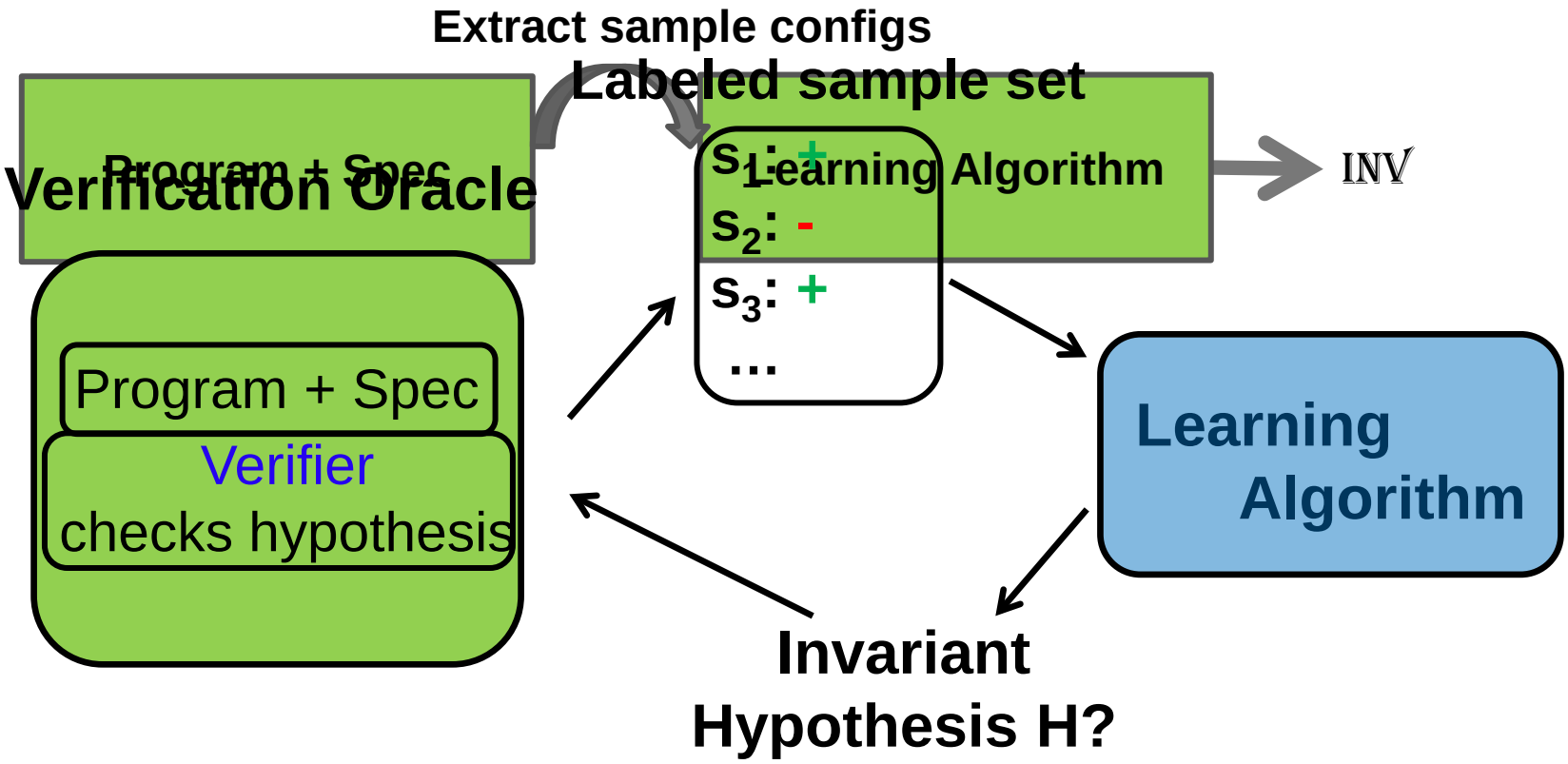
- **Issue #1:**

  ML algorithms need not return hypotheses consistent with (training) sample.
  This is a feature! Not fitting the sample is a means to generalization.

  In invariant synthesis, not having a hypothesis consistent with the sample is awkward.

  Need to address this difference in some way.

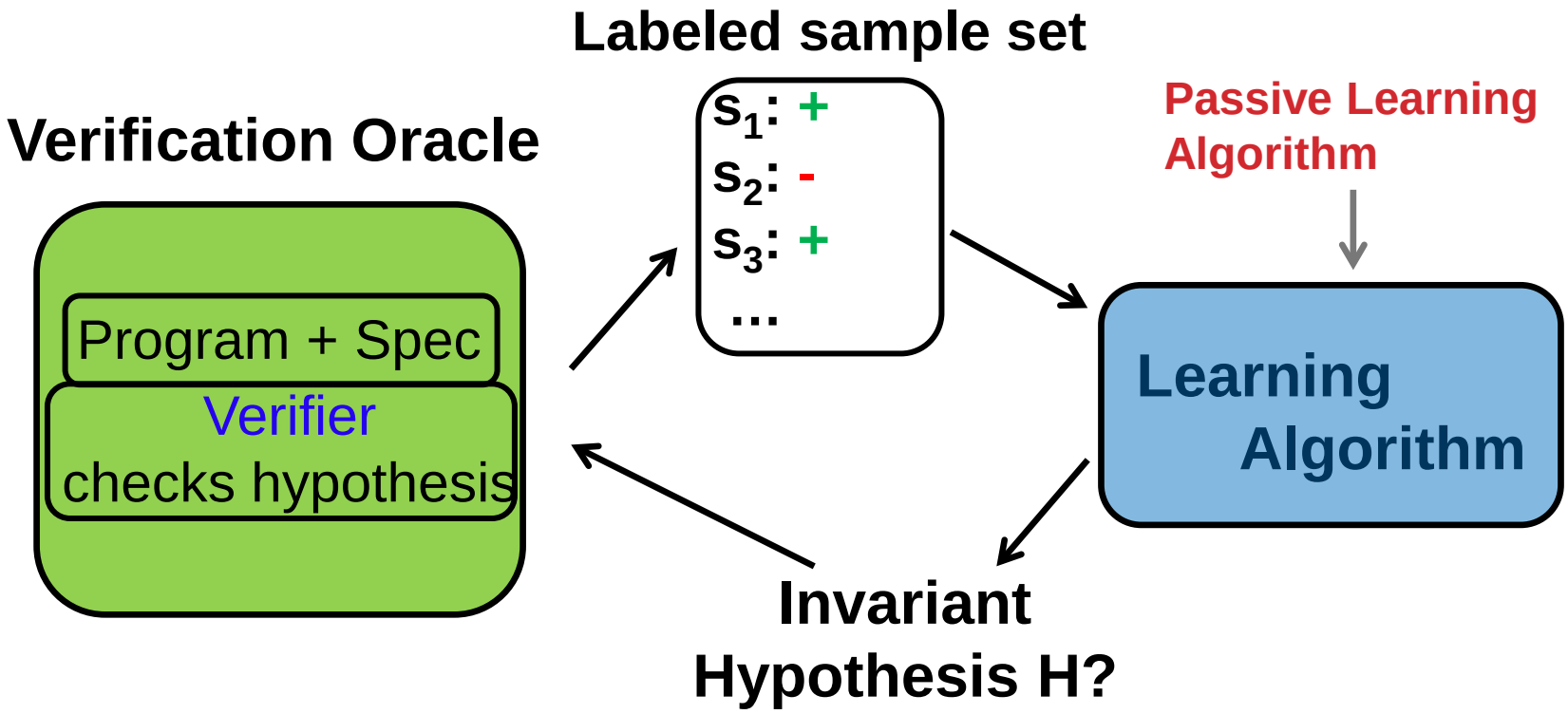
PASSIVE LEARNING ISN'T ENOUGH



Iterative learning: the learner and teacher work in rounds,

If hypothesis is not a valid invariant, the teacher returns a counterexample program configuration appropriately labeled

PASSIVE LEARNING ISN'T ENOUGH



Not really a problem.

We can plug in *any passive machine learning algorithm* for the learning algorithm.

But will this iteration CONVERGE?

ITERATIVE LEARNING

Issue #2:

When we plug in machine learning algorithms in an iterative setting with a verification oracle, will the learning converge?

More simply, assume the teacher knows a UNIQUE invariant and gives counterexamples with respect to it.
Then will the learner eventually learn this invariant?

ONLINE LEARNING MODEL

Online machine learning model: closest to iterative model

Samples arrive in a stream.

Learner continually proposes hypothesis.

E.g.: Stock-market prediction

Some of the algorithms we study have online-learning analogs and analysis (SVMs, Winnow, etc.)

GENERIC MISTAKE BOUND ALGORITHMS

- **Is it clear that we can bound the number of mistakes ?**
- **Let C be a finite concept class. Learn $f \in C$**
- **CON:**
  - In the *i'th* stage of the algorithm:
  - C_i all concepts in C consistent with all i-1 previously seen examples
  - Choose arbitrary $f \in C_i$ and use to predict the next example
  - Clearly, $C_{i+1} \subseteq C_i$ and, if a mistake is made on the ith example, then $|C_{i+1}| < |C_i|$ so progress is made.
- **The CON algorithm makes at most |C|-1 mistakes**
- **Can we do better ?**

THE HALVING ALGORITHM

Let C be a concept class. Learn f ϵ C

Halving:

- **In the ith stage of the algorithm:**
  - C_i : all concepts in C consistent with all i-1 previously seen examples
- **Given an example e_i consider the value $f_j(e_i)$ for all $f_j \in C_i$ and predict by majority.**
- Predict 1 iff

$$|\{f_j \in C_i; f_j(e_i) = 0\}| < |\{f_j \in C_i; f_j(e_i) = 1\}|$$

- **The Halving algorithm makes at most log(|C|) mistakes**

THE HALVING ALGORITHM

- **Hard to compute**
- **In some cases Halving is optimal (C - class of all Boolean functions)**
- **In general, not necessarily optimal!**

  to be optimal, instead of guessing in accordance with the majority of the valid concepts, we should guess according to the concept group that gives the least number of expected mistakes (even harder to compute)

POSITIVE / NEGATIVE LABELS ARE NOT SUFFICIENT

In reality, the teacher does not have ONE invariant.

The teacher knows NO invariants. And yet there may be many.

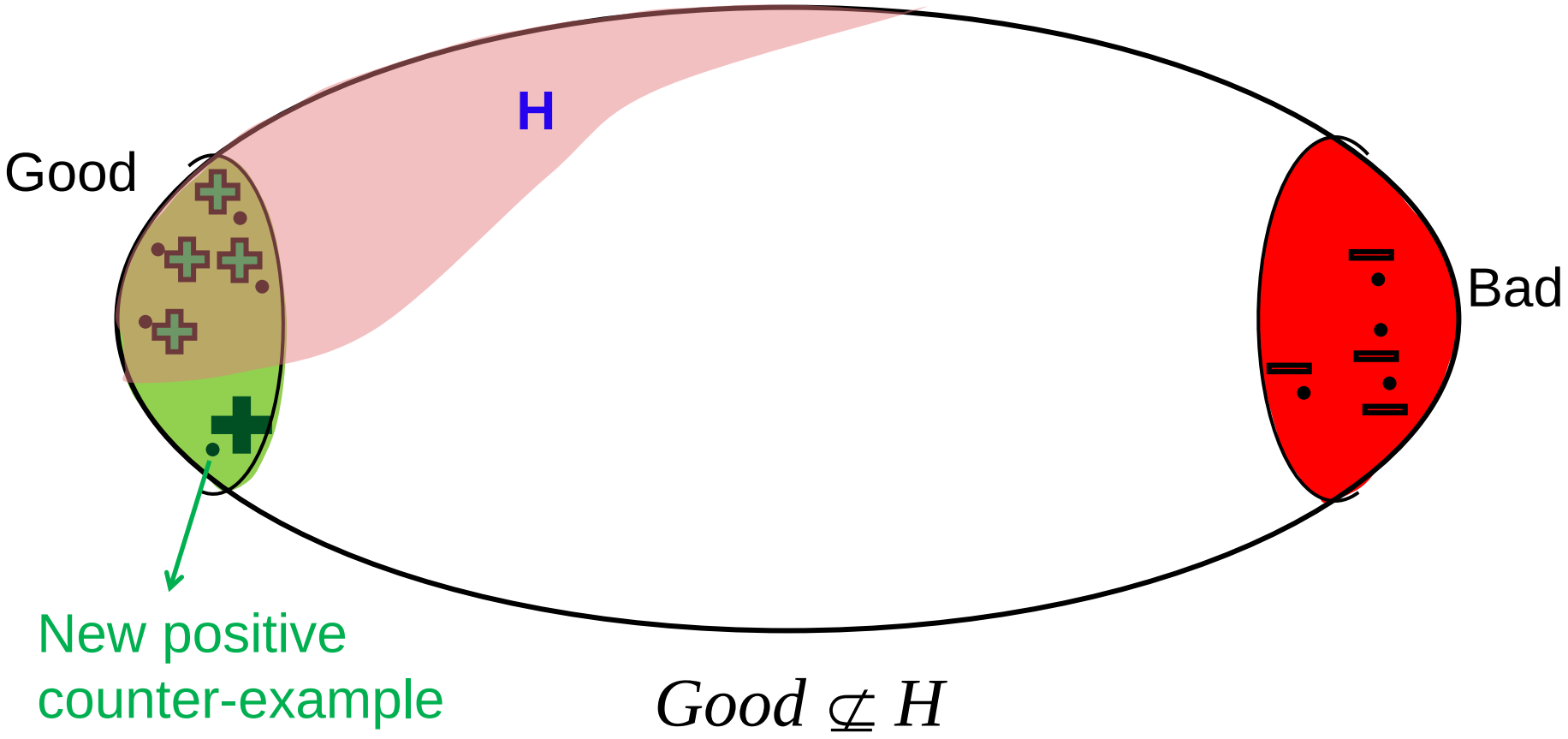
The teacher knows only the *properties* an invariant must satisfy:

1. Includes Init:
2. Excludes Bad:
3. Is inductive:

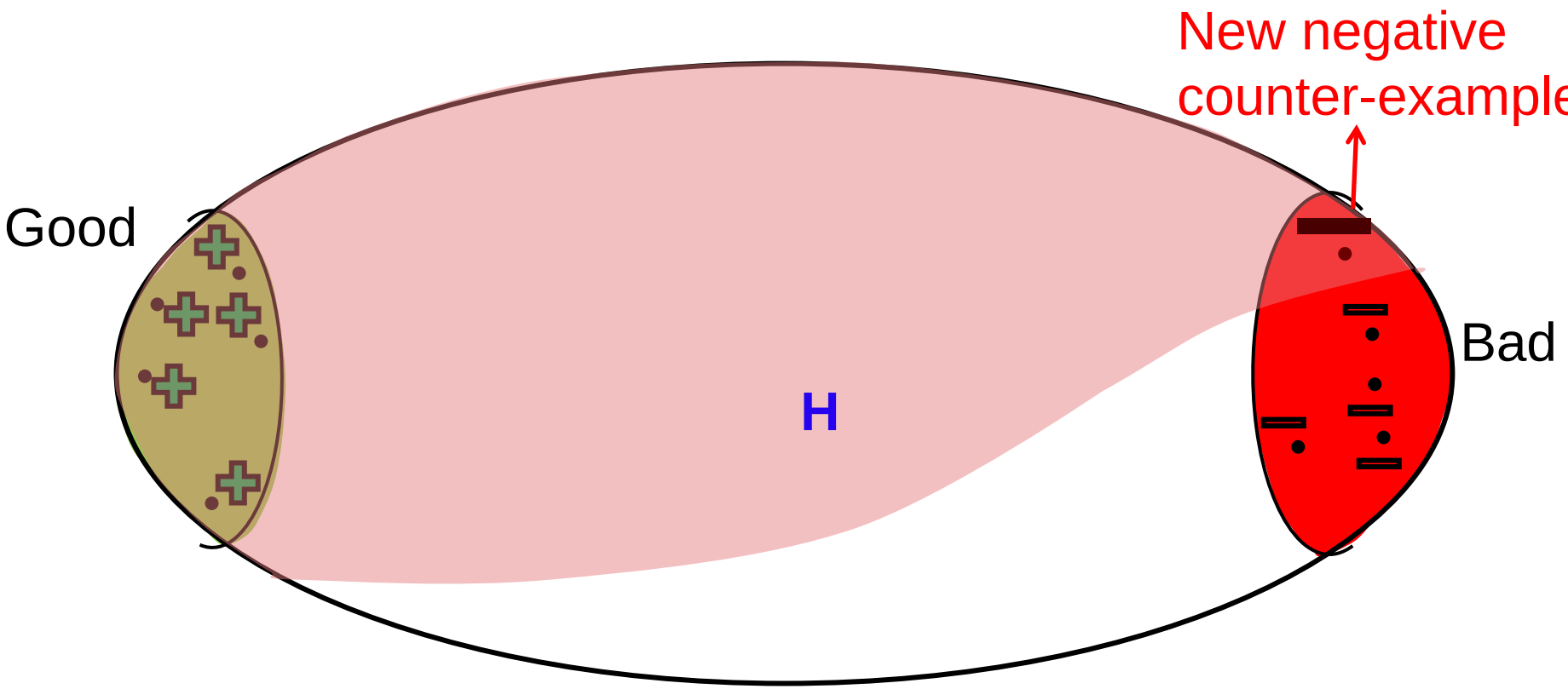
It turns out that the teacher cannot give only

+ or -- labeled counterexamples!

