CS 4700:
Foundations of Artificial Intelligence

Prof. Bart Selman
selman@cs.cornell.edu

Machine Learning:
The Theory of Learning
R&N 18.5

Machine Learning Theory

Central question in machine learning:

How can we be sure that the hypothesis produced by a learning alg. will produce the correct answer on previously unseen examples?

Question in some sense too vague… Leads to the general problem of inductive inference: How can we generalize from data?

Major advance: Computational Learning Theory

Valiant 1984; Turing Award 2010.

Valiant's Theory of Learning

Arguably the first big step to a rigorous understanding of what it means for a machine to learn.

Compare: development of the **theory of computation**
Turing, Godel, Von Neumann

Just the beginning, still many open issues.

Computational Learning Theory

How can we put machine learning on a rigorous footing?

Major advance: Valiant, 1984.

Starting point:

Induction. So far, given **training set**,
learning algorithm generates **hypothesis.**

Run hypothesis on **test set**. Says something about how good our hypothesis is. **But,** how much does it tell you?

Can you be certain??

Can never be absolutely certain about generalization...
(would have to see **all** examples)

Valiant's insight: introduce probabilities to measure a degree of certainty.

Need **Stationary assumption**: that is,
training and test examples are drawn from the **same** probability distributions.

Ex. try using "height" to distinguish men and women — better draw people form the same distribution for training and testing!

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

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

Luckily, we will see that it's generally **highly unlikely** to see a long series of bad examples!

Probabilities to the rescue: certain "bad" events (e.g. learner learns the wrong hypothesis) become exponentially rare with enough data.

Aside: Flipping A Coin

Assume, we're flipping a coin m times. We expect to observe roughly $0.5 \times m$ "heads".

Let's say we have a "bad run" (i.e., one that suggests that the coin is not fair. Say, the bad run contains 10% more heads than expected. I.e., p would appear to be 0.55!

(or higher.)

How likely / unlikely is that?

Concretely — What's the probability of

1) $m = 100$, run with more than 55 heads.
2) $m = 1000$, run with more than 550 heads.
3) $m = 10,000$, run with more than 5500 heads.

We can calculate these probabilities using the so-called *Chernoff bounds*. (Also, Hoeffding.)

We have,

$$Pr[S > (p+\gamma)m] \leq e^{-2m\gamma^2}$$

$$Pr[S > (p-\gamma)m] \leq e^{-2m\gamma^2}$$

Here, we have $p = 0.5$, $\gamma = 0.05$.

S = # of "heads"

S = # of "heads"

$m = 100$ — $Pr[S > 55] \leq 0.6$

$m = 1,000$ — $Pr[S > 550] \leq$ [redacted]

$m = 10,000$ — $Pr[S > 5500] \leq$ [redacted]

Wow!

So, when flipping a fair coin 10,000 times, you'll "never, ever" see more than 5,500 heads. And, no one ever will… I.e., with enough data, we can be very certain that if coin is fair, we won't see a "large" deviation in terms of #heads vs. #tails.

Some more experimental data

C program

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)

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)

(note the difference with trying to reach 10% over!)

Coin example is the key to **randomized algorithms.**

You get pretty accurate very fast.
(relatively few flips.)

Also, makes interesting learning algorithms possible!

Bounds show how "rare" bad runs become in large samples!

Exponential drop off — can get good results
with modest number of examples ("polynomially many")

Hope for polytime algorithms!

Aside: Can get pretty much **"certainty"** out of
probabilistic phenomena / secret behind randomized algs

E.g. estimating **integrals** / **Monte-Carlo** methods.

Worth considering!

PAC

Introduce: Probably Approximately Correct Learning

That is,

For our learning procedures, we will try to prove that:

With **high probability** our learning algorithm will find an hypothesis that is **approximately** identical to the hidden target concept.

Note: the double "hedging" — probably ... approximately...

