Foundations of Artificial Intelligence

CS472/3 — Fall 1999

Lecture #26

Bart Selman

Learning, so far.

Decision Tree Learning

perhaps the most common used method in practice

datamining, IBM etc.

Evaluation:

training set and test set / direct tradeoff

overfitting / pruning: learning curves

Next: example of very different method

"Cased-Based Learning"

Case-Based Reasoning/Learning

Quite different from standard logical reasoning.

Idea: store large number of previously seen cases
Attempt to match current (unseen) case to **closest** stored one.

Appears quite suitable for legal reasoning (1,000,000+ cases).

What would be some of the difficulties?

Slide CS472–3

Closely related: Case-Based Learning

k-nearest neighbor learning.

A: set of features, $A_1, \ldots, A_n$ that describe the problem

$X = X_{a_1} X_{a_2} \ldots X_{a_n}$, where X_{a_i} is the value of feature A_i in example X

$f(X) : X \rightarrow c \in C = \{c_1, \ldots, c_m\}$

The *case base* is the set of training examples $(X_1, f(X_1)), (X_2, f(X_2)), \ldots$

Slide CS472–4

k-nearest neighbor algorithm for computing $f(X)$:

1. Compare new example, X, to each case, Y, in the case base and calculate for each pair:

$$sim(X,Y) = \sum_{i=1}^{n} match(X_{A_i}, Y_{A_i})$$

   where $match(a,b)$ is a function that returns 1 if a and b are equal and 0 otherwise.

2. Let $R =$ the top k cases ranked according to sim
3. Return as $f(X)$ the class, c, that wins the majority vote among $f(R_1), f(R_2), \ldots, f(R_{|k|})$

Slide CS472–5

Example of case retrieval

outlook	temp	humidity	windy	plan	*sim*
sunny	hot	high	false	cs472	3
sunny	hot	high	true	cs472	2
overcast	hot	high	false	soccer	2
overcast	mild	normal	true	football	1
rain	mild	high	false	cs472	3
rain	cool	normal	false	soccer	1

A: outlook, temp, humidity, windy
$C = \{$soccer, cs472$\}$
$k = 3$ (majority vote top 3 cases)

Slide CS472–6

test case: X = sunny mild high false

top three: case 1, 5, and 2 (random tie breaking)
majority vote: cs472.

Slide CS472–7

Case-Based Learning (Reasoning)
    Drawbacks? Difficulties?
    Would it work on our restaurant examples?

Still, algorithms outperforms all others on
    certain challenging tasks.

E.g. handwritten character recognition (postal service).

Slide CS472–8

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!

Slide CS472–11

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!

Slide CS472–12

Aside: Flipping A Coin

Slide CS472–13

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!

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.

Slide CS472–14

$m = 100$ — $Pr[S > 55] \leq 0.6$
$m = 1,000$ — $Pr[S > 550] \leq 0.007$ $(\leq 0.1\%)$
$m = 10,000$ — $Pr[S > 5500] \leq 10^{-22}$

Slide CS472–15

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 $p = 0.5$, $\gamma = 0.05$.

Slide CS472–16

Some more experimental data

Slide CS472–17

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)

Slide CS472–18

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!)

Slide CS472–19

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

You get pretty accurate very fast.

(relatively few flips.)

Slide CS472–20

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!