Counterexample: Positive

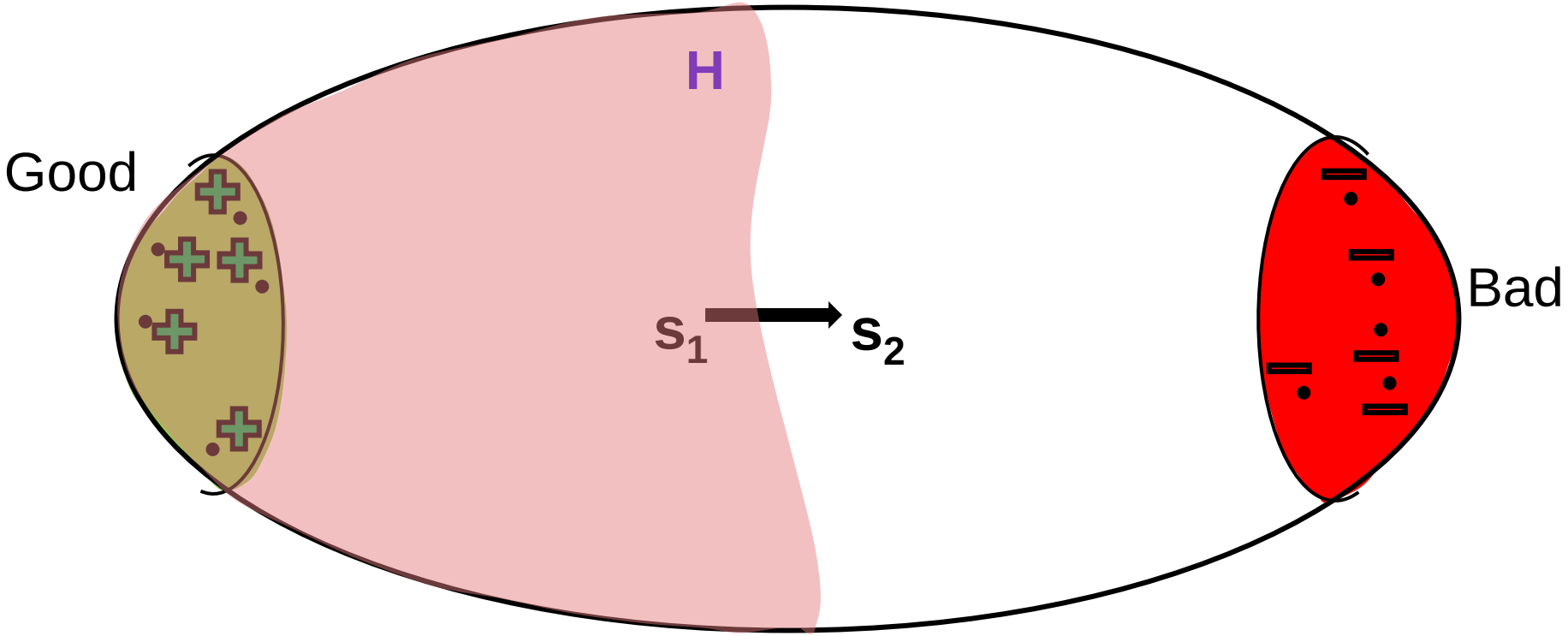


Counter-example: Negative



$$H \cap Bad \neq \phi$$

What happens if the invariant is not inductive?



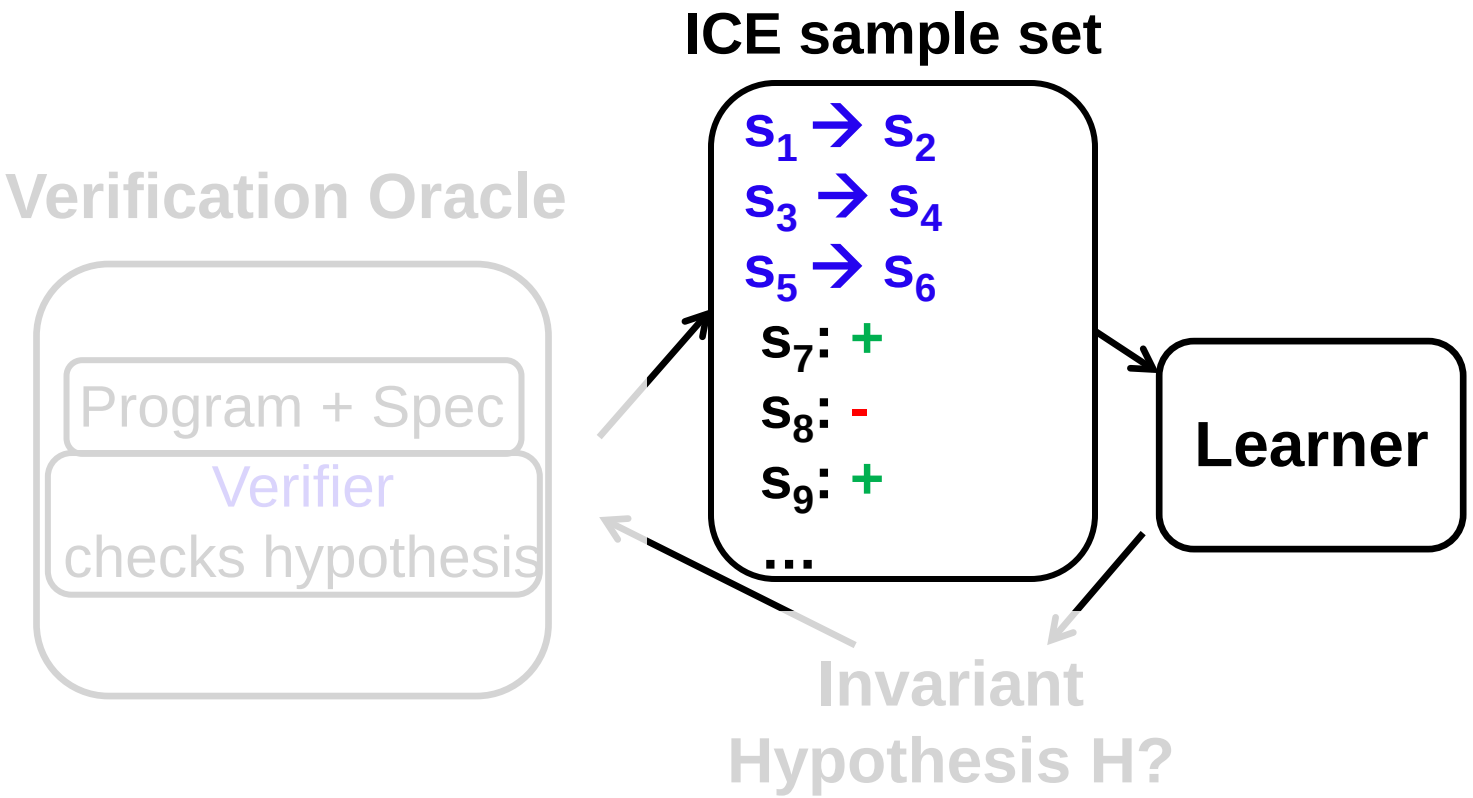
$s_1 \in H$ *and* $s_2 \notin H$

Don’t know whether these states should be included in the invariant or not.

All we know: $s_1 \in H \Rightarrow s_2 \in H$

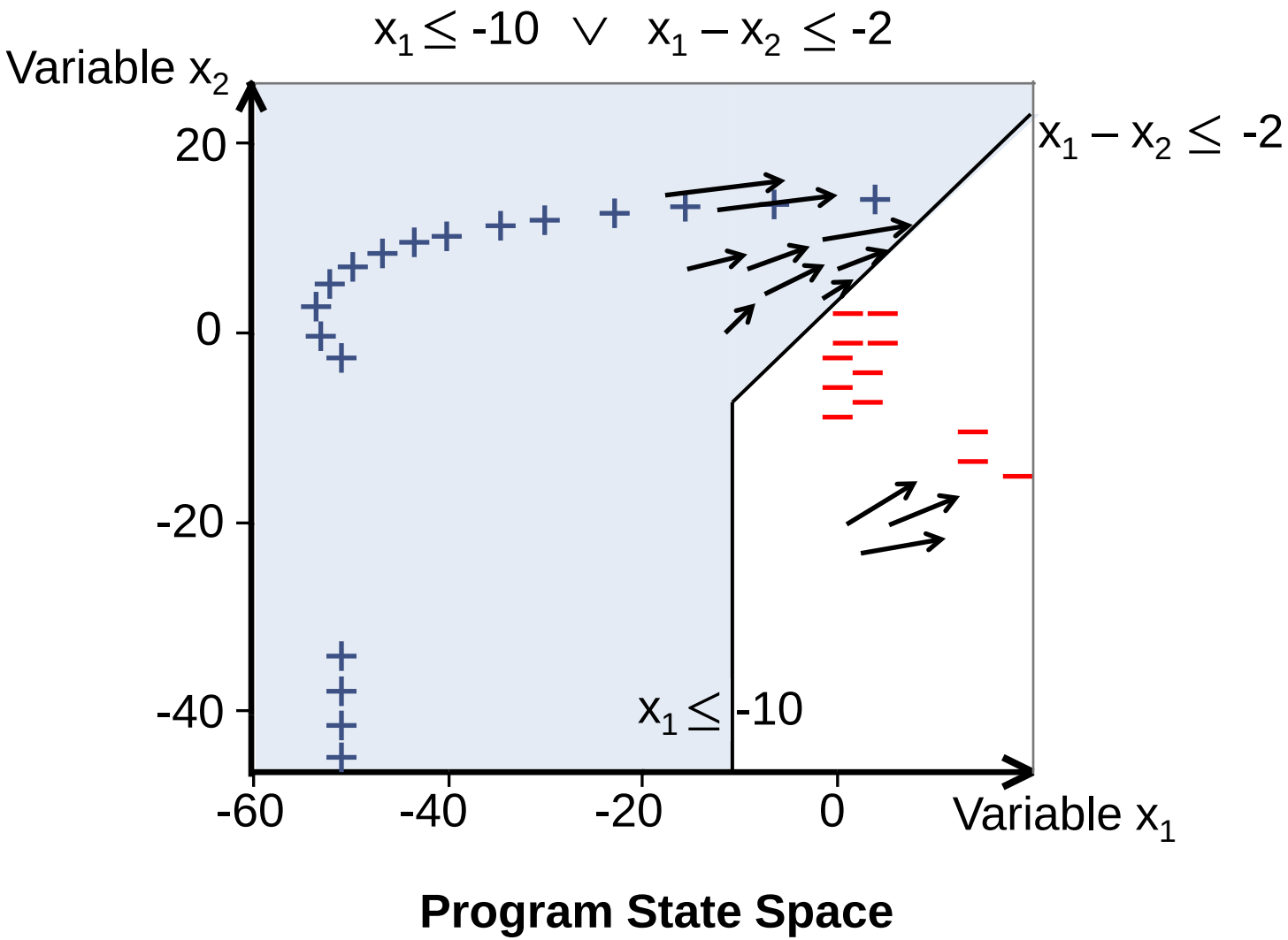
ICE: Learning invariants using Implication Counter-Examples

([Garg et al. CAV 14])



New learning model for robust synthesis of invariants

Learning an invariant given an ICE sample



Program State Space

ICE LEARNING

Issue #3:

Traditional machine learning algorithms learn from (+,-) examples.

How do we extend them so that they can learn from ICE samples?

Issue #4:

And how do we make sure these ICE learning algorithms converge in an invariant eventually?

CHALLENGES TO USING MACHINE LEARNING FOR INVARIANT SYNTHESIS

1. **Removing fuzziness for machine learning algorithms**

   How do we make them learn hypotheses consistent with the sample?

2. **Will they converge, even with (+,-) counterexamples, to any target concept?**

3. **How can we adapt them to handle implication counterexamples (ICE)?**

4. **How do we make these ICE learning algorithms converge?**

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE	Iterative ICE convergence
Conjunctions **Elimination Alg**	1	1	1	2 Houdini	2 Houdini
Conjunctions(LTF) **Winnow**	1	1	1	Open	--
Conjunctions(LTF) **SVM**	1	1	1	Open	--
Boolean functions **Decision trees**	1	1	1	2	2
Bool Threshold fns **Decision trees w/ cont attribs**	1	1	2	2	2

OVERVIEW

1. ~~**Introduction: Invariant Synthesis and Machine Learning (40m)**~~

   ~~Inductive synthesis of invariants using learning: *iterative ICE model*~~
   ~~The ``gaps'' we need to bridge in adapting ML algorithms to invariant synthesis~~

2. **Machine learning: Classification algorithms (1:10hr)**

   Learning conjunctions, Linear Threshold Functions, Winnow, Perceptron, SVM, Decision trees with Bool and continuous attribs
   Tool: Weka, C5.0

3. **Building bridges from ML to Inv Synthesis (1:10hr)**
   Tool: ICE-DT (with Demo)

4. **Open problems**
   Broader perspective: Inv Synthesis to Program Synthesis (20m)

LEARNING CONJUNCTIONS

$\boldsymbol{B_n = \{ b_1, \ldots, b_n \}}$

Concept space: All Boolean functions over $\boldsymbol{B_n}$

Hypothesis space: Conjunctions of atoms in $\boldsymbol{B_n}$

Note: There are $\boldsymbol{O(2^n)}$ such conjuncts

Very simple algorithm:

Given a sample $\{ (p_1, +), (p_2, +), \ldots., (p_r, +), (n_1, -), (n_2, -), \ldots, (n_s, -) \}$

Return the conjunction consisting of every b_i

such that no positive sample $(p, -)$ has $p(b_i) = F$

LEARNING CONJUNCTIONS

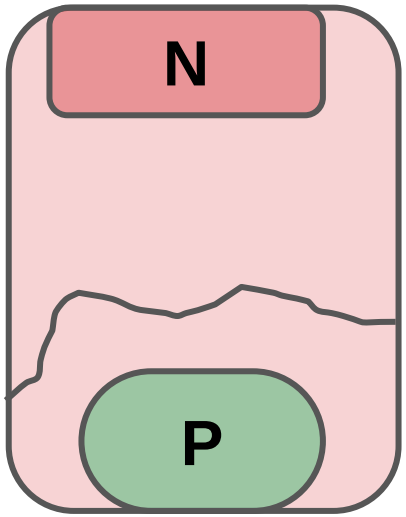
$\boldsymbol{Example}$:

$\boldsymbol{B_5 = \{ b_1, \ldots, b_5 \}}$

Given sample

P = ((1,1,1,1,1),+), (1,1,0,1,0), +)

N= ((0,0,0,0,0),-), ((1,0,0,1,0),-)



return $b_1 \wedge b_2 \wedge b_4$

Why? This is semantically the smallest set that excludes

includes all positive points.

It will exclude the negative points

(provided some conjunction satisfies the sample).

So, if it doesn't satisfy N, we can abort..

ITERATIVE SETTING (+,-)

Initially, when sample is empty propose

$$b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

In each round, learner will propose the semantically smallest set that contains the positive points.

Hence teacher can never give a negative example!

Whenever teacher returns (p, +),

``knock off'' all conjuncts b_i such that p[b_i] = F

Converges in O(n) rounds! (learning from 2^n concepts)

[This is the algorithm Daikon uses.]

LEARNING DISJUNCTIONS

We can also learn disjunctions.

To learn $b_1 \vee b_2 \vee b_4$

we can learn instead its negation:

$$\neg b_1 \wedge \neg b_2 \wedge \neg b_4$$

Complement the samples; treat Boolean variables as negated.

Once you learn the formula, negate it.

Caveat: Assumption that b_i is logical.

K-CNF AND K-DNF

For a (small) fixed k, we can also learn k-CNF formulae:

- Introduce new variables for each possible disjunction of k variables (n^k blowup)
- Compute samples accordingly.
- Learn a conjunction of these variables.

Similarly, k-DNF.

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	2 Houdini	2 Houdini
Conjunctions(LTF) **Winnow**	1	1	1	Open	--
Conjunctions(LTF) **SVM**	1	1	1	Open	--
Boolean functions **Decision trees**	1	1	1	2	2
Bool Threshold fns **Decision trees w/ cont attribs**	1	1	2	2	2

LINEAR THRESHOLD FUNCTIONS

Widely used in machine learning.

Let features be *reals* $\boldsymbol{X_n = \{ x_1, \ldots, x_n\}}$

Learning functions from $\mathbb{R}^n \longrightarrow \{T, F\}$

Hypothesis space:

$$\boldsymbol{\sum_{i=1}^{n} w_i x_i \geq \theta} \quad \text{where } w_i,\ \theta \in \mathbb{Q}$$

Example: $0.43x_1 + 3.2x_2 - 1.57x_3 \geq 3.5$

LINEAR THRESHOLD FUNCTIONS

Linear threshold functions can express *Boolean functions*.

< T, F, F, T, F > → < 1, 0, 0, 1, 0 >

$\mathbb{B}^n \rightarrow \mathbb{R}^n$

$\sum_{i=1}^{n} \boldsymbol{w_i x_i} \geq \boldsymbol{\theta}$ then represents a Boolean function.

Disjunctions: $b_1 \vee b_9 \vee b_{15}$

Represented by LTF: $1.x_1 + 1.x_9 + 1.x_{15} \geq 1$

At least two of b_1, b_3, b_7 must be true:

Represented by LTF: $1.x_1 + 1.x_3 + 1.x_7 \geq 2$

Cannot express XOR. Cannot express nontrivial DNF.

LINEAR THRESHOLD FUNCTIONS

Linear threshold functions can hence express some class $\mathbb{C}$ of Boolean functions.

$\mathbb{C}$ includes disjunctions, and several other functions.

LTFs make the discrete Boolean function space more "continuous", allowing for more robust learning algorithms in some cases.

WINNOW ALGORITHM

$\boldsymbol{B_n = \{ b_1, \ldots, b_n \}}$

Assume n is very large.

But that we are learning a set of at most r disjuncts r << n

Iterative Elimination algorithm: Could take O(n) rounds.

Can we do better?

Yes. O(r log n) rounds! [Littlestone]

In invariant synthesis applications, very interesting, since we can throw in a large number of predicates, though invariant will be a small set of conjuncts.

WINNOW ALGORITHM

$\boldsymbol{B_n = \{ b_1, \ldots, b_n\}}$ **r << n**

Let's iteratively learn a linear threshold function.

$$\sum_{i=1}^{n} \boldsymbol{w_i x_i} \geq \boldsymbol{n}$$

Initialize each $\boldsymbol{w_i} = 1$.

If we get a positive counterexample $< v_1, v_2, \ldots., v_n >$

then for every i such that $v_i = T$ **set** $\boldsymbol{w_i ::= 2w_i}$

If we get a negative counterexample $< v_1, v_2, \ldots., v_n >$

then for every i such that $v_i = T$ **set** $\boldsymbol{w_i ::= \frac{w_i}{2}}$

Beautiful proof showing that this converges in O(r log n) rounds

if a target concept has r conjuncts only.

Note: $O(2^r n^r)$ *concepts and hence halving takes* $O(r \log n)$ *rounds.*

WINNOW ALGORITHM

Theorem: **The Winnow learns the class of disjunctions with mistake bound of $2 + 3r(\log n + 1)$, when the target concept f is an OR of r variables.**

Proof: Assume target is a disjunct of elements in T.

$$\sum_{i=1}^{n} w_i x_i \geq n$$

- **Mistakes on positive examples:**

  On any positive example, we will *double wt of least one element in T.* And a negative example cannot reduce wt of any element in T.

  Can't make positive mistake when at least one weight of T is $\geq n$.

  So #post-mistakes is at most r(log n).

- **Mistakes on negative examples:** Consider $W = \sum_{i=1}^{n} w_i$.

  Each positive mistake increases W by at most n.

  (Since $\sum_{i=1}^{n} w_i \leq n$ is what caused example to be negative.)

  Each negative mistake decreases W by at least n/2.

  Some simple math gives #neg-mistakes < 2#pos-mistakes +2

  So total #-mistakes $\leq 3r \log n + 2$

WINNOW ALGORITHM FOR ICE?

Open Problem:

Is there an iterative learning algorithm that can learn target functions with r conjuncts over n Boolean variables in time O(r log n)?

Or poly in r, and sublinear in n.