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

Two issues: (1) (Re: High prob.) May get a "bad" sequence of training examples (many "relatively short men). Small risk but still small risk of getting wrong hypothesis.
(2) We only use a small set of all examples (otherwise not real learning). So, we may not get hypothesis exactly correct.

Finally, want efficient learning --- polytime!!

How Many Examples Are Needed?

(rather technical)

- $\mathbf{X}$ is the set of all possible examples.
- D the distribution with which we draw examples.
- $\mathbf{H}$ set of possible hypotheses.
- m number of examples in training set.

Assume, the true function f is in $\mathbf{H}$.

Valiant's genius was to focus in on the "simplest" models that still captured all the key aspects we want in a "learning machine." PAC role has been profound even though mainly to shine a "theoretical" light.

error of a hypothesis h wrt f is defined as

the probability that h differs from f on a randomly picked example:

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

This is what we were trying to measure with our **test set** before.

But, we don't have f ...

Where do we get info about f?

Approximately Correct

h is **approximately correct** iff

$$error(h) \leq \epsilon.$$

Such an hypothesis lies withing an ϵ-ball of f. Rest of hypotheses $\mathbf{H}_{bad}$.

In words: We want h such that when we pick random examples, only **rarely** does h misclassify an example (i.e, with probability less than ϵ).

Idea: show that after seeing m examples, with high probability, all consistent hypothesis will be approximately correct.

I.e., chance of a "bad" hypothesis (*but consistent with the examples*) is small (less than δ).

For our learning algorithm, we simply use a method that keeps the hypothesis consistent with all examples seen so far. Can start out with an hypothesis that says "No" to all examples. Then when first positive example comes in, minimally modify hypothesis to make it consistent with that example. Proceed doing that for every new pos example.

Let h_b be a bad hypothesis, i.e, $error(h_b) > \epsilon$.

So, chance h_b disagrees with an example $> \epsilon$.

So, prob. it agrees with a given example is $\leq (1 - \epsilon)$.

We have **So, h_b could mistakenly be learned!**

$$P(h_b \text{ agrees with } m \text{ examples }) \leq (1 - \epsilon)^m$$

$$P(\mathbf{H}_{bad} \text{ contains a hypth. consistent with } m \text{ exs. }) \leq |\mathbf{H}_{bad}|(1 - \epsilon)^m \quad \leq |\mathbf{H}|(1 - \epsilon)^m$$

We would like to make this unlikely ($\leq \delta$).

Note: we want to make it likely that all consistent hypotheses are approximately correct. So, no "bad" consistent hypothesis occur at all.

So,

$$|\mathbf{H}|(1-\epsilon)^m \leq \delta.$$

It follows:

$$m \geq \frac{1}{\epsilon}(ln\frac{1}{\delta} + ln|\mathbf{H}|). \quad (1)$$

So, how can we keep the number of examples we need down?

ϵ and δ are assumed given
(set as desired)
Keep size hypothesis class H down.
Another showing of Ockham's razor!

(1) says that if a learning alg. returns a hypothesis that is consistent with this many examples, then with prob. at least $1 - \delta$, the hypothesis has error of at most ϵ.

(1) as a function of ϵ and δ is called the **sample complexity** of the hypothesis space.

Note that we only ask from the learner to find some hypothesis consistent with the m examples.

So, in this setting the requirements on our learning algorithm are quite minimal. We do also want poly time though.

Consider: **H** space of all Boolean functions.

$|\mathbf{H}| = 2^{2^n}$.

Sample complexity grows with 2^n.

Same as number of **all possible** examples!

Therefore, learning algorithm cannot do better than lookup table, if it merely returns hypothesis consistent with given examples.

What is this saying intuitively about **H**?

What does this mean for *e.g* **neural nets**?

Learning in general?

Aside: Shannon already noted that the vast majority of Boolean functions on N letters look "random" and cannot be compressed. (No structure!)