PERCEPTRON ALGORITHM

$\boldsymbol{B_n = \{ b_1, \ldots, b_n\}}$

Iteratively learn a linear threshold function.

$$\boldsymbol{\sum_{i=1}^{n} w_i x_i \geq n}$$

Update rule is *additive:*

If we get a positive counterexample $< v_1, v_2, \ldots, v_n >$

then for every i such that $v_i = T$ **set** $\boldsymbol{w_i ::= w_i + \alpha}$

If we get a negative counterexample $< v_1, v_2, \ldots, v_n >$

then for every i such that $v_i = T$ **set** $\boldsymbol{w_i ::= w_i - \alpha}$

Mistake bounds exists; depend on the margin [Novikoff'63].

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	**2** Houdini	**2** Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	**Open**	--
Conjunctions(LTF) **SVM**	**1**	**1**	**1**	**Open**	--
Boolean functions **Decision trees**	**1**	**1**	**1**	**2**	**2**
Bool Threshold fns **Decision trees w/ cont attribs**	**1**	**1**	**2**	**2**	**2**

AND NOW TO PRANAV…

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	2 Houdini	2 Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	Open	--
Conjunctions(LTF) **SVM**	1	1	1	Open	--
Boolean functions **Decision trees**	1	1	1	2	2
Bool Threshold fns **Decision trees w/ cont attribs**	1	1	2	2	2

Support Vector Machines

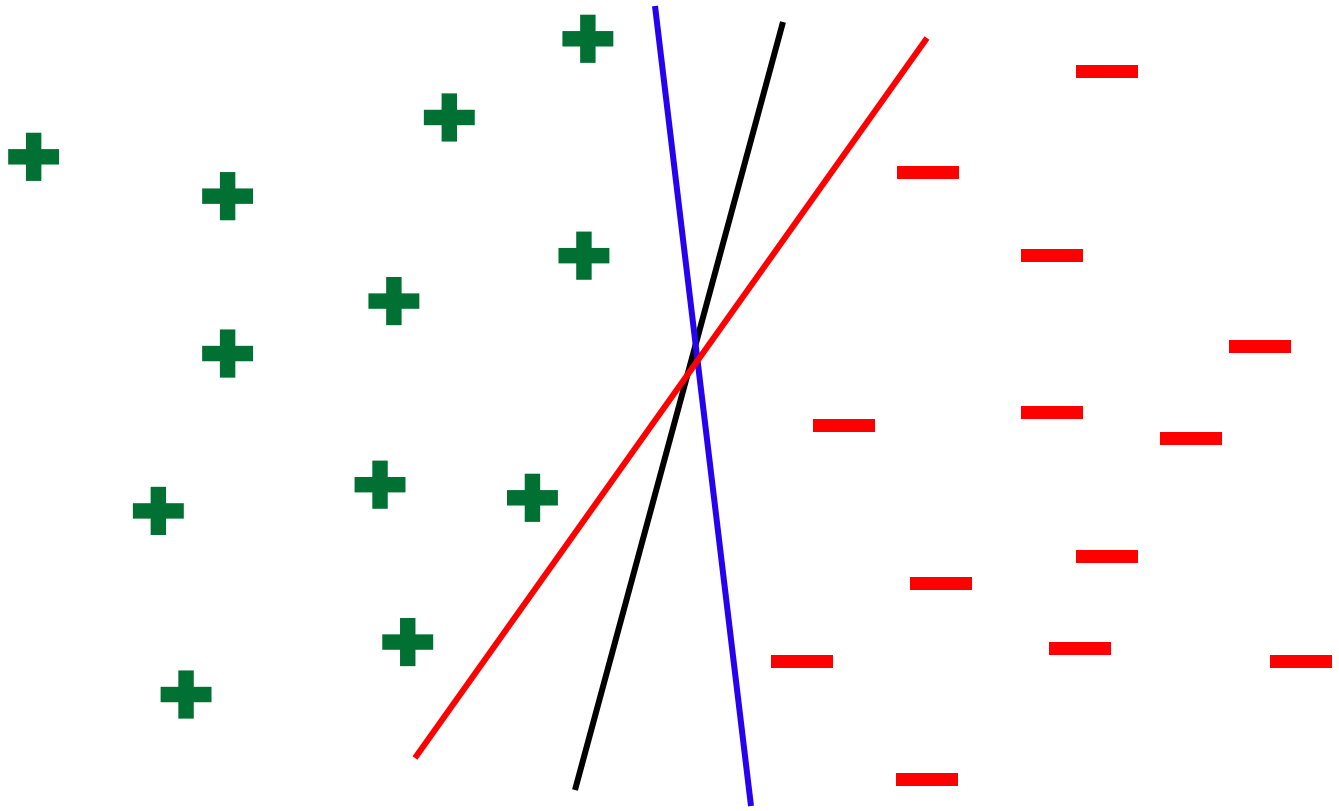
Learn $\sum_{1 \le i \le n} w_i x_i + b \ge 0$, where $x_i \in \{0,1\}^n$

- represents a Boolean function

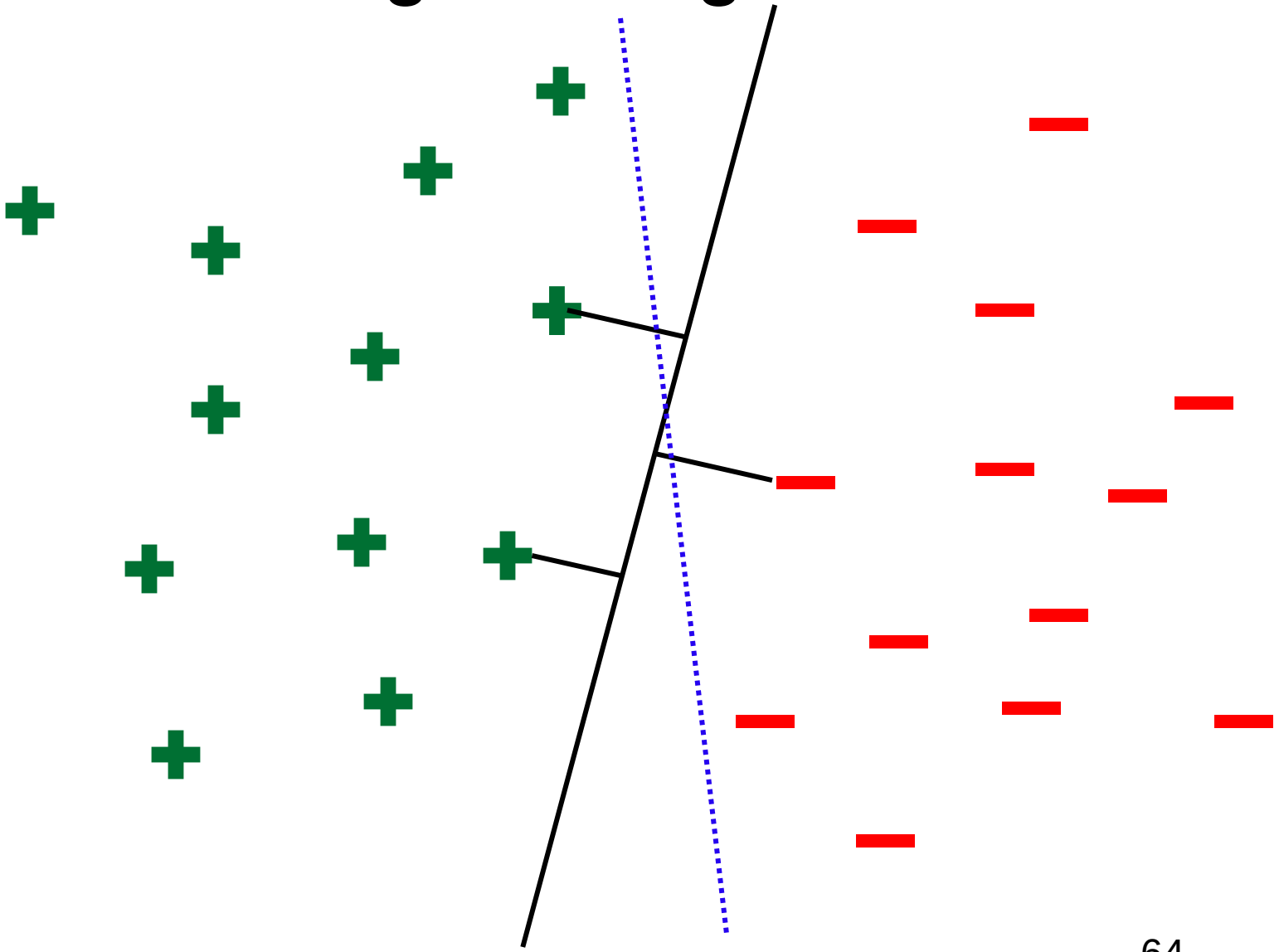
Support Vector Machines

Learn $\sum_{1 \le i \le n} w_i x_i + b \ge 0$, where $x_i \in \{0,1\}^n$

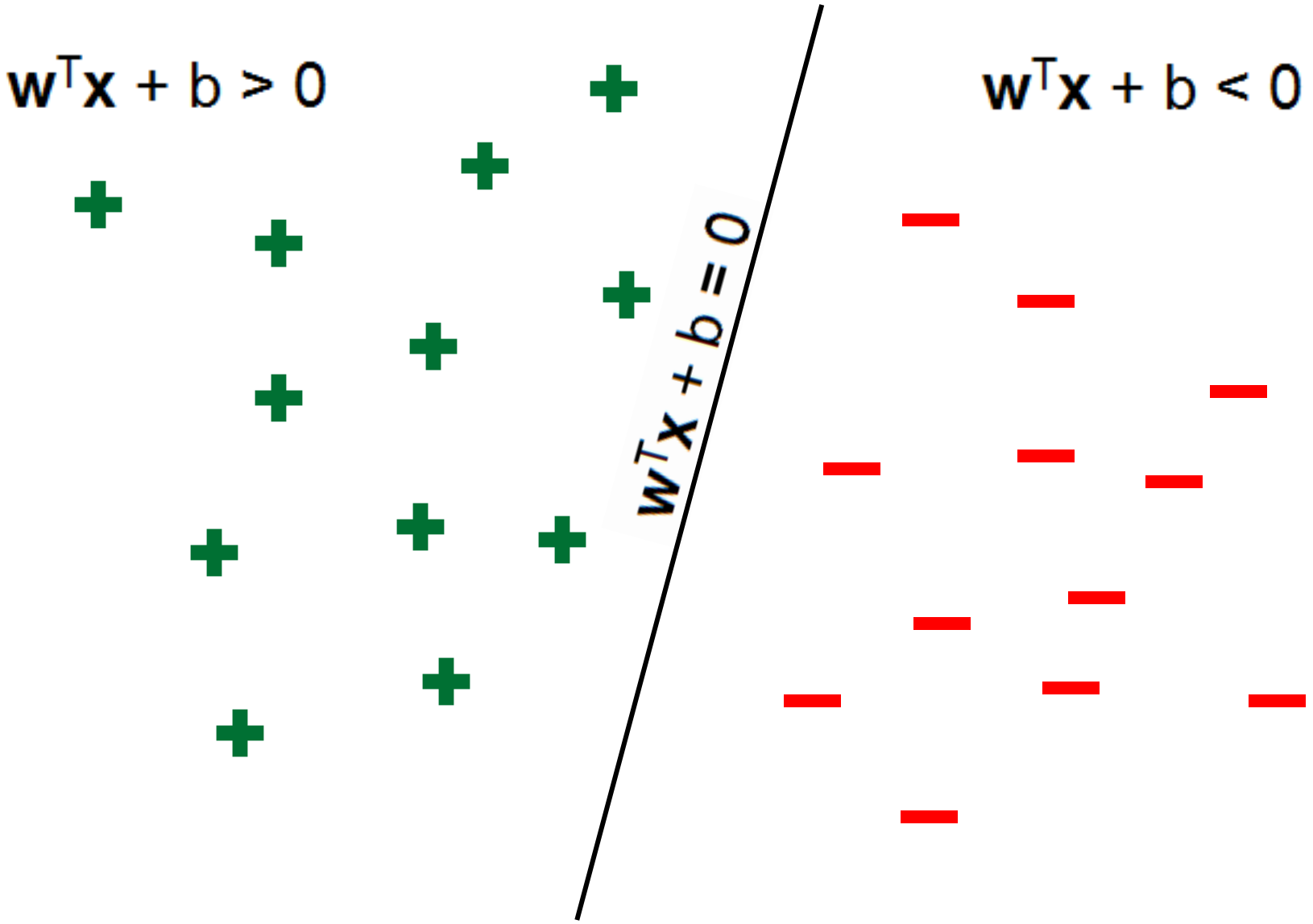
- represents a Boolean function



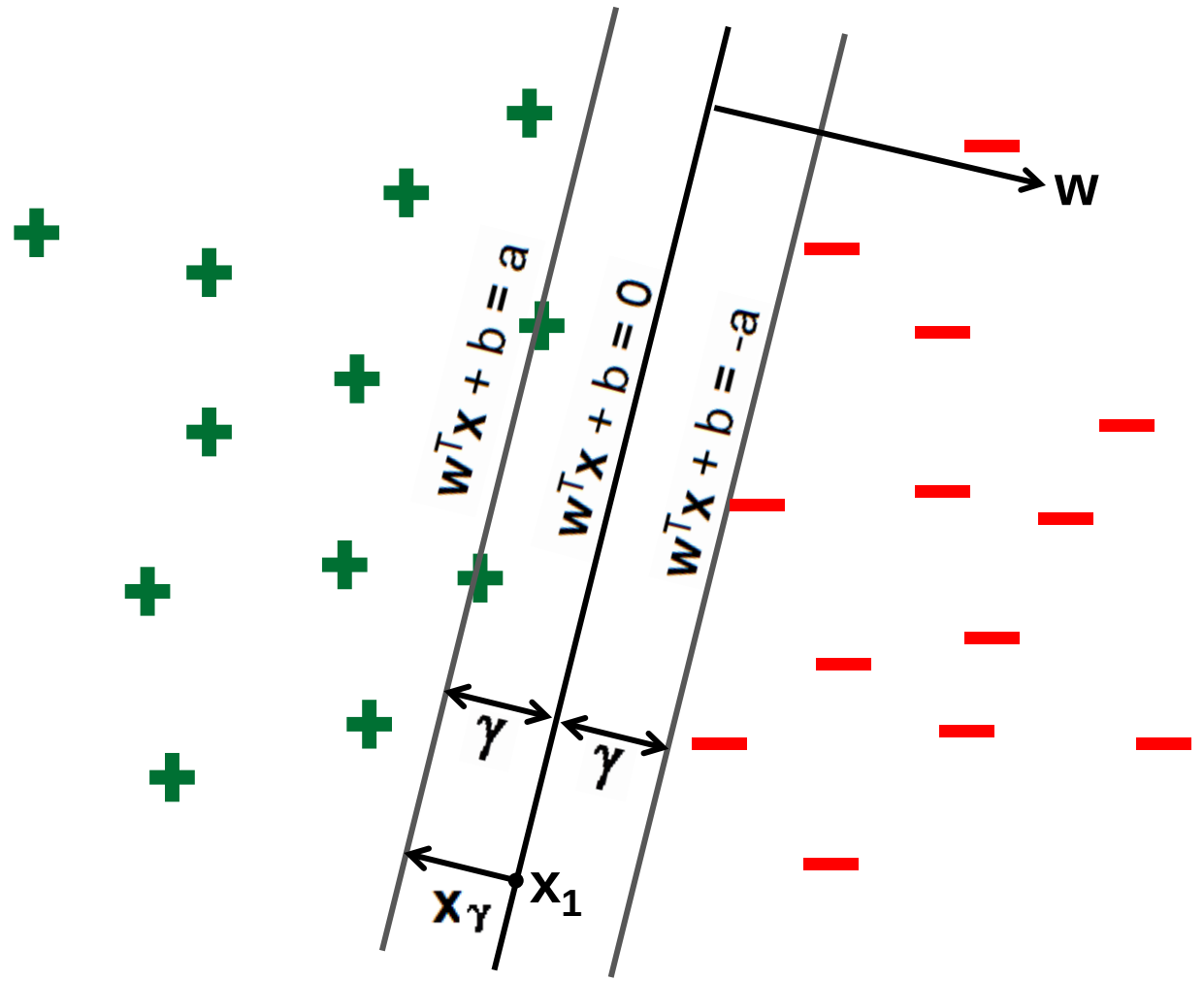
Support vector machines: pick the one with the largest margin!



Parameterizing the decision boundary



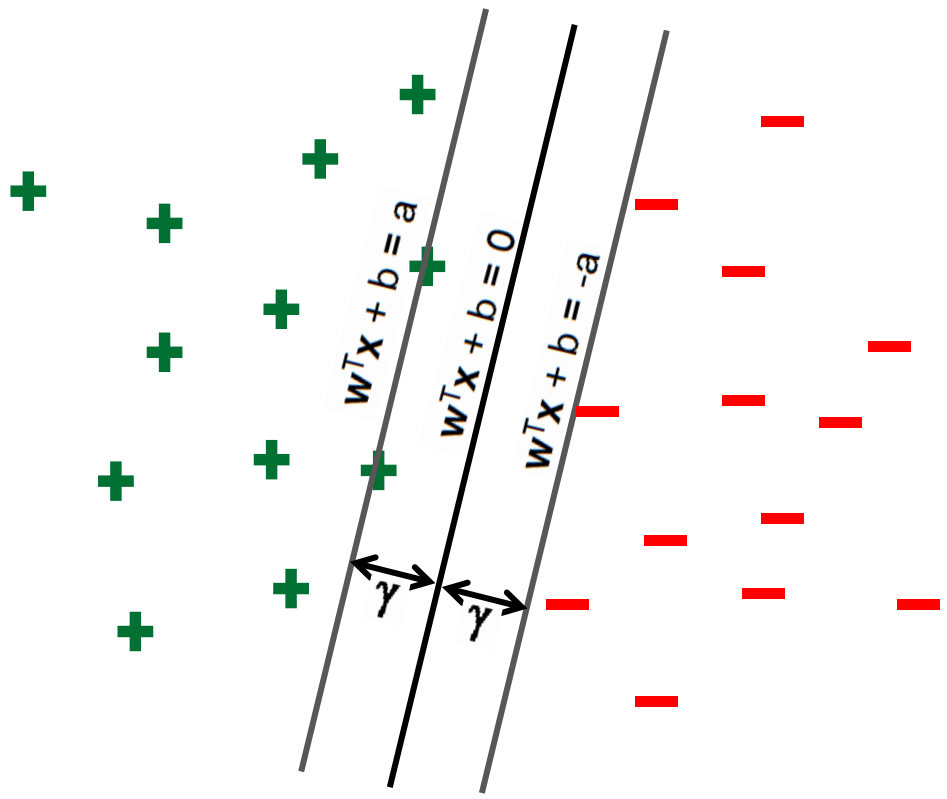
How do we compute the margin?