Solution

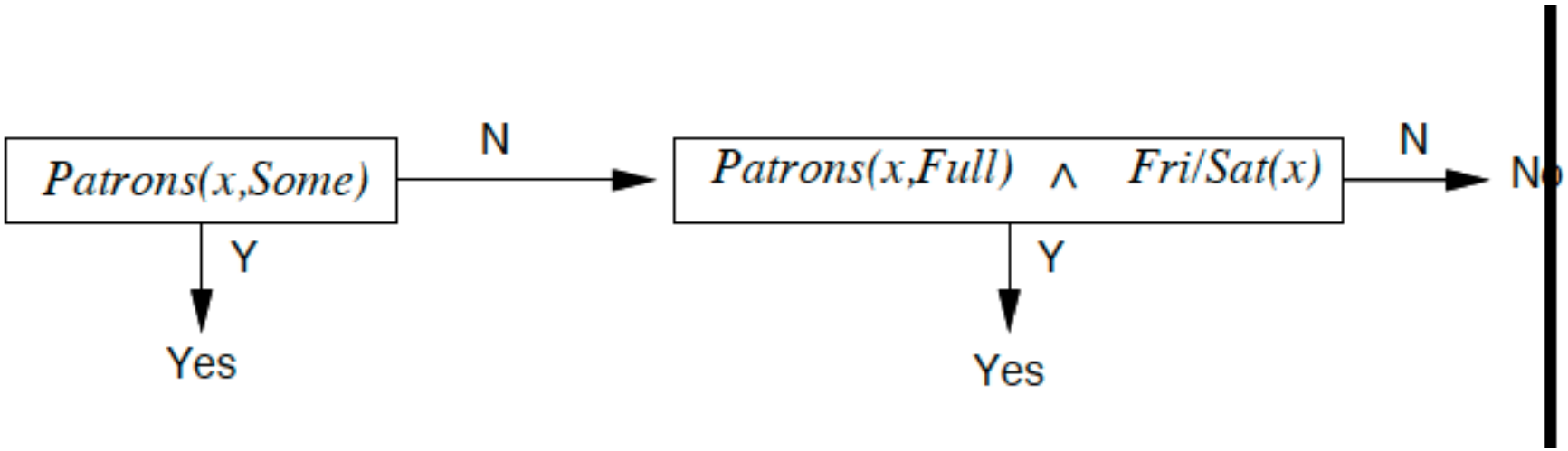
1) Force algorithm to look for "smallest" consistent hypothesis
   We considered this for decision tree learning.
   (often worst-case intractable)

2) Restrict form of Boolean function — size of hypotheses space.
   E.g. if only hypotheses are **conjunctions of literals**,
   then we only need poly number of examples!
   Example: decision lists

Decision lists

Resemble decision trees, structure is simpler, decisions are more complex.

$$\forall\, WillWait(x, Some) \Leftrightarrow (Patrons(x, Some) \vee (Patrons(x, Full) \wedge Fri/Sat(x)))$$


Patrons(x,Some)
N
Patrons(x,Full) ∧ Fri/Sat(x)
N
Y
Yes
Y
Yes

Each test is a conjunction of literals.

Labels can be "yes" or "no".

If you allow arbitrarily many lits per tests, then decision lists can express all Boolean functions

Consequence for PAC learning?

(Can view as "decision tree"; what form?)

Limit expressiveness of tests:

involve at most k literals.

Becomes PAC learnable!

We have to show that we don't need too many

examples to find a good k-DL(n) hypothesis.

n Boolean attributes.

Limit expressiveness of tests:

Language of tests: $Conj(n, k)$

At most $3^{|Conj(n,k)|}$ sets of tests (Yes/No/absent).

All possible orders, so:

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

After some work, we get

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

(useful exercise! try mathematica)

plug in (1), what is the key point here?

What if k approaches n?