$\mathbf{w}^T.(\mathbf{x}_1 + \mathbf{x}_\gamma) + b = a$; $(\mathbf{w}^T. \mathbf{x}_1 + b) + \mathbf{w}^T. \mathbf{x}_\gamma = a$;

$\|\mathbf{w}\| . \|\mathbf{x}_\gamma\| = |a|$ ➔ margin $\gamma = |a| / \|\mathbf{w}\|$

Support Vector Machines

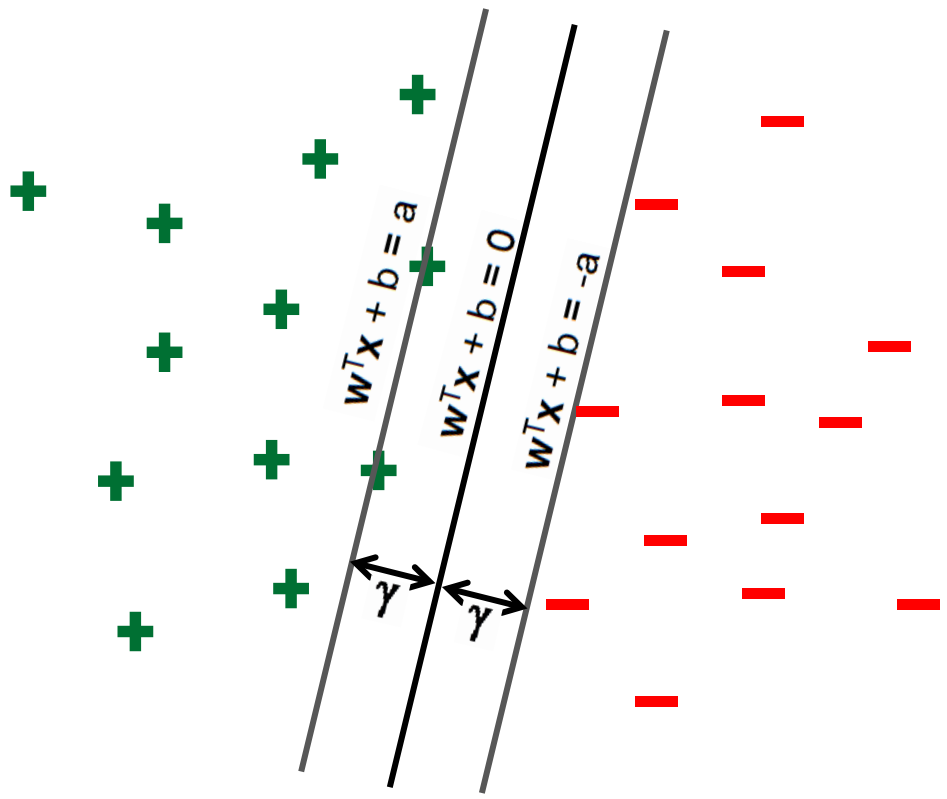


Labels: $y_i \in \{+1,-1\}$

$$\max_{w,b} \frac{\cancel{a}\ 1}{\|\mathbf{w}\|}$$

$$\text{where, } (\mathbf{w}^T \mathbf{x_i} + b)\, y_i \geq \cancel{a}\ 1$$

Support Vector Machines

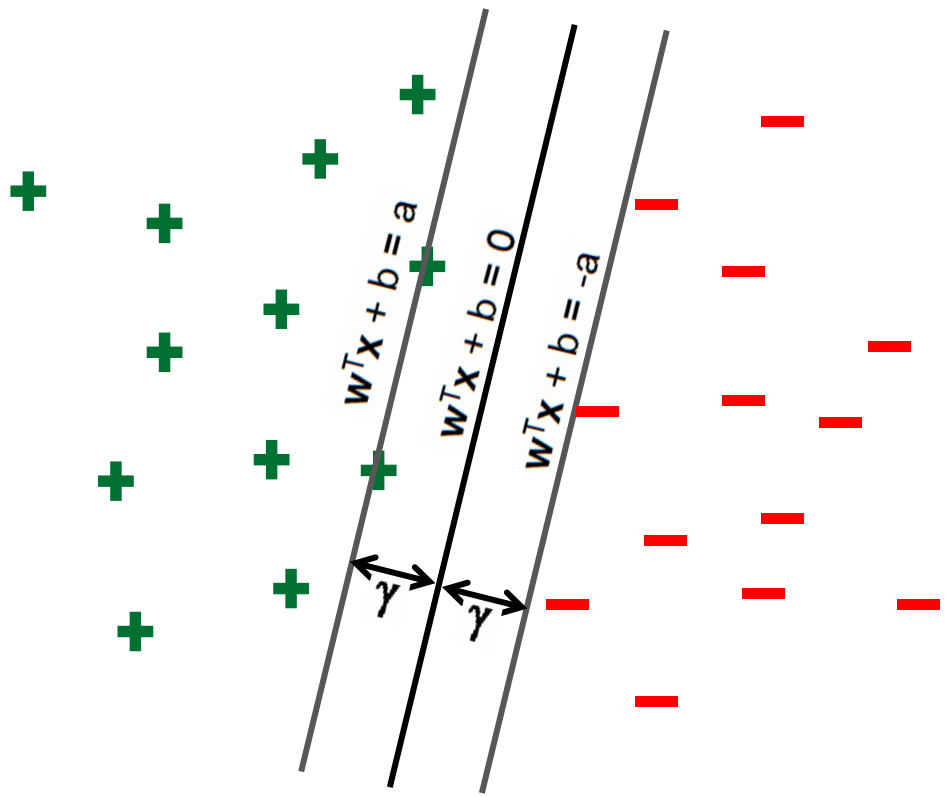


Labels: $y_i \in \{+1,-1\}$

$$\max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|}$$

$$\text{where, } (\mathbf{w}^T \mathbf{x_i} + b)\, y_i \geq 1$$

Support Vector Machines



Labels: $y_i \in \{+1,-1\}$

$$\min_{\mathbf{w},b} \| \mathbf{w} \|^2$$

$$\text{where, } (\mathbf{w}^T \mathbf{x_i} + b)\, y_i \geq 1$$

- Quadratic programming (QP) problem:
  - QP solvers
  - gradient descent

Consistency and Iterative Convergence

If Boolean formula is expressible as a linear threshold function, the concept (LTF) learned is consistent with the sample.

- Extensions to soft-margin SVMs

Convergence

- There are a finite number of Boolean formulas
- Current hypothesis is (semantically) different from all previous hypotheses.
  - Sample contains at least one counterexample for every previous hypothesis

Above learning algorithm is convergent in the iterative setting

Support Vector Machines for ICE?

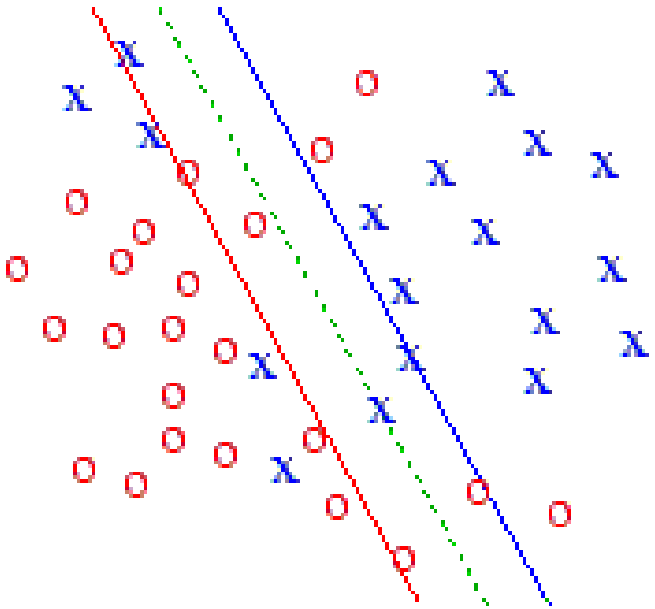
Open Problem:

- How do you define margins?

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE samples	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	2 Houdini	2 Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	Open	--
Conjunctions(LTF) **SVM**	✓	✓	✓	Open	--
Boolean functions **Decision trees**	1	1	1	2	2
Bool Threshold fns **Decision trees w/ cont attribs**	1	1	2	2	2

If the sample is not linearly separable …

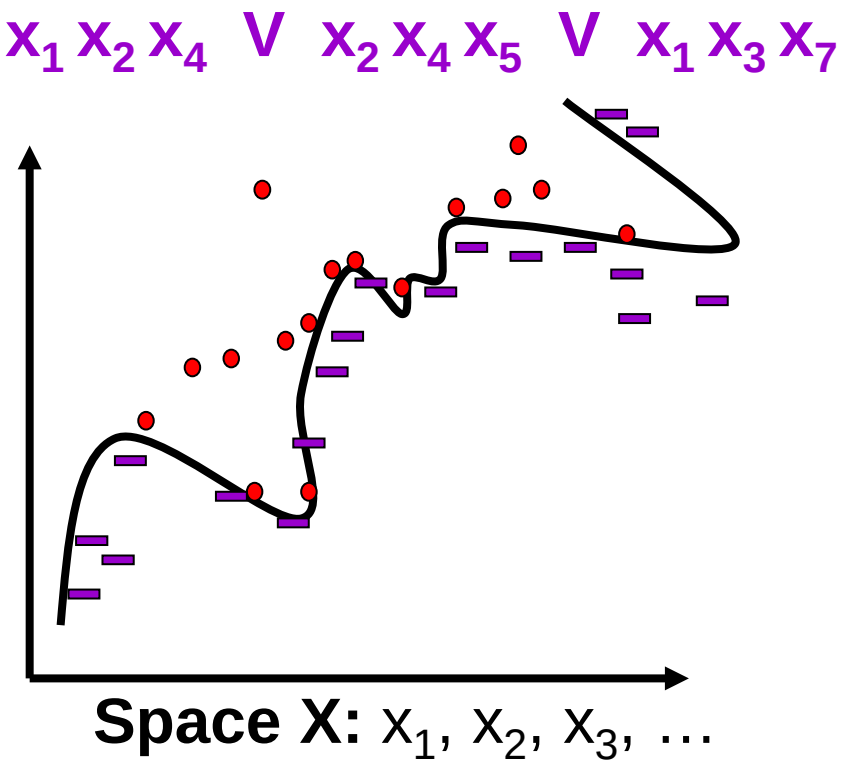


Any linear classifier would introduce errors on the sample

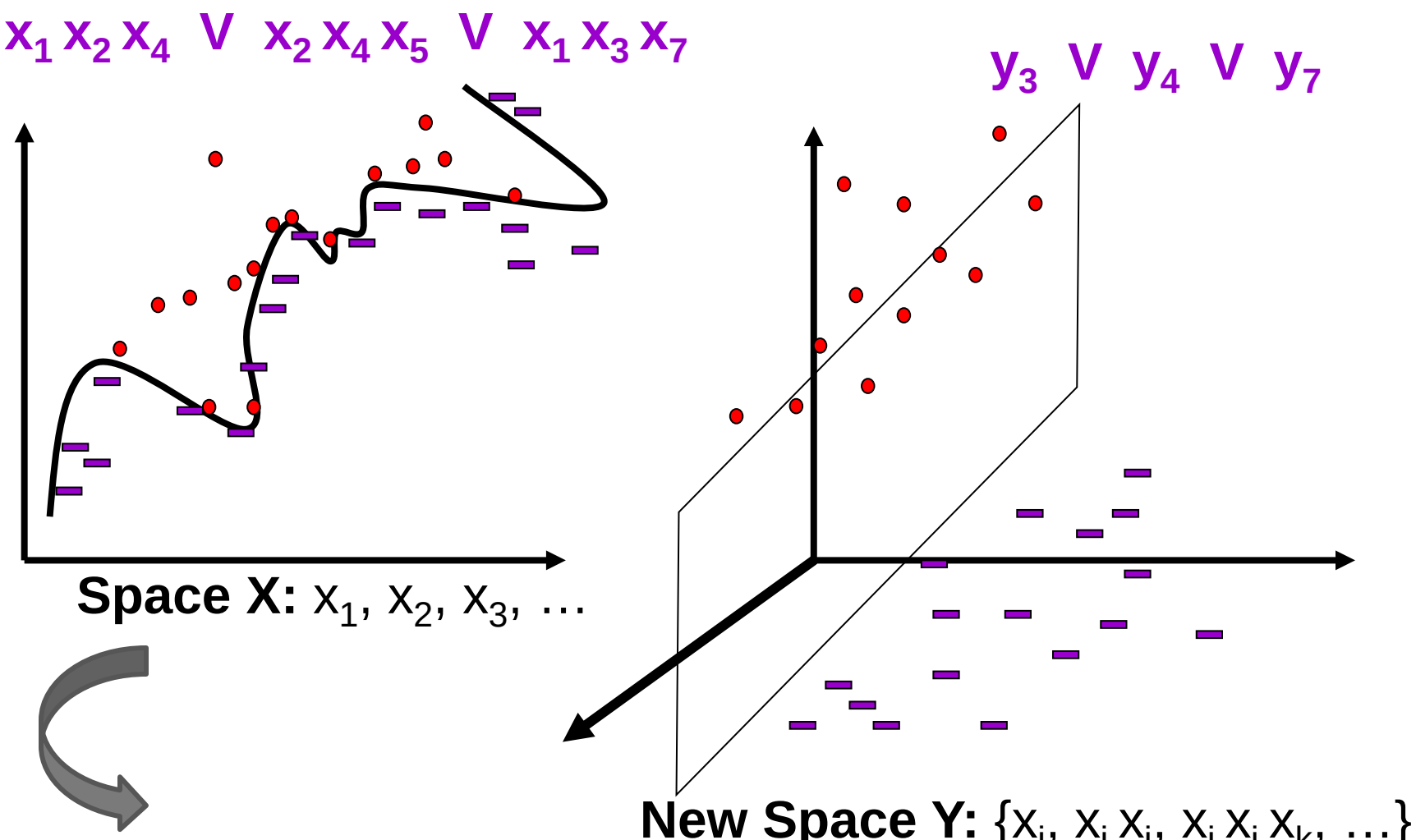
To deal with possible inconsistencies:

- Move to a more expressive feature space
- Learn over a more expressive hypothesis space

More expressive feature space



More expressive feature space



More expressive hypothesis space

$B_n = \{b_1, b_2, \ldots, b_n\}$

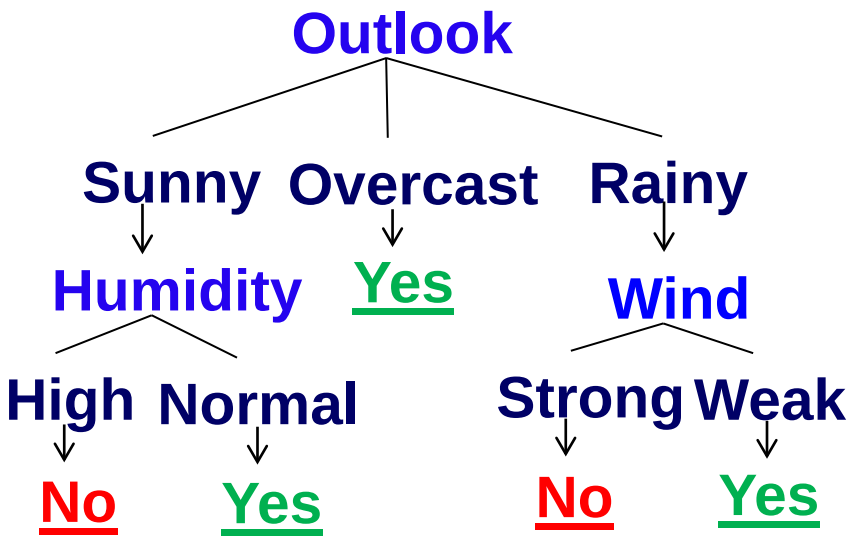
Concept space: All Boolean functions over B_n

Hypothesis space: All Boolean functions over B_n

Note: There are $O(2^{2^n})$ such Boolean formulas

Decision Trees

Hierarchical data that expresses arbitrary Boolean functions



Example:
Decision Tree for "play tennis?"

- Each internal node is labeled with an (discrete) attribute
- Each branch corresponds to a value for the attribute
- Each leaf node is labeled with a classification

Will my friend play tennis today?

Features:

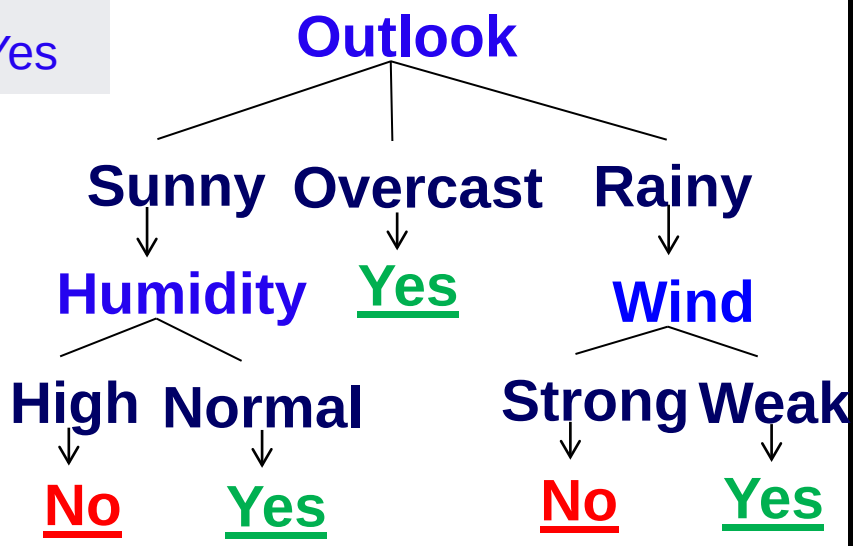
- Outlook: { Sunny, Overcast, Rainy }
- Temperature: { Hot, Mild, Cold }
- Humidity: { High, Normal }
- Wind: { Strong, Weak }

Classification:

- Binary classification (play tennis?): { Yes, No }

Will my friend play tennis today?