We get

$$m \geq \frac{1}{\epsilon}\left(ln(\frac{1}{\delta}) + O(n^k\ log_2(n^k))\right)$$

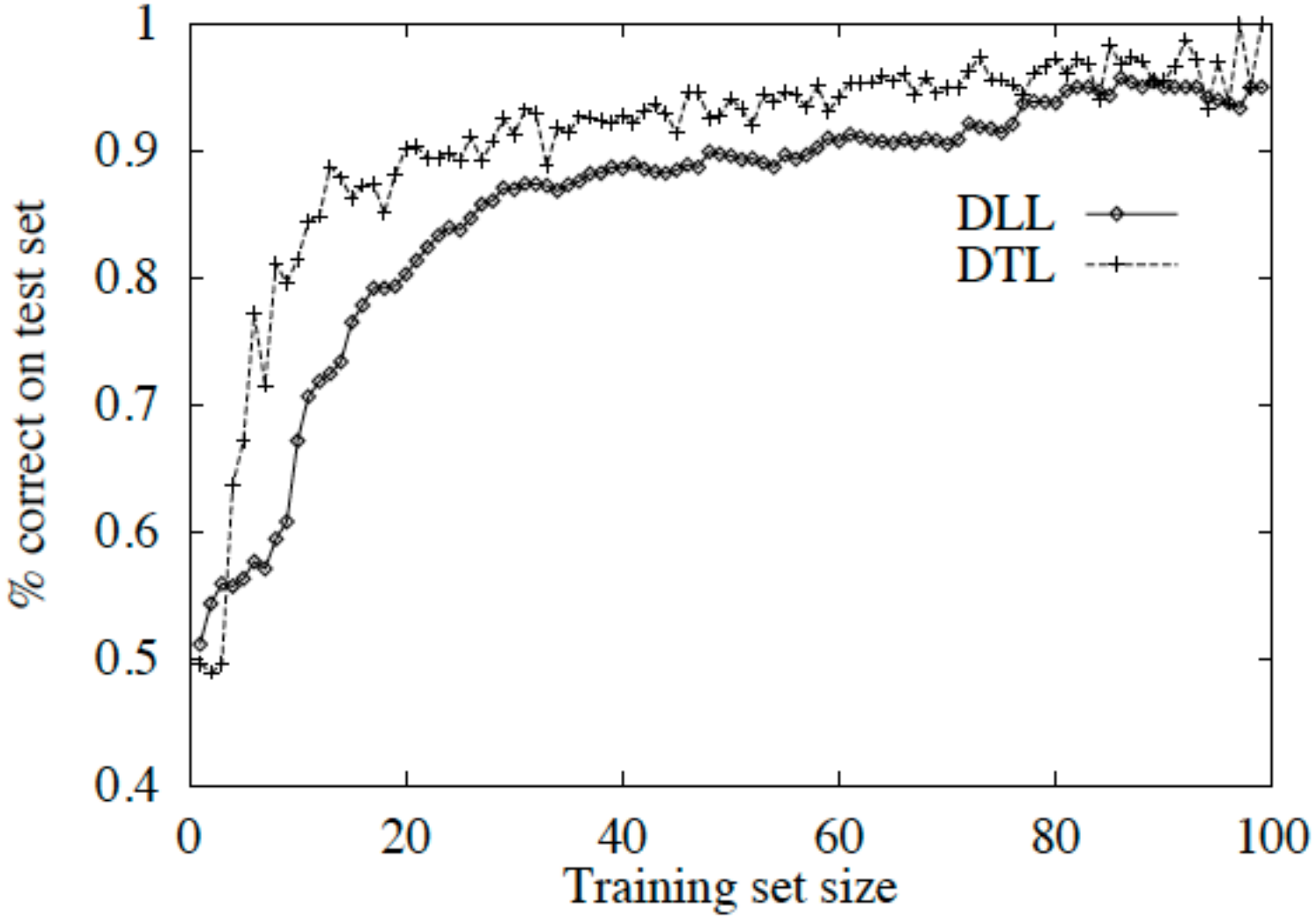
So, for fixed k, need only a polynomial number of examples!

Still need polytime learning alg. to generate a consistent decision list with m examples.

Now we need an algorithm that can find a consistent hypothesis. Use simple greedy algorithm.

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* − $examples_t$)


1
0.9
0.8
0.7
0.6
0.5
0.4
% correct on test set
0
20
40
60
80
100
Training set size
DLL
DTL

Why decision list somewhat worse than decision tree?

Can we actually be sure that our concept can be learned as a k-DL?

(Note not clear figure is for fixed k.)

Learning a **conjunction of literals** is another example of a PAC learnable language.

PAC learning is an important advance in learning theory. Unfortunately, also many negative results. Practical algorithms somewhat limited.

Limitations? Is it the right model?

Examples

1) **H** space of all Boolean functions.
   not PAC learnable; space too big;
   need too many examples.

2) Decision lists (with limited test size).
   PAC learnable.

3) Conjunction of literals.
   PAC learnable.

Examples

4) k-term DNF.

$T_1 \vee T_2 \vee \ldots \vee T_k$, with each T_i conjunction of literals.

not PAC learnable.

Polynomial sample complexity (small $|H|$) but intractable to find consistent hypothesis (most likely).

5) k-CNF is PAC learnable!

Some more PAC learnability examples.

Learning a **conjunction of literals** is another example of a PAC learnable language.

E.g. concept: $Old \wedge \neg Tall \wedge Rich \wedge \ldots$

We have: $|H| = 3^n$

Substitution in (1) gives us:

$m \geq \frac{1}{\epsilon}(n\, ln(3) + ln(\frac{1}{\delta}))$

Example: Learn a conjunction with up to 10 literals.

We desire 95% probability that we find an hypothesis with an error of less than 0.1.

How many examples do we need?

$m = \frac{1}{0.1}(10\ ln(3) + ln(\frac{1}{0.05})) = 140$

Surpise: **Not that many!**

Again, the laws of probability...

E.g. in restaurant case we got something with only 12 examples. Figures with up to 100 examples already very good.

PAC: Concluding Remarks

Valiant showed:

a.) Machines can provable learn certain classes of concepts.

b.) Classes are nontrivial and of interest from a "knowledge representation" perspective.

c.) Process takes only a feasible number of steps (and thus feasible number of examples).
(contrast with previous models)

Some issues to consider:

- Impact on the practice of Machine Learning?
  (consider decision tree learning / backpropagation etc.)
- Connection to reasoning and acting?
- Place for "background" knowledge?

Note: in PAC, background knowledge only through **syntactic** form of concepts.

Major Spinoff from PAC: Boosting

Question: Given a learning algorithm that does only slightly better than **random guessing**, can we make it better?

Surprising answer: **Yes**, we can **boost** its performance to be **almost perfect!**

How? (Schapire 1990) — Basic ideas: run many copies of of the algorithm in a clever way.

Technique is now also popular in practice:

Learn many different decision trees / neural nets / decision lis etc. Let them **vote** for the answer on an unseen case.

Works well on stockmarket data!

Also called: "combining experts"

Concludes PAC learning.

Before going to **Neural Netsworks** let's consider

Question 18.1 R&N

Discuss how infant learns to speak and understand a language.

Possibly most powerful example of learning (if it is learning!).

Issues

Need to: *recognize speech, learn vocabulary,*
learn grammar, and learn semantics and pragmatics.

What feedback does child receive?

Enough examples? What kinds of examples?

How many examples sentences?

Reinforcement learning based on general well-being?

Possible natural tendency for "mimicry" essential...

Some direct feedback (parents: "shoe", "table", etc.)

But adults do not appear to "correct' child's speech!

Also, mainly (only?) positive examples.

Chomsky (1960's) **"poverty of the stimulus"** argument:

basic universal grammar of language **must be innate.**

(simply not enough examples; also no negative ones.)

Recent advances in learning e.g. on **probabilistic context free grammars** re-visits the issue!

Resolution possibly within next 10 years.