Outlook	Temperature	Humidity	Wind	Tennis?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cold	Normal	Weak	Yes
Rainy	Cold	Normal	Strong	No
Overcast	Cold	Normal	Strong	Yes
Sunny	Mild	Normal	Weak	Yes



Learning decision trees: ID3 [Quinlan 1986]

procedure ID3(*Sample* = <*Pos, Neg*>, *Attributes*)
{
Create a *root* node of the tree
if all examples are positive or all negative then
Return a single node tree *root* with label + or -
else
Select a "*good*" decision attribute *A* for *Sample*
Label the *root* of the tree with this attribute

Divide the *Sample* into two sets: $Sample_A$ and $Sample_{\neg A}$

Return tree with *root* node and left tree
ID3($Sample_A$, *Attributes*) and right tree
ID3($Sample_{\neg A}$, *Attributes*)
}

Learning decision trees: ID3 [Quinlan 1986]

- Polynomial-time algorithm
  - No backtracking
  - Linear in the size of the output tree and the sample

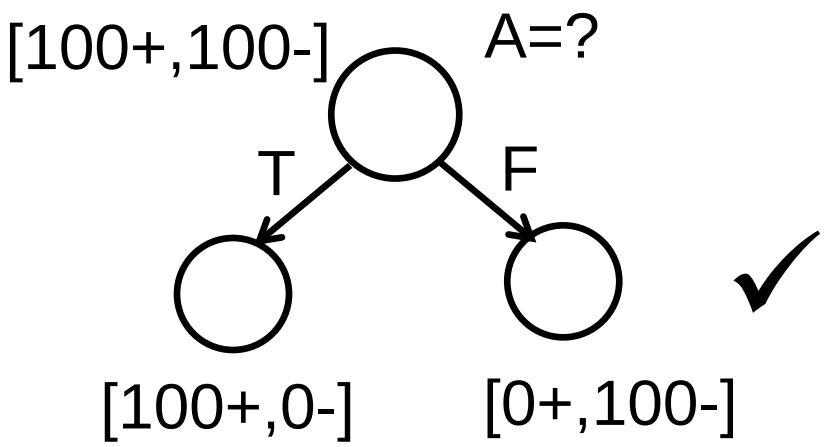
- Pick decision attributes in a “good” way
  - Important since we do not backtrack
  - Inductive bias: smaller trees

Note:

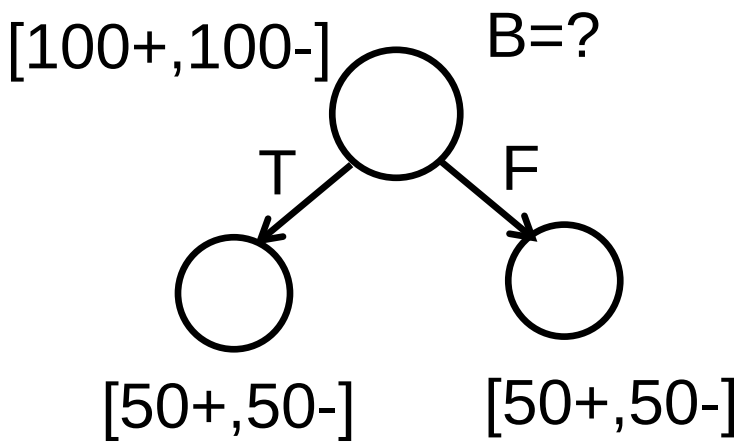
Even though picking good decision attributes is “fuzzy”, the learning algorithm by itself is not “fuzzy”.

- Bad ways of dividing the sample just leads to larger trees

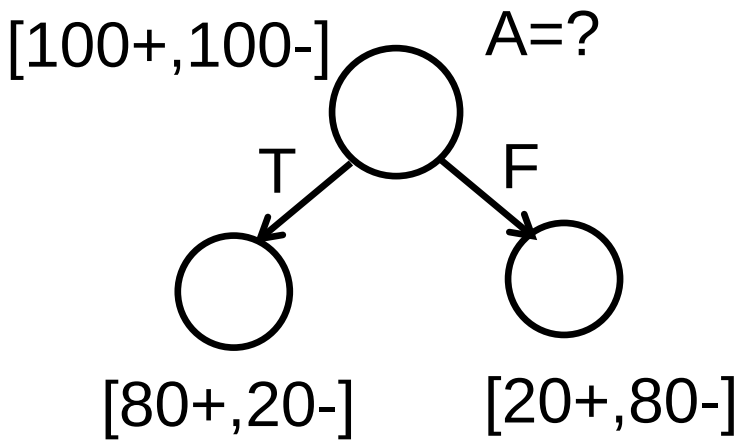
Picking a “good” decision attribute

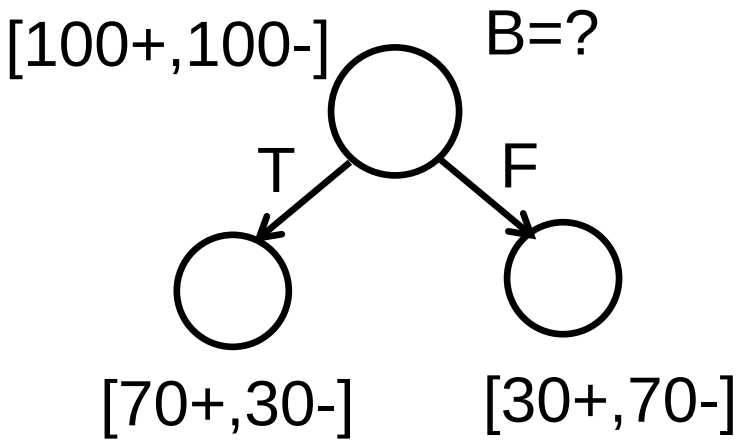


✓

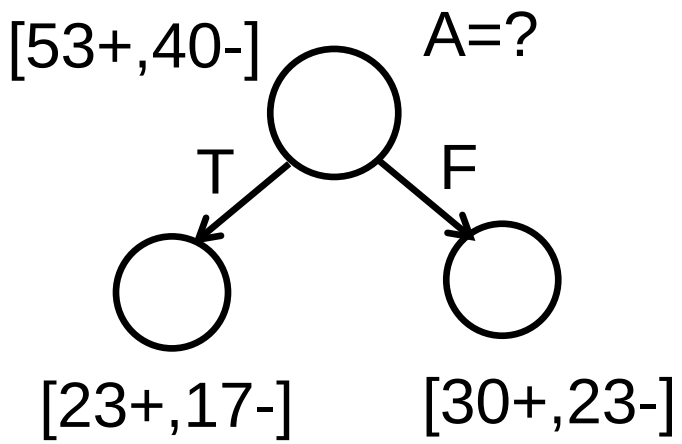


Picking a “good” decision attribute



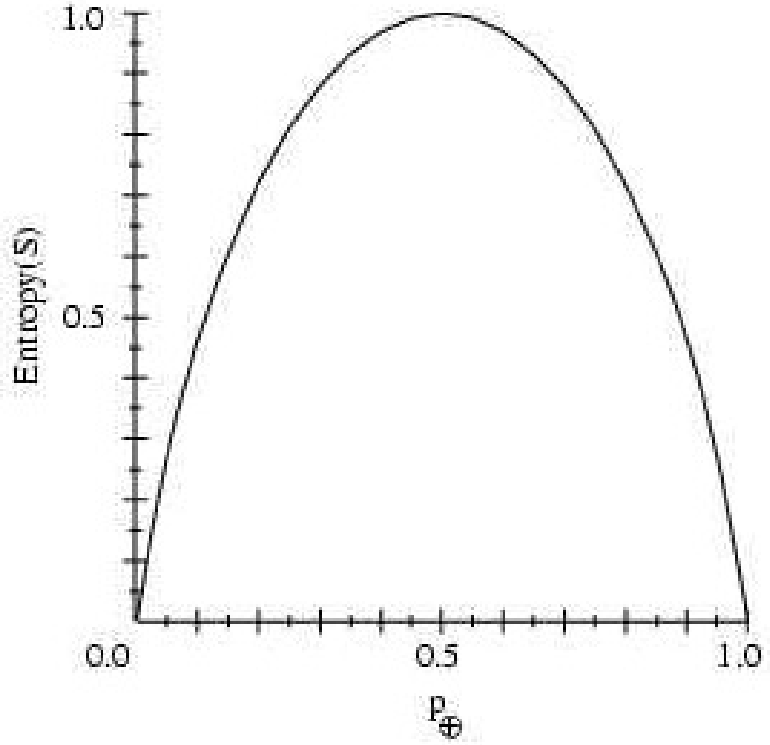


✓



Entropy

- S is a sample of training examples
- $p_{\oplus}/p_{\ominus}$ is the proportion of positive/negative examples in S



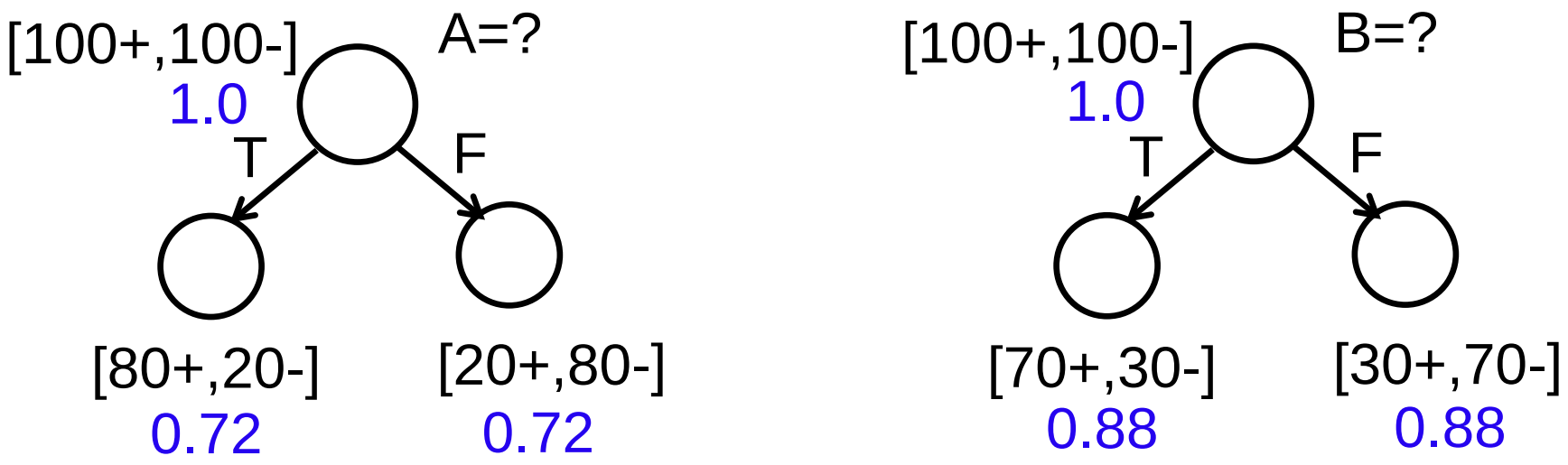
Entropy measures the impurity of S:

$$\textbf{Entropy(S)} = - p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Information Gain

Gain(S, A) is expected reduction in entropy on partitioning the sample S with respect to A

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$



Gain(S, A) = 1.0 – (100/200*0.72 + 100/200*0.72) = **0.28** ✓

Gain(S, B) = 1.0 – (100/200*0.88 + 100/200*0.88) = **0.12**

Demo

Comparing Information gain heuristic with a random splitting criteria

Consistency and Iterative Convergence

Decision tree learned is consistent with the sample

Disable default settings related to:

- Pruning: MDL, …
- not further partitioning a sub-sample with few examples

Convergence

- There are a finite number of Boolean formulas
- Current hypothesis is (semantically) different from all previous hypotheses.
  - Sample contains counterexamples to all previous hypotheses

Decision tree learner is convergent in the iterative setting

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE samples	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	2 Houdini	2 Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	Open	--
Conjunctions(LTF) **SVM**	✓	✓	✓	Open	--
Boolean functions **Decision trees**	✓	✓	✓	2	2
Bool Threshold fns **Decision trees w/ cont attribs**	1	1	2	2	2

Decision Trees over Continuous attributes

$S_n = \{b_1, b_2, \ldots, b_n, v_1, v_2, \ldots, v_m\}$

where, b_i are over {True, False}, v_i are over **Z**

Concept space: All functions from S_n to {True, False}

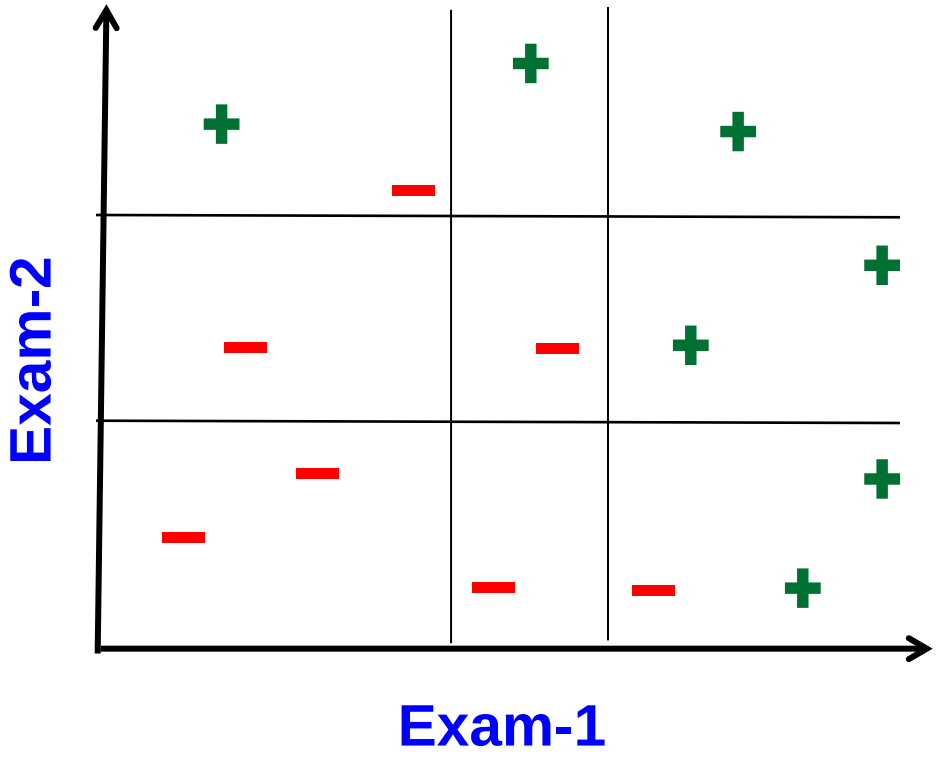
Hypothesis space: All Boolean functions over predicates:

b_i and $v_i \leq c$, for $c \in$ **Z.**

Note: There are infinitely many formulas

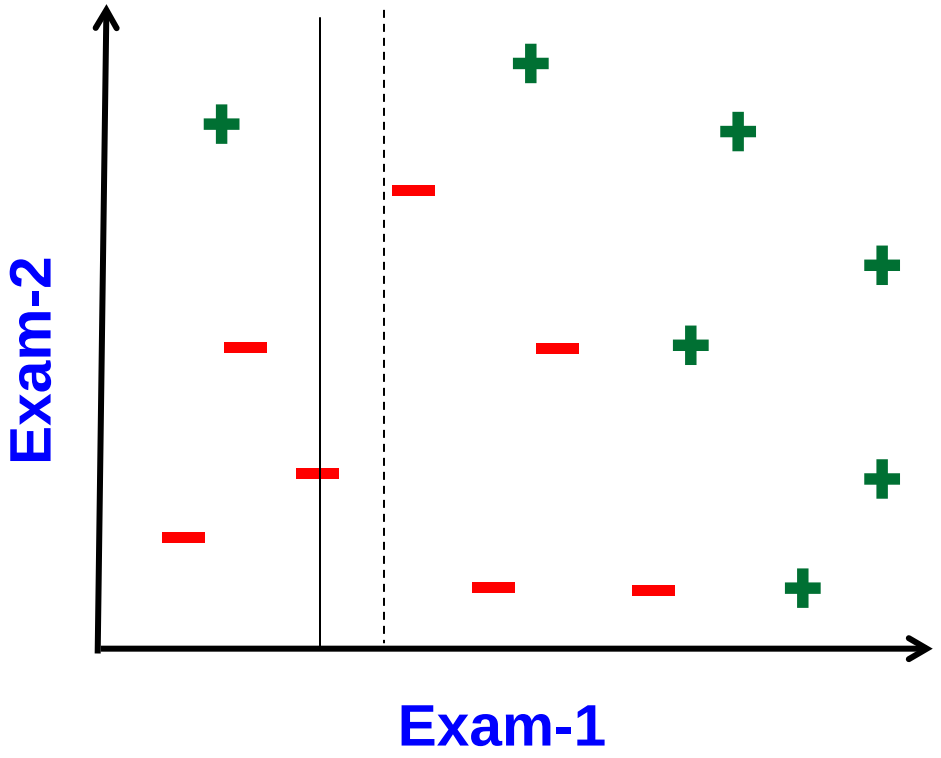
Decision Trees over Continuous attributes

Exam-1	Exam-2	Passed/ Failed?
5	15	No
10	80	Yes
15	45	No
25	25	No
40	68	No
50	10	No
55	90	Yes
60	45	No
70	10	No
75	45	Yes
80	80	Yes
85	10	Yes
95	25	Yes
95	55	Yes



Decision Trees over Continuous attributes

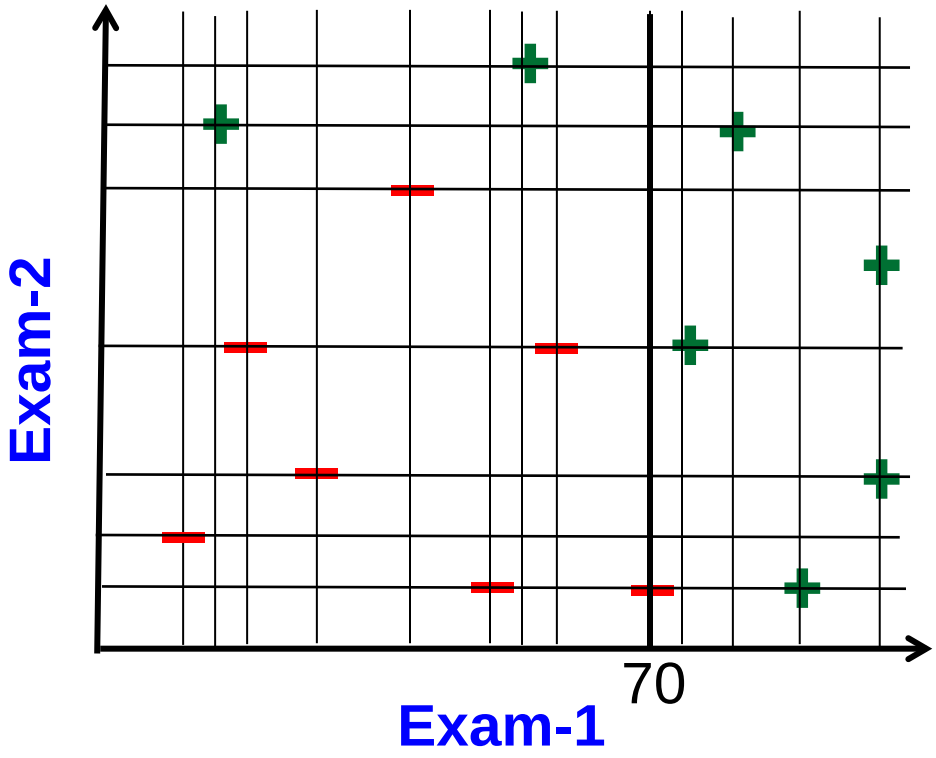
Exam-1	Exam-2	Passed/ Failed?
5	15	No
10	80	Yes
15	45	No
25	25	No
40	68	No
50	10	No
55	90	Yes
60	45	No
70	10	No
75	45	Yes
80	80	Yes
85	10	Yes
95	25	Yes
95	55	Yes



For thresholds c, consider only the values in the sample

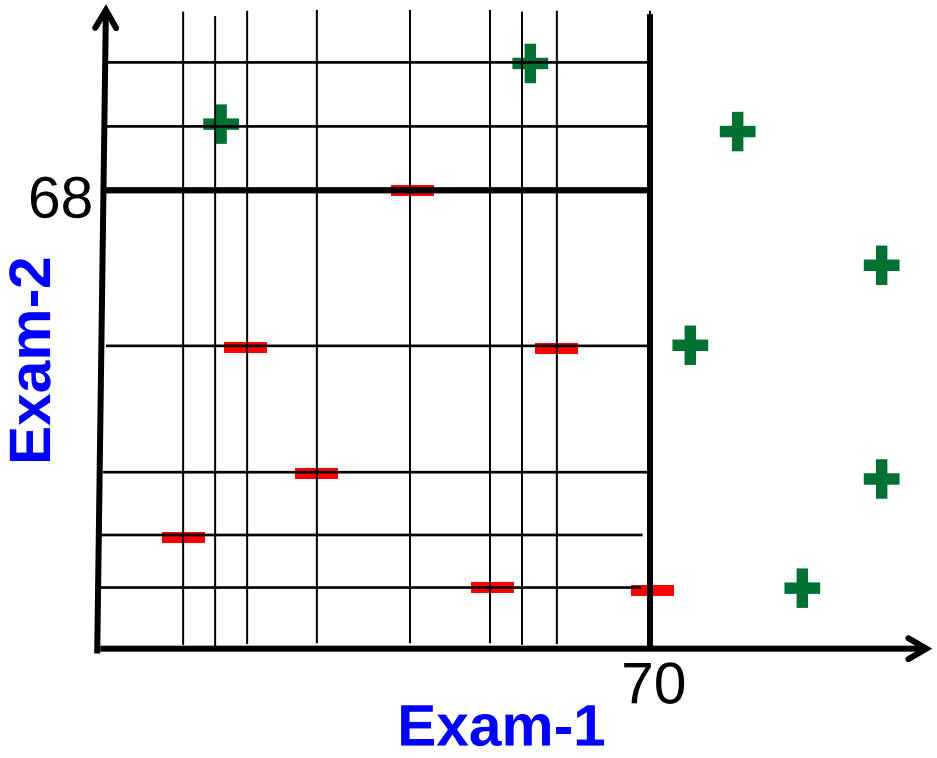
Decision Trees over Continuous attributes

Exam-1	Exam-2	Passed/ Failed?
5	15	No
10	80	Yes
15	45	No
25	25	No
40	68	No
50	10	No
55	90	Yes
60	45	No
70	10	No
75	45	Yes
80	80	Yes
85	10	Yes
95	25	Yes
95	55	Yes

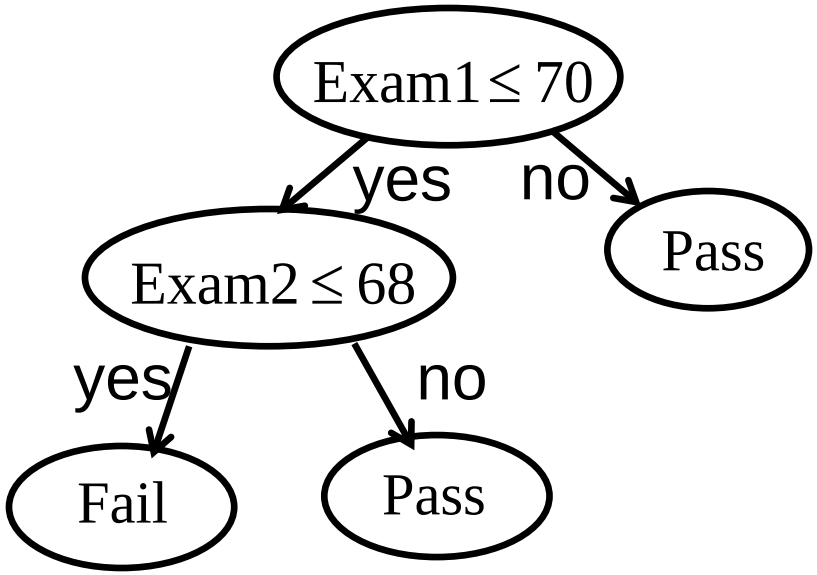


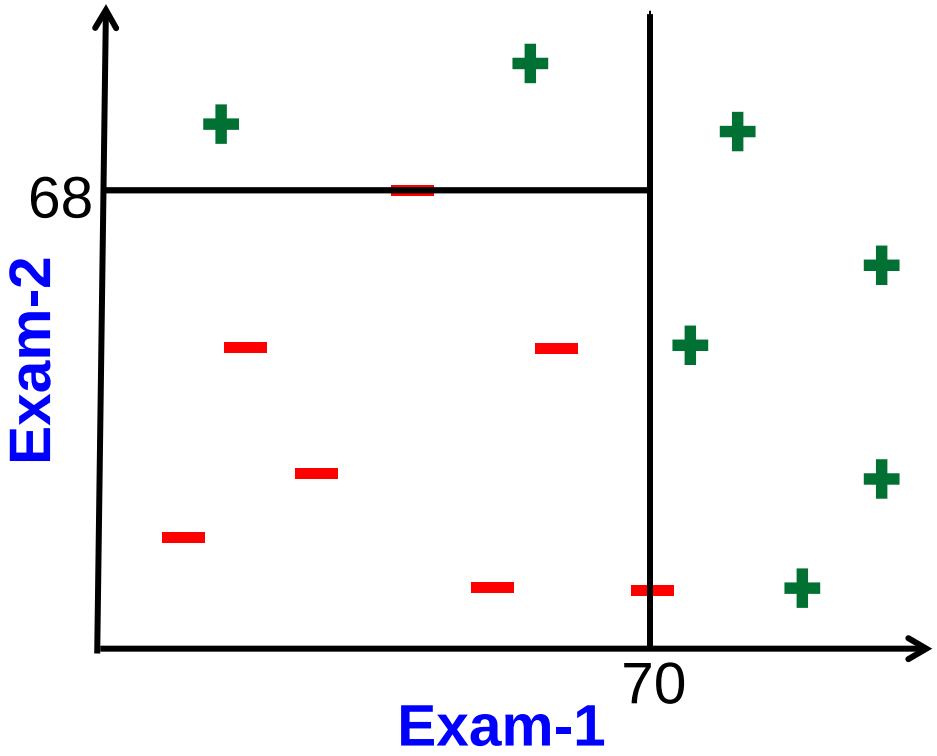
Decision Trees over Continuous attributes

Exam-1	Exam-2	Passed/ Failed?
5	15	No
10	80	Yes
15	45	No
25	25	No
40	68	No
50	10	No
55	90	Yes
60	45	No
70	10	No
75	45	Yes
80	80	Yes
85	10	Yes
95	25	Yes
95	55	Yes



Decision Trees over Continuous attributes





- Decision tree learned is consistent with the sample (no pruning, …)
- Iterative convergence: hard to build (described later)

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE samples	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	2 Houdini	2 Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	Open	--
Conjunctions(LTF) **SVM**	✓	✓	✓	Open	--
Boolean functions **Decision trees**	✓	✓	✓	2	2
Bool Threshold fns **Decision trees**	✓	✓	2	2	2

CHALLENGES TO USING MACHINE LEARNING FOR INVARIANT SYNTHESIS

1. ~~**Removing fuzziness for machine learning algorithms**~~

   ~~**How do we make them learn hypotheses consistent with the sample?**~~

2. ~~**Will they converge, even with (+,-) counterexamples, to any target concept?**~~

3. **How can we adapt them to handle implication counterexamples (ICE)?**

4. **How do we make these ICE learning algorithms converge?**

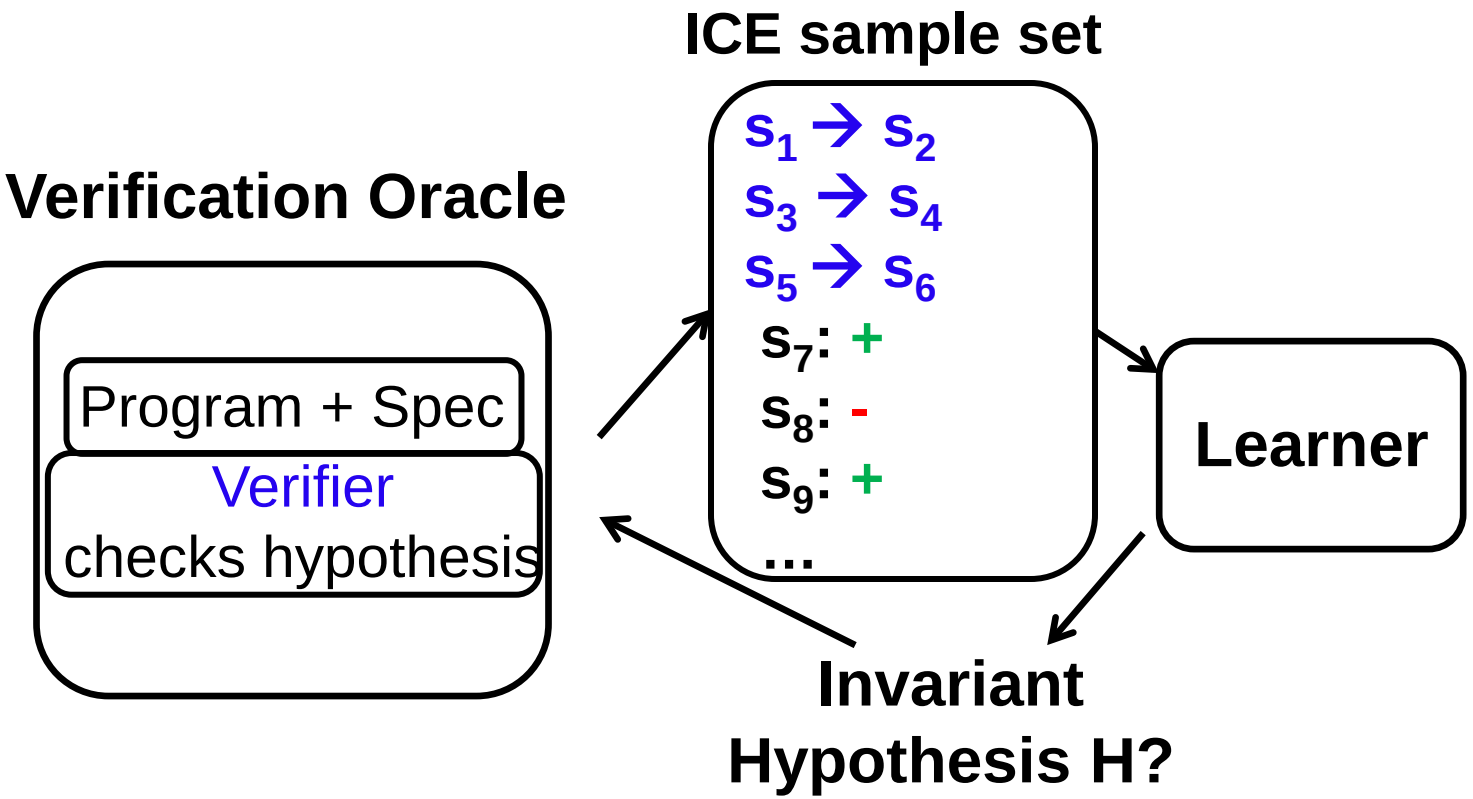
DEMO FOR WEKA

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE samples	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	2 Houdini	2 Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	Open	--
Conjunctions(LTF) **SVM**	✓	✓	✓	Open	--
Boolean functions **Decision trees**	✓	✓	✓	2	2
Bool Threshold fns **Decision trees w/ cont attribs**	✓	✓	2	2	2

ICE: Learning invariants using Implication Counter-Examples

([Garg et al. CAV 14])



Extending Elimination algorithm to ICE

$B_n = \{b_1, b_2, \ldots, b_n\}$

Concept space: All Boolean functions over B_n

Hypothesis space: Conjuncts of atoms over B_n

Main idea:
Return the smallest set that includes all positive points and is consistent with implication counterexamples

Extending Elimination algorithm to ICE

Input: sample { $(p_1, +), \ldots, (p_u, +), (n_1, -), \ldots, (n_v, -),$
$(s_1, t_1), \ldots, (s_w, t_w)$ }

Algorithm:

$\varphi := \bigwedge b_i$ s.t. for no positive example $(p, +)$ is $p(b_i) = F$
while(φ is not consistent with all implications)
{
Pick an implication (s, t) inconsistent with φ , i.e.,
$s \models \varphi$ and $t \not\models \varphi$
Update φ to remove all atoms b_i where $t(b_i) = F$
}

φ excludes the negative points
(provided some conjunction satisfies the sample)
So if it doesn't satisfy $(n, -)$, we can abort.

Convergence in iterative ICE

Initially when the sample is empty, propose

$$b_1 \wedge b_2 \wedge \ldots \wedge b_n$$

In each round, learner will propose the semantically smallest set that contains positive and implication counterexamples

Hence, teacher can never give a negative counterexample

Whenever, teacher returns *(p, +)* or *(s, t)*

"knock off" all conjuncts b_i for which $p(b_i) = F$ or $t(b_i) = F$

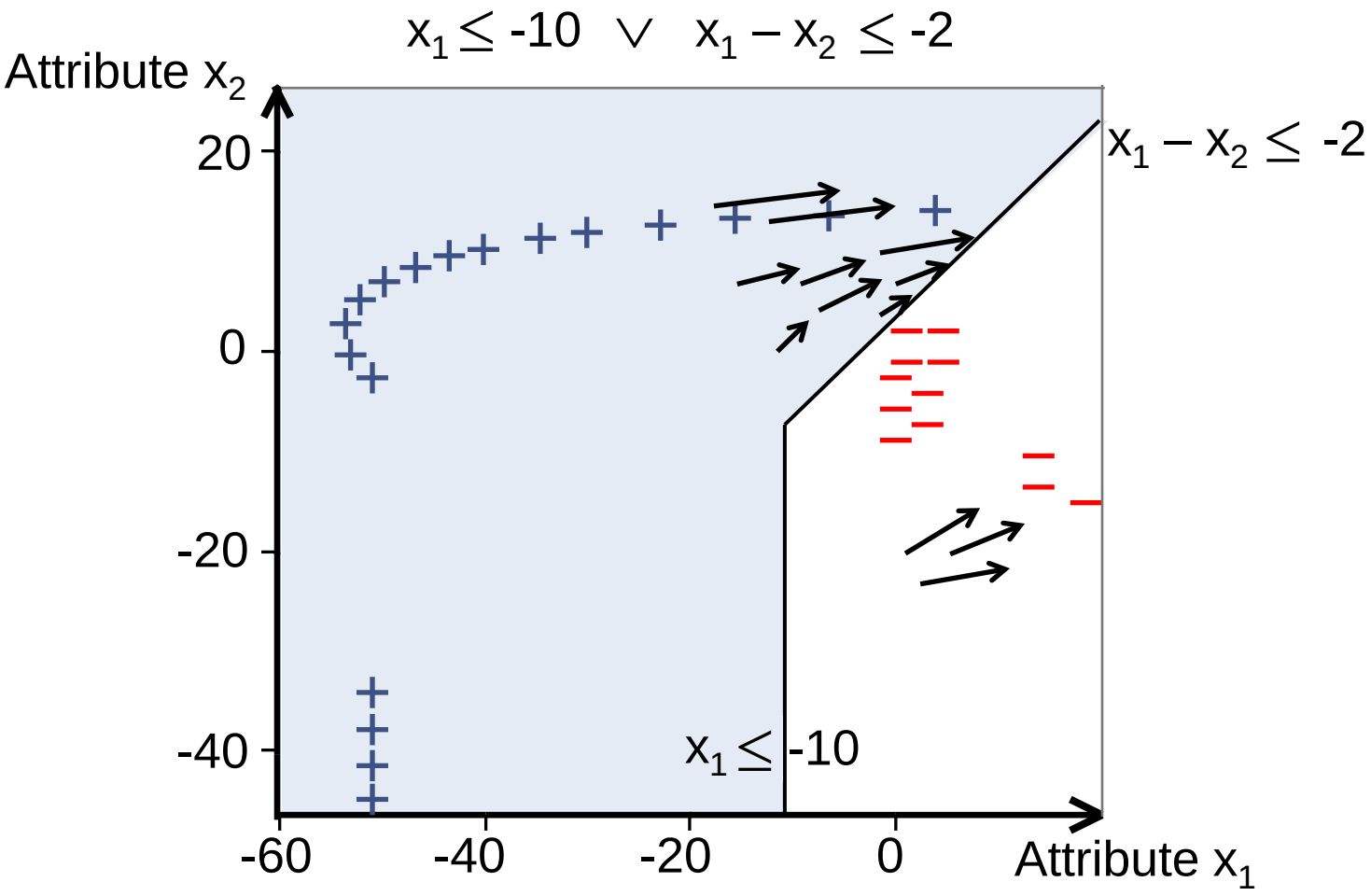
Converges in O(n) rounds!

(is precisely the **Houdini** algorithm that learns invariants which are conjunctions over atomic predicates)

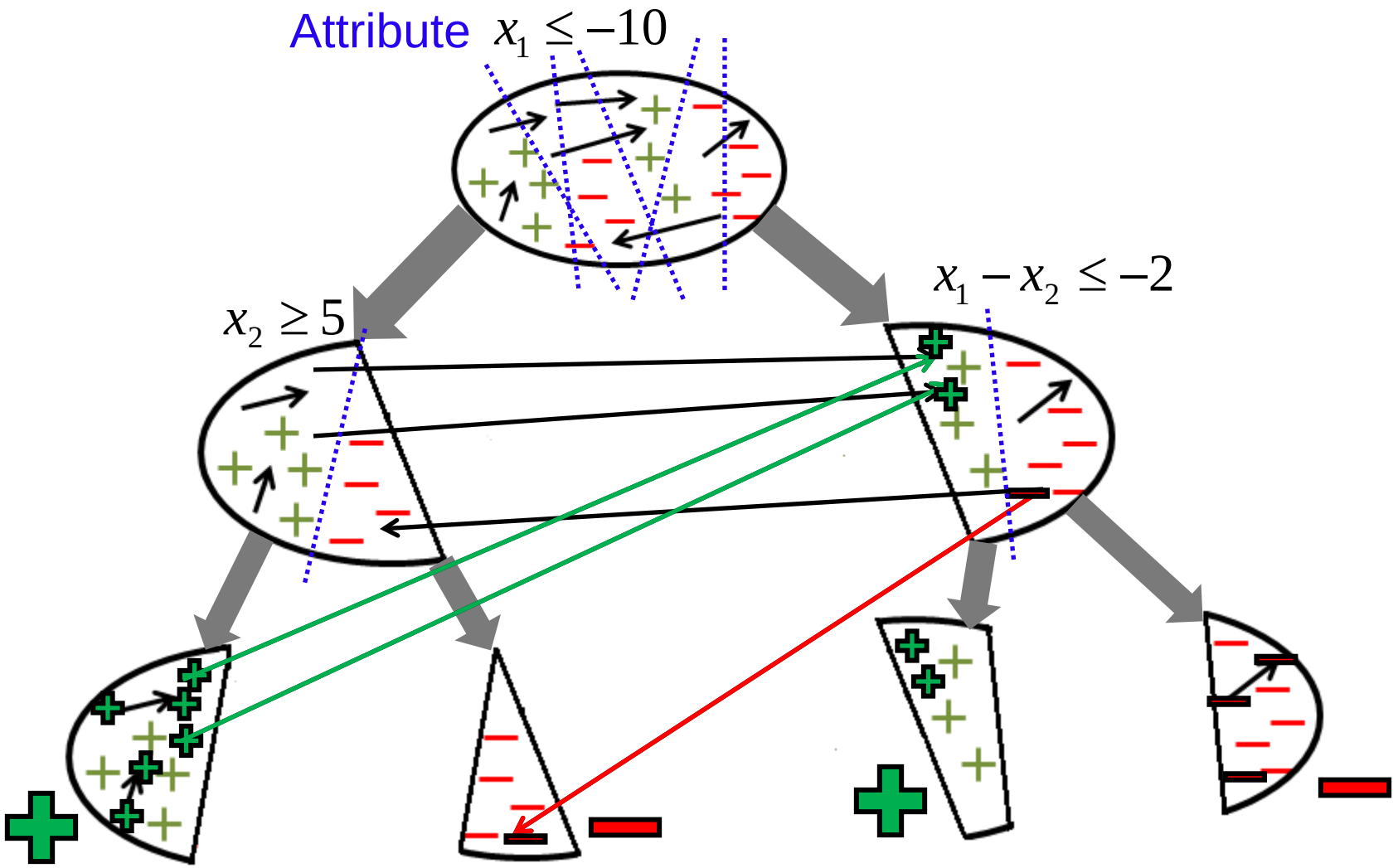
Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE samples	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	✓ Houdini	✓ Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	**Open**	--
Conjunctions(LTF) **SVM**	✓	✓	✓	**Open**	--
Boolean functions **Decision trees**	✓	✓	✓	**2A**	**2A**
Bool Threshold fns **Decision trees**	✓	✓	**2A**	**2A**	**2A**

Adapting decision tree learning to ICE



Learning decision trees given an ICE sample



[Garg et al. UIUC-TR-2015]

Theorem:

The decision-tree based learning algorithm always produces a decision tree that is consistent with the input ICE sample.

Valid Sample:

- No point is classified as ✚ and ━

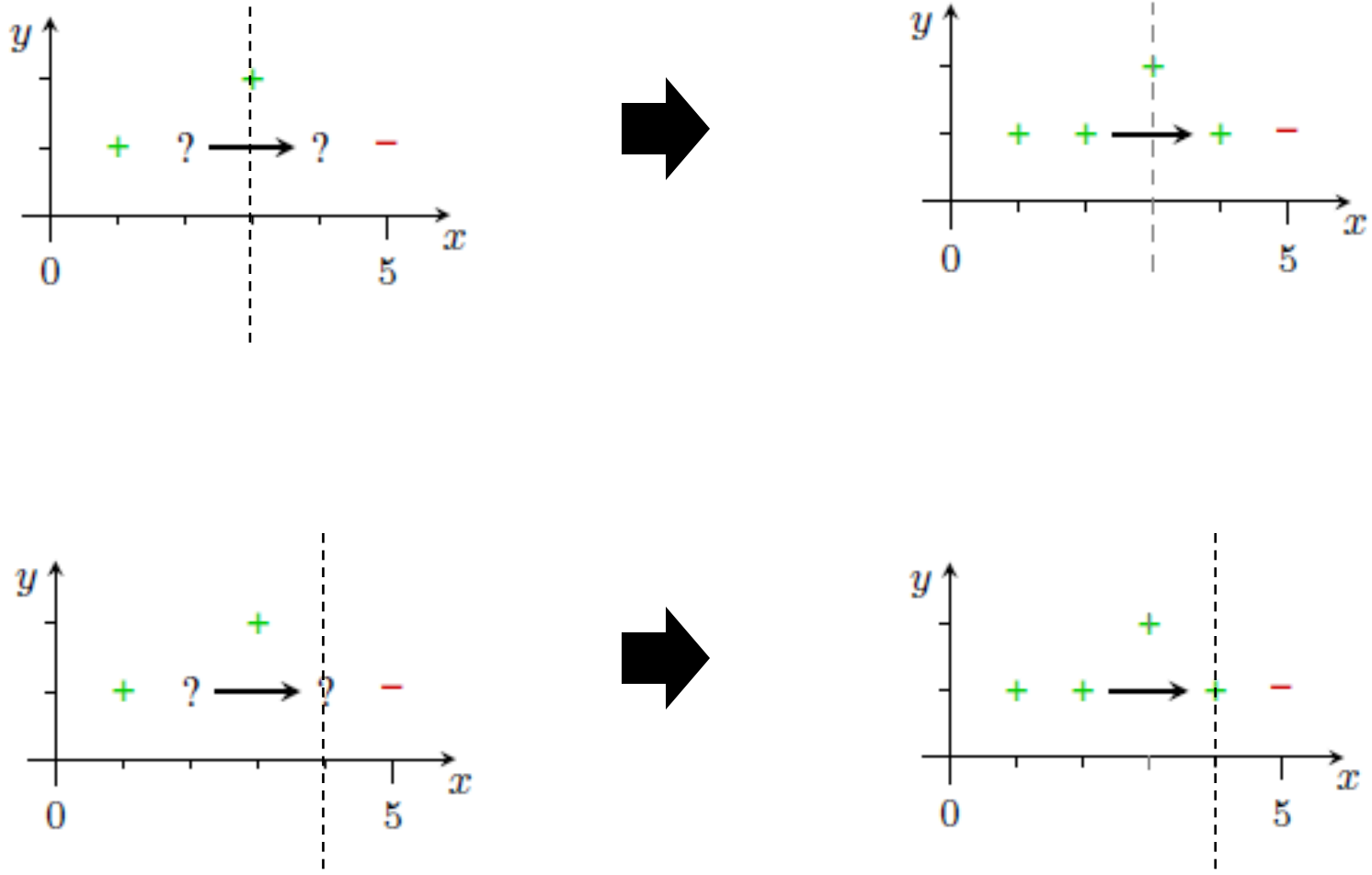
- For an implication, s1 ⟶ s2 :

  label(s1) = ✚ ➡ label(s2) = ✚

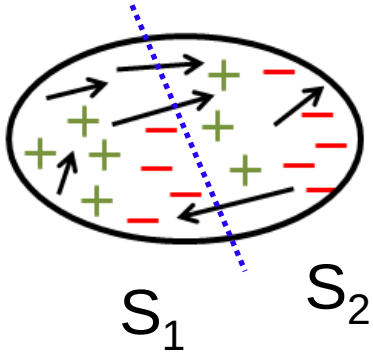
  label(s2) = ━ ➡ label(s1) = ━

Lemma: In a valid sample if, we label a subset of implication endpoints as purely positive (or purely negative), it results in classification that is consistent with the ICE sample.

Ignoring implications while picking the decision attribute is not a good idea



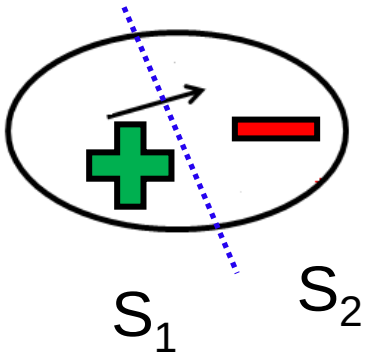
Picking a “good” decision attribute



Measure 1: Penalize implications “cut” by attribute

$$InfoGain_{ICE} = InfoGain_{classical} - Penalty(S_1, S_2, Implications)$$

Picking a “good” decision attribute



Measure 1: Penalize implications “cut” by attribute

$$InfoGain_{ICE} = InfoGain_{classical} - Penalty(S_1, S_2, Implications)$$

$$Penalty\ (S_1, S_2, Implicatio\ ns) = \frac{|S_1^{Pos}|}{|S_1|}.\frac{|S_2^{Neg}|}{|S_2|}.|I_{S_1 \to S_2}| + \frac{|S_2^{Pos}|}{|S_2|}.\frac{|S_1^{Neg}|}{|S_1|}.|I_{S_2 \to S_1}|$$

Picking a "good" decision attribute

Measure 2: Extends Shannon entropy and Information Gain to implications using probabilistic dependencies

$$\mathbf{InfoGain}_{\mathbf{classical}} = \mathbf{Entropy(S)} - \left(\frac{|\mathbf{S}_1|}{|\mathbf{S}|}\mathbf{Entropy}(\mathbf{S}_1) + \frac{|\mathbf{S}_2|}{|\mathbf{S}|}\mathbf{Entropy}(\mathbf{S}_2) \right)$$

$$\mathbf{Entropy(S)} = -\mathbf{P(S,+)}\log_2\mathbf{P(S,+)} - \mathbf{P(S,-)}\log_2\mathbf{P(S,-)}$$

$$\mathbf{P(S,+)} = \frac{1}{|\mathbf{S}|}\left(\sum_{\mathbf{x}\in\mathbf{S}} \mathbf{P(x=+)} \right)$$

$$= \frac{1}{|\mathbf{S}|}\left(\sum_{\mathbf{x}\in\mathbf{Pos}} \underbrace{\mathbf{P(x=+)}}_{\mathbf{1}} + \sum_{\mathbf{x}\in\mathbf{Neg}} \underbrace{\mathbf{P(x=+)}}_{\mathbf{0}} + \left(\sum_{(\mathbf{x}_1,\mathbf{x}_2)\in\mathbf{Pos}} \underbrace{\mathbf{P(x_1=+)}}_{\mathbf{P(S,+)}} + \underbrace{\mathbf{P(x_2=+)}}_{\text{expressed as a conditional probability}} \right) \right)$$

Picking a "good" decision attribute

Measure 2: Extends Shannon entropy and Information Gain to implications using probabilistic dependencies

$$\mathbf{InfoGain}_{\mathbf{classical}} = \mathbf{Entropy(S)} - \left(\frac{|\mathbf{S_1}|}{|\mathbf{S}|}\mathbf{Entropy(S_1)} + \frac{|\mathbf{S_2}|}{|\mathbf{S}|}\mathbf{Entropy(S_2)} \right)$$

$$\mathbf{Entropy(S)} = -\mathbf{P(S,+)}\,\mathbf{log_2 P(S,+)} \; - \mathbf{P(S,-)}\,\mathbf{log_2 P(S,-)}$$

$\mathbf{P(S,+)}$ is the root of the following quadratic equation over **t**:

$$\mathbf{it^2 + (p + n - i)t - p = 0}$$

where: **p, n, i** is the number of positive points, negative points and implications in S

Meeting the challenges

	Classical Machine Learning	Ensuring cosistent learning	Iterative (+,-) convergence	Passive learning ICE samples	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	✓ Houdini	✓ Houdini
Conjunctions(LTF) **Winnow**	1A	1A	1A	Open	--
Conjunctions(LTF) **SVM**	✓	✓	✓	Open	--
Boolean functions **Decision trees**	✓	✓	✓	✓	✓
Bool Threshold fns **Decision trees w/ cont attribs**	✓	✓	2A	✓	2A

Building a convergent decision tree learner for Positive/Negative samples

Recall:

Hypothesis space:
All Boolean functions over predicates:

$$b_i \text{ and } v_i \leq c, \text{ for } c \in \mathbf{Z},$$

where b_i are over {True, False} and v_i are over $\mathbf{Z}$.

Note: There are infinitely many formulas

Building a convergent decision tree learner for Positive/Negative samples

Restricting thresholds c $\in$ Z to values such that | c | < m

Boolean functions over predicates

$$b_i \text{ and } v_i \leq c \text{ for } |c| < m,\ c \in \mathbf{Z},\ m \in \mathbf{Z^+}$$

is finite.

Main loop:

If possible, then construct a consistent decision tree with thresholds at most m.

Otherwise, increment m.

Building a convergent decision tree learner for Positive/Negative samples

ID3$_{<m}$: decision tree learning algorithm where only thresholds at most **m** are considered.

- If **ID3$_{<m}$** constructs a tree → has thresholds at most m.
- If **ID3$_{<m}$** is unable to construct a tree → there is no consistent tree with thresholds at most m

Biases the learner to learn decision trees with smaller constants

Building a convergent decision tree learner for Positive/Negative samples

Algorithm:

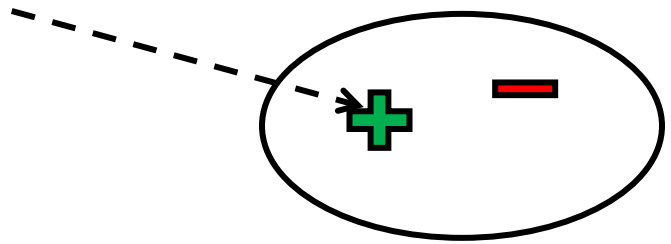
```
m := 1;
while (true)
{
        if (t := ID3(sample, attributes, m))
                return t;
        else
                m := m + 1;
}
```

Theorem:

If there is a target function **f**, which is expressible as a Boolean combination of attributes with arbitrary inequalities, then the above decision-tree based learning algorithm will always eventually learn **f**.

Building a convergent decision tree learner for ICE

The above algorithm does not work for ICE!



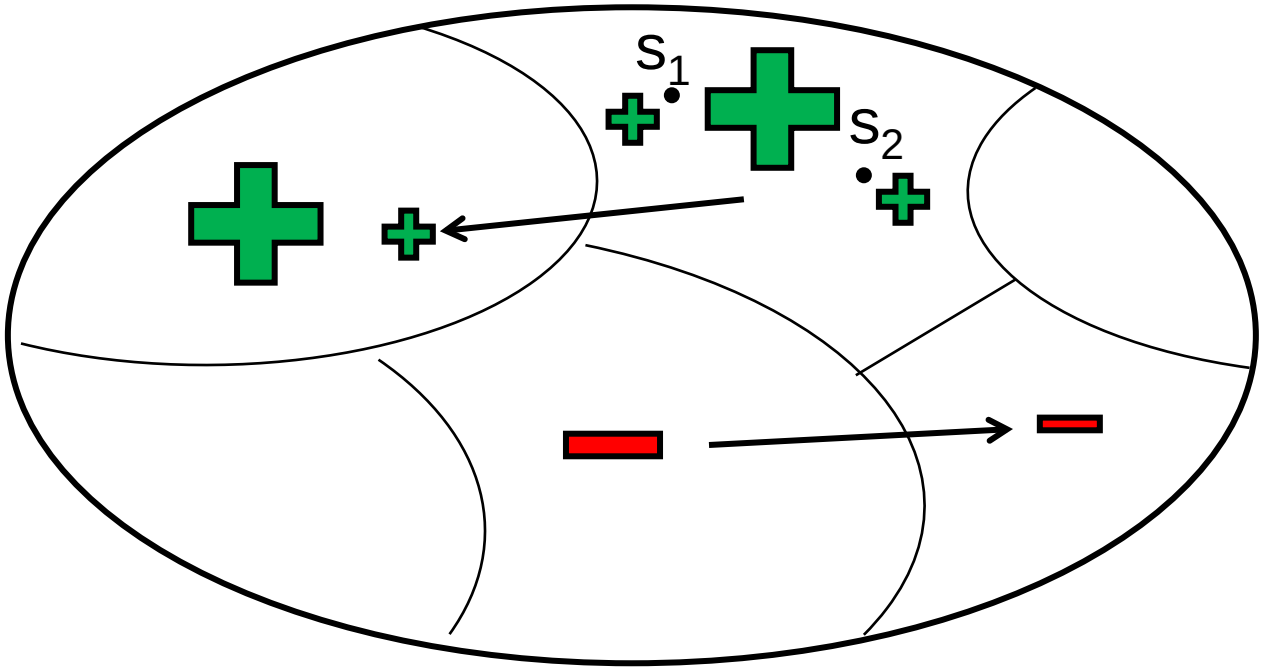
⊕ and ⊖ examples cannot be separated by any predicate with thresholds at most m.

- Might be implication end-points labeled as ⊕ / ⊖ by the learning algorithm
- Back-tracking is hard to control!

Building a convergent decision tree learner for ICE

Equivalence relation $\sim_m$:

$s_1 \sim_m s_2$ if there is no predicate with thresholds at most m that separate s_1 and s_2.



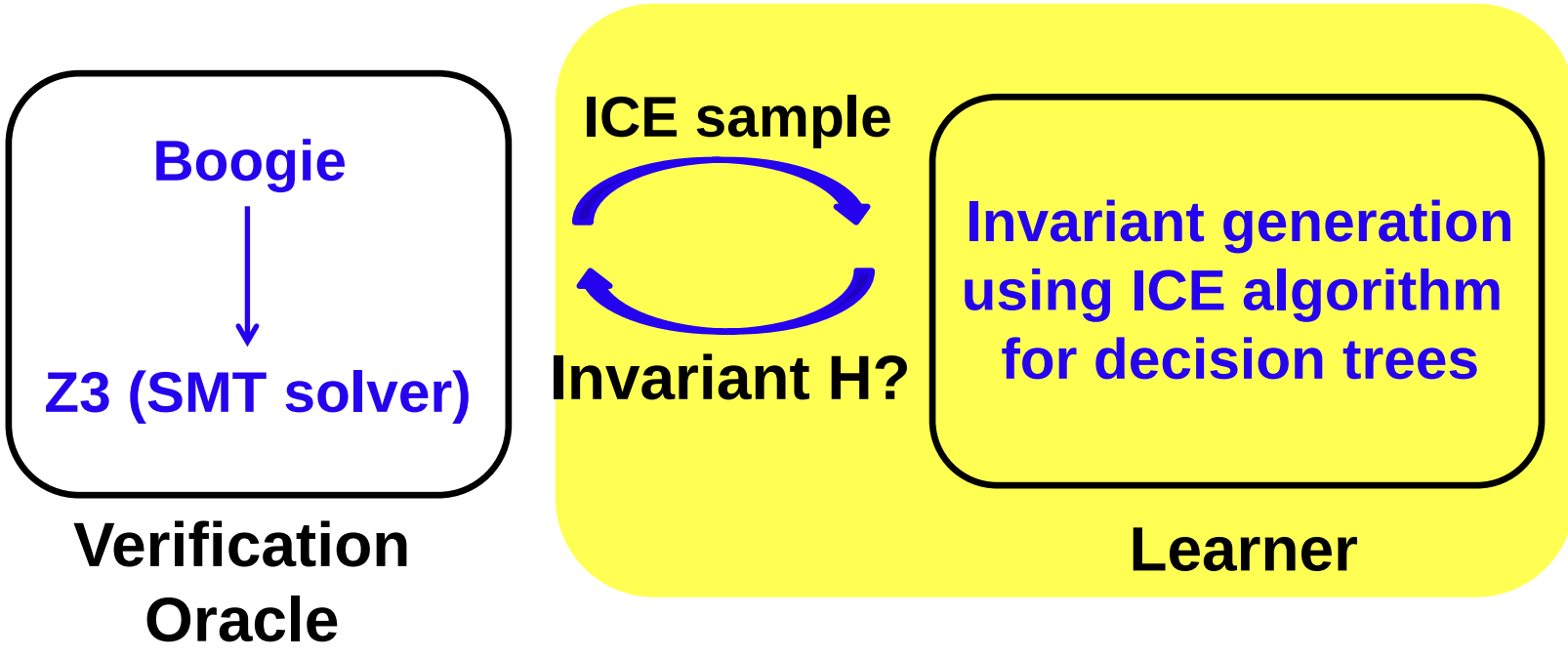
Building a convergent decision tree learner for ICE

Theorem:
The above decision tree ICE learning algorithm converges to an invariant, if an invariant expressible as a Boolean combination of attributes with arbitrary inequalities exists.

Meeting the challenges

	Classical Machine Learning	Ensuring cosistent learning	Iterative (+,-) convergence	Passive learning ICE samples	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	✓ Houdini	✓ Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	**Open**	--
Conjunctions(LTF) **SVM**	✓	✓	✓	**Open**	--
Boolean functions **Decision trees**	✓	✓	✓	✓	✓
Bool Threshold fns **Decision trees w/ cont attribs**	✓	✓	✓	✓	✓

ICE-DT: System Architecture



Attributes for the learner:

- Octagonal attributes, $\pm x \pm y$ for every integer variable x, y
- Can manually specify additional attributes, such as non-linear terms, eg., $x^2 + y^2$

Invariant synthesis using ICE-DT

Invariant synthesis for:

- More than 50 programs from various sources:
  - Up to 100 LOC
  - 5 - 15 variables
  - 50 - 200 attributes

- Won the SyGus competition on the invariant-synthesis track (vs CVC4, enumeration, stochastic search, sketch, …)

Some Invariants Learned
$(2a > t-2) \wedge ((t \leq (a+1)^2 - 1 \wedge 2a \leq t-1 \wedge 2a > 0 \wedge n > 0 \wedge s = (a+1)^2) \vee (t \geq (a+1)^2 \wedge s > 0 \wedge s \leq 1))$
$(a[0][0] \leq m) \vee (m < a[0][0] \wedge j \geq k \wedge j \geq 0)$
$y \leq n \wedge n \leq x + y \wedge n \geq x + y$
$(s^2 + y^2 \leq 0) \vee (s^2 + y^2 > 0 \wedge s \leq j.y \wedge s \geq j.y \wedge j \leq x)$
$(0 \leq y \wedge y \leq z) \wedge (z \leq c + 4572)$
$(N \leq 0) \vee (0 \leq x \wedge 0 \leq m \wedge m \leq N - 1)$
$(m \leq y - 1 \wedge y \leq x \wedge x \leq n) \vee (y \leq m)$

Invariants involve:

- Arbitrary Boolean functions (up to 50 atomic predicates)
- Large constants
- Non-linear terms

Results

Program	ICE-CS [CAV-14]	ICE-DT [UIUC-TR-15]
afnp	3.6	0.5
multiply	✘	59.6
square	✘	0.4
arrayinv1	✘	13.1
inc	1.7	3.9
loops	0.2	0.5
dillig12	✘	3.4
dillig28	0.2	0.5
dtuc	0.7	0.8
● ● ●		
Aggregate	50/58 programs	58/58 programs

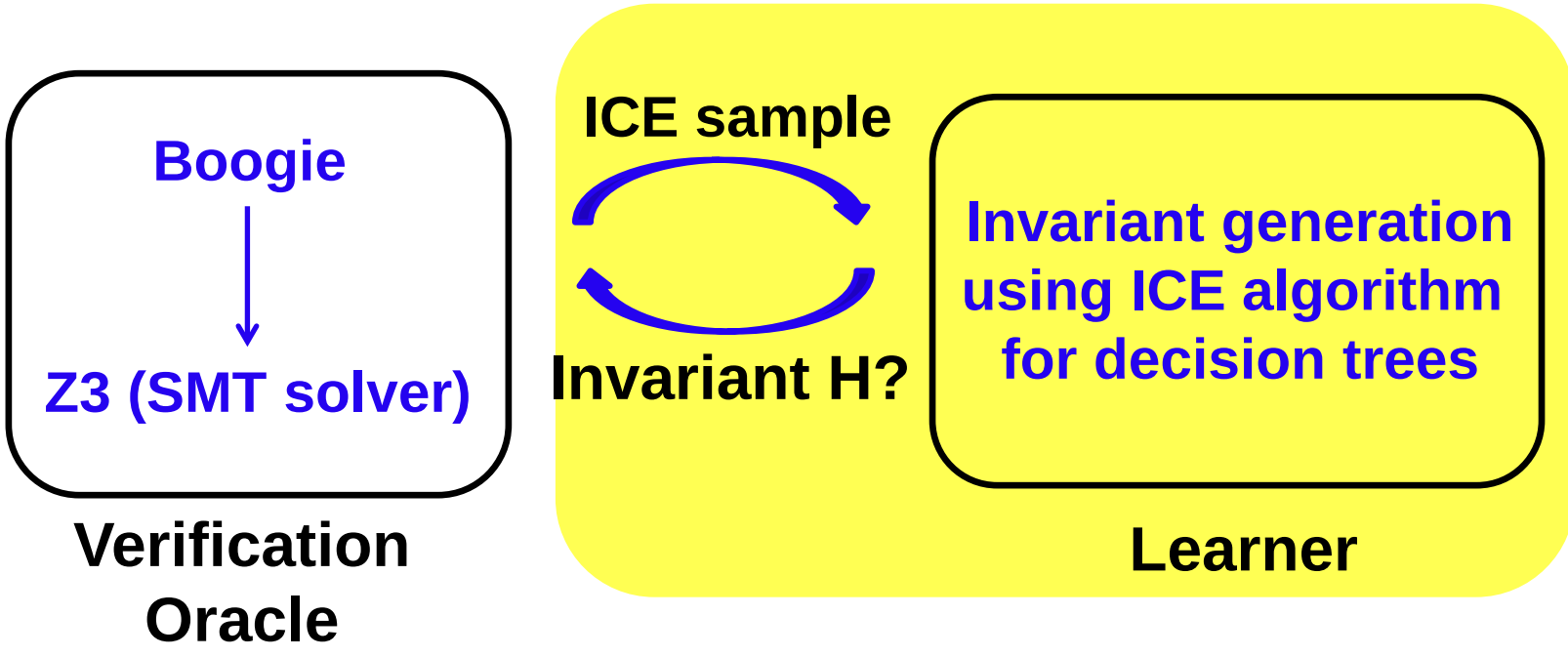
Comparison to white-box invariant synthesis

Invariant Synthesis using interpolation [CPAChecker]	ICE Decision Tree Learner
41/58 programs	**55/58 programs**

- CPAchecker does not restrict itself to invariants over octagonal attributes

- White-box invariant synthesis is hard:
  - Invariants over non-linear terms
  - Synthesizing invariants in partially annotated programs (eg., Houdini)

ICE-DT Demo



Learner is available as a separate module and can be combined with other verification engines.

CHALLENGES TO USING MACHINE LEARNING FOR INVARIANT SYNTHESIS

1. ~~**Removing fuzziness for machine learning algorithms**~~

   ~~**How do we make them learn hypotheses consistent with the sample?**~~

2. ~~**Will they converge, even with (+,-) counterexamples, to any target concept?**~~

3. ~~**How can we adapt them to handle implication counterexamples (ICE)?**~~

4. ~~**How do we make these ICE learning algorithms converge?**~~

Over to Madhu …

Meeting the challenges

	Classical Machine Learning	Ensuring consistent learning	Iterative (+,-) convergence	Passive learning ICE	Iterative ICE convergence
Conjunctions **Elimination Alg**	✓	✓	✓	✓ Houdini	✓ Houdini
Conjunctions(LTF) **Winnow**	✓	✓	✓	**Open**	--
Conjunctions(LTF) **SVM**	✓	✓	✓	**Open**	--
Boolean functions **Decision trees**	✓	✓	✓	✓	✓
Bool Threshold fns **Decision trees w/ cont attribs**	✓	✓	✓	✓	✓

SOME OPEN PROBLEMS

- Winnow for ICE, with linear in r and sublinear in n convergence

- SVM for ICE

- We have only scratched the surface:

  Boolean + Thresholds ($a_i \leq c$)

  Can we derive Boolean combinations of linear threshold functions?

  E.g., $(3x + 4y \leq 9 \wedge 9y \leq 234) \vee (4y \leq x)$

SOME OPEN PROBLEMS

- Black-box learning of invariants in more complex settings:
  - Quantified array invariants
    - Array property fragment / e-matching / Strand
  - Heap invariants
    - Separation logic / decidable fragments / natural proofs
  - Applications:
    - Verifying GPU kernels for race freedom (GPUVerify)
    - Memory safety

FROM INVARIANT SYNTHESIS TO EXPRESSION SYNTHESIS

In program synthesis, inductive synthesis is the *norm!*

Expression synthesis:

Find expression f such that $\forall \bar{x}. \psi(f, \bar{x})$

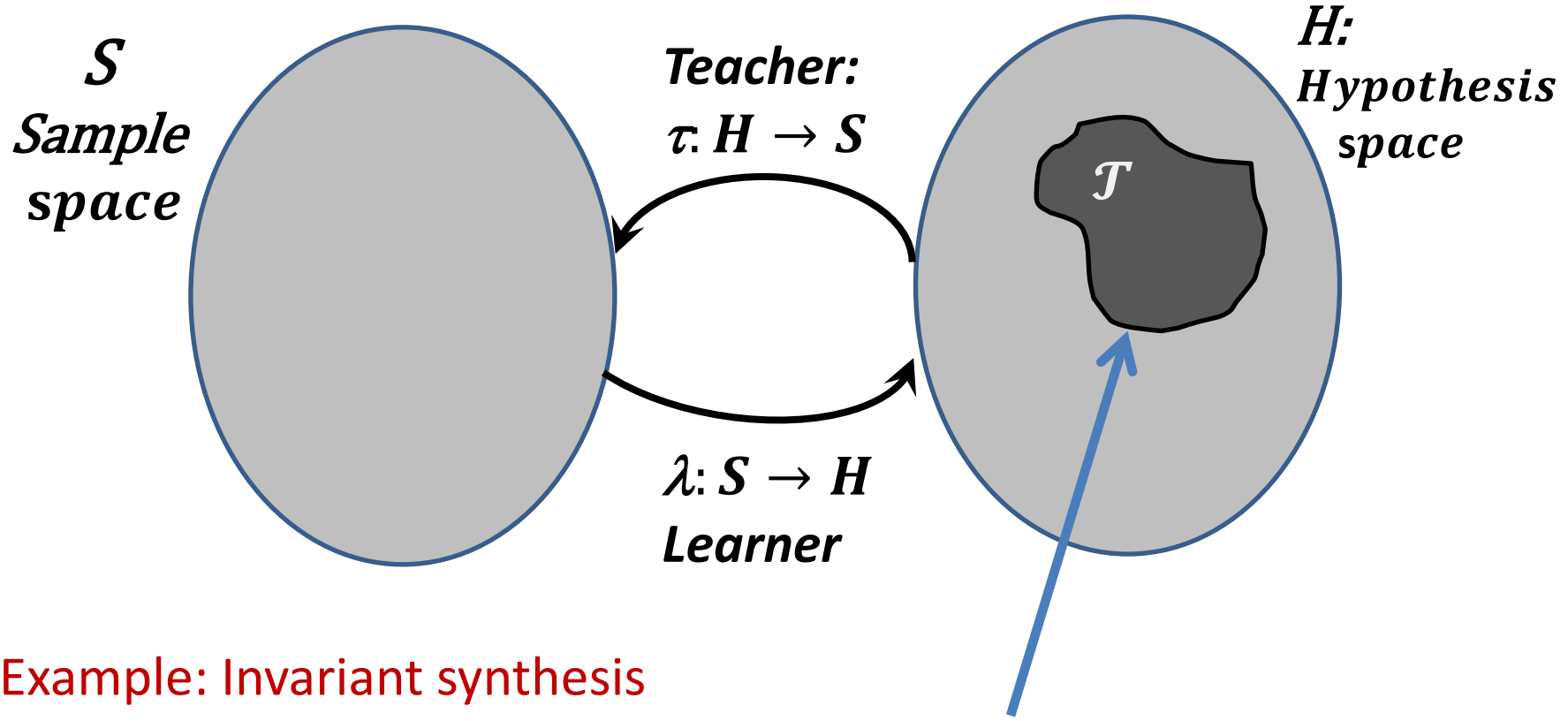
CEGIS loop:
while (true) {

- Propose an f that works every val in S $\Lambda_{v \in S}\ \psi(f, \bar{s})$
- Oracle checks whether $\forall \bar{x}. \psi(f, \bar{x})$ holds.

  If yes, done, HALT.

  If not, finds a new counterexample valuation for $\bar{x}$.
- Add this counterexample to S.

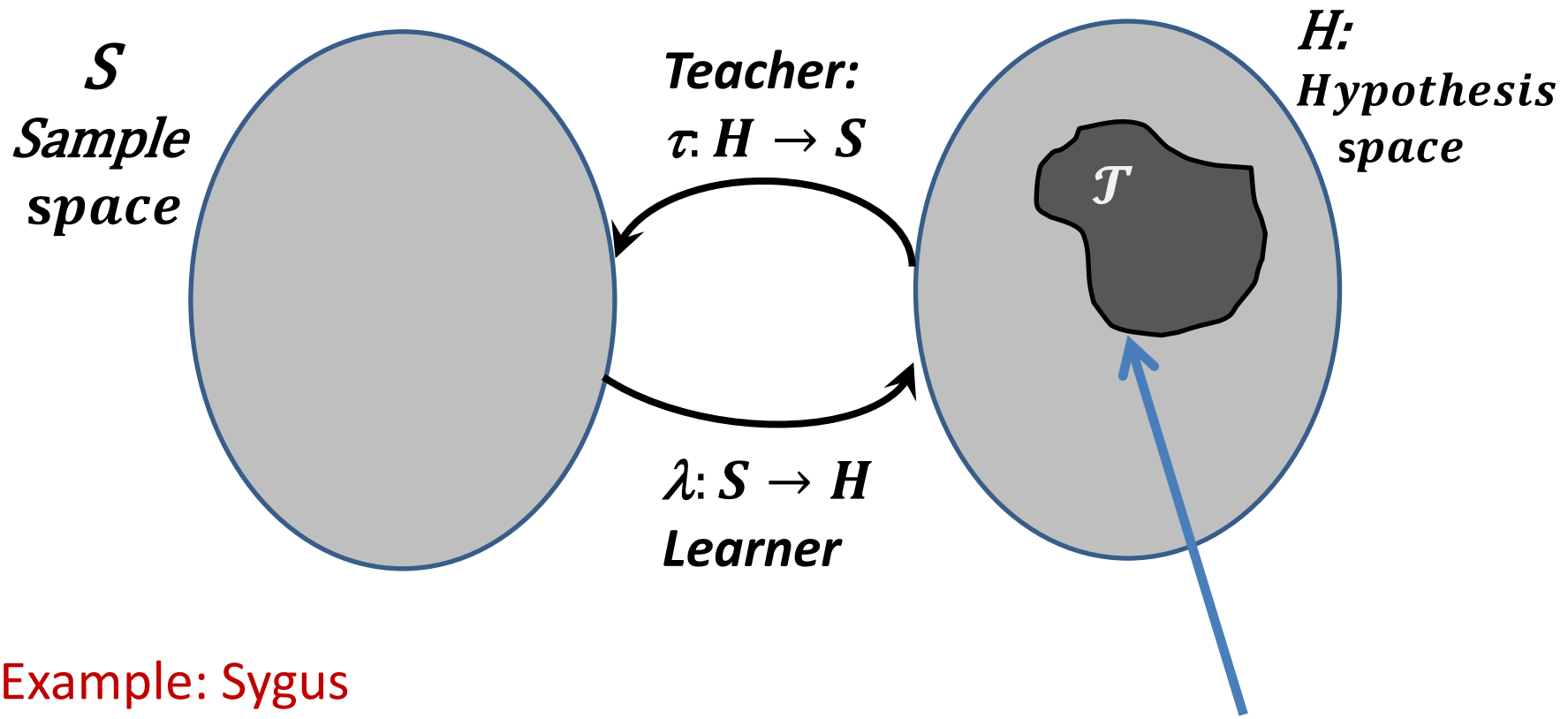
}



Example: Invariant synthesis

- Includes Init (+ve sample)
- Excludes Bad (-ve sample)
- Inductive (implication sample)

Teacher knows
Properties of target:
$P_1, P_2, \ldots P_n$
Gives samples that show
H does not satisfy P_i



Example: Sygus

- $\psi(f, \bar{v})$: **grounded formulas**

Teacher knows
$\forall \bar{x}. \psi(f, \bar{x})$

SyGuS solvers work with such samples:

- Enumerative
- Stochastic
- Symbolic
- Sketch

Question: Can we adapt machine learning algorithms to work on such samples?

Alchemist [CAV15]

- Technique based on geometry and machine learning for a small class of SyGuS problems.

- A newer version Alchemist_CSDT can learn specifications that demand ``point-wise'' functions *f expressible in LIA*
  - *Combination of constraint solvers and decision-tree learning*
  - *Alchemist_CSDT did well in the SyGuS competition.*

References

- T. M. Mitchell. Machine learning. McGraw Hill series in Computer Science. McGraw-Hill, 1997.
- Kearns, Vazirani: An Introduction to Computational Learning Theory. MIT Press, 1994
- Pranav Garg, Christof Loeding, P. Madhusudan, and Daniel Neider. ICE: A Robust Learning Framework for Synthesizing Invariants, CAV 2014.
- Pranav Garg, Daniel Neider, P. Madhusudan, Dan Roth. Learning Invariants using Decision Trees and Implication Counterexamples, July 2015.
- J.R. Quinlan. Induction on Decision Trees, Machine Learning Vol 1, Issue 1, 1986.
- Nick Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm, Machine Learning 285–318(2).
- Dan Roth: CS446 Machine Learning, Course at UIUC: http://l2r.cs.illinois.edu/~danr/Teaching/CS446-14/schedule.html
- Christof Loding, P. Madhusudan, Daniel Neider,: Abstract Learning Frameworks for Synthesis. Technical Report.

Conclusions

- Several machine learning algorithms *are robust,* or can be made robust, for use in verification problems.
- Learning Boolean formulas and with atomic formulas of the kind $n(x) \leq c$.

  Conjunctions, linear threshold functions (SVMs, Winnow, Perceptron), Decision trees (Bool + Numerical Attributes)
- Bridges:
  - Understand algorithm and make sure they learn consistent hypotheses
  - Adapt them to ICE
  - Build in convergence, in the iterative model.

- Inductive synthesis emerging as a viable alternative to white-box synthesis

THANK YOU. QUESTIONS?