Lecture07a: No Free Lunch

CS540 2/27/18

Announcements

On-campus students:
- Make sure I am wearing the microphone
- Check the clip!

All students:
- Reading assignment
  - D. Whitley *A Genetic Algorithm Tutorial* (1994)
  - Link is on the resources page of the class web site
  - Emphasis on hyperplane analysis and theory of GAs

Where are we?

We have looked at local search algorithms

We have looked at population-based algorithms

You are experimented with algorithms on specific problems

Big Question: What algorithm is best?
- Overall?
- For a specific problem?

So far, practice but no theory

No Free Lunch (NFL)

A set of theorems about search & optimization

General idea:
- No algorithm is best for all problems
- No algorithm is better than random over the space of all possible problems
- In fact, all algorithms are equal over the space of all possible problems

All you can claim:
- Certain algorithms are better on certain types of problems

Wolpert, David H., and William G. Macready. "No free lunch theorems for optimization." *IEEE transactions on evolutionary computation* 1.1 (1997): 67-82.

NFL: Setup

Assume a discrete search space X
- |X|is finite (but arbitrarily large)

Assume a space of possible "cost" values Y
- |Y| is finite (but arbitrarily large)
- E.g. 32 bit or 64 bit representations

Cost functions are mappings $f: \mathrm{X} \rightarrow \mathrm{Y}$
- Deterministic, static
- There are versions of NFL that apply to time varying f's

The set of all possible problems F is therefore $F = \mathrm{Y}^{\mathrm{X}}$
- $|F| = |\mathrm{Y}|^{|\mathrm{X}|}$

Search Algorithm Complexity

Given a state $\chi \in \mathrm{X}$
- Algorithms use the cost function to get $y = f(\chi)$
- Count the number of *unique* calls $y = f(\chi)$
- In essence, assumes perfect memory
  - No penalty for asking again for $f(\chi)$ for the same χ
- Note that χ may be an element of the neighborhood, not the current state.
  - We are counting unique call to the cost function

Algorithms

A sample is a state with its computed cost function
- $d = (\chi \in X, y \in Y)$

An optimization path is a time series of samples
- $d_m \equiv \left\{\left(d_m^x(1), d_m^y(1)\right), \left(d_m^2(2), d_m^y(2)\right), \cdots, \left(d_m^x(m), d_m^y(m)\right)\right\}$

Optimization algorithms are
- $a: d \in D \rightarrow \{\chi | \chi \notin d_\chi\}$

Performance measures are $\Phi(d_m^y)$
- Computed over every sample
- E.g. max.

NFL Theorem (version 1)

For any pair of algorithms a_1 and a_2:

$$\sum_f P(d_m^y | f, m, a_1) = \sum_f P(d_m^y | f, m, a_2)$$

In other words: $\sum_f P\left(\Phi(d_m^y | f, m, a)\right)$ is independent of a!

Proof

General Idea
- By summing over all possible *f*, we ensure that the past states have no bearing on the future states

Proof Method: Induction

Base Case: m = 1
- $d_1 = \{d_1^x, f(d_1^x)\}$ note: d_1^x selected by a
- $\sum_f P(d_1^y | f, m = 1, a) = \sum_f \delta\left(d_1^y, f(d_1^x)\right)$
- $\sum_f P(d_1^y | f, m = 1, a) = |Y|^{|X|-1}$
- Therefore, the sum is independent of a

Proof: Induction Step

Show: if $\sum_f P(d_m^y | f, m, a)$ is independent of *a*, then $\sum_f P(d_{m+1}^y | f, m+1, a)$ is independent of *a*.

$$\begin{aligned} &P(d_{m+1}^y | f, m+1, a) \\ &= P\left(\left\{d_{m+1}^y(1), \cdots, d_{m+1}^y(m)\right\}, d_{m+1}^y(m+1) | f, m+1, a\right) \\ &= P\left(d_m^y, d_{m+1}^y(m+1) | f, m+1, a\right) \\ &= P\left(d_{m+1}^y(m+1) | d_m^y, f, m+1, a\right) P\left(d_m^y | f, m+1, a\right) \end{aligned}$$

1st step is definitional, 2nd is substitution, 3rd is Bayes Rule

Proof: Induction (cont.)

$d_{m+1}^y(m+1)$ depends on *x*, *f* and nothing else

$$\begin{aligned} &\sum_f P\left(d_{m+1}^y(m+1) | d_m, f, m+1, a\right) \\ &= \sum_{f,x} P\left(d_{m+1}^y(m+1) | f, x\right) P\left(x | d_m^y, f, m+1, a\right) \\ &= \sum_{x,f} \delta\left(d_{m+1}^y(m+1), f(x)\right) P\left(x | d_m^y, f, m+1, a\right) \end{aligned}$$

So

$$\begin{aligned} &\sum_f P\left(d_{m+1}^y | f, m+1, a\right) \\ &= \sum_{f,x} \delta\left(d_{m+1}^y(m+1), f(x)\right) P\left(x | d_m^y, f, m+1, a\right) P\left(d_m^y | f, m+1, a\right) \end{aligned}$$

Proof: Induction (III)

$$P(x | d_m, a) = \delta\left(x, a(d_m)\right)$$
$$P(d_m | f, m+1, a) = P(d_m | f, m, a)$$

$a = \left(d_m^x, d_m^y\right)$ does not depend directly on f, so

$$P\left(x | d_m^y, f, m+1, a\right) = P(x | d_m, a) P\left(d_m^x | d_m^y, f, m+1, a\right)$$

$$\begin{aligned} &\sum_f P\left(d_{m+1}^y | f, m+1, a\right) \\ &= \sum_{f, d_m^x} \delta\left(d_{m+1}^y(m+1), f\left(a(d_m)\right)\right) \times P(d_m | f, m, a) \end{aligned}$$

Proof: Induction (IV)

We are summing over f and d_x^m. Split the summation over f:

$$\sum_f P\left(d_{m+1}^y | f, m+1, a\right)$$
$$= \sum_{d_m^x} \sum_{f(x \in d_m^x)} P(d_m | f, m, a) \sum_{f(x \notin d_m^x)} \delta\left(d_{m+1}^y (m+1), f\left(a(d_m)\right)\right)$$

But $\sum_{f(x \notin d_m^x)} \delta\left(d_{m+1}^y (m+1), f\left(a(d_m)\right)\right) = |Y|^{|X|-m-1}$

Proof: Induction (V – and last!)

$$\sum_f P\left(d_{m+1}^y | f, m+1, a\right) = |Y|^{|X|-m-1} \sum_{f(x \in d_m^x), d_m^x} P(d_m | f, m, a)$$

$$\sum_f P\left(d_{m+1}^y | f, m+1, a\right) = \frac{1}{|Y|} \sum_{f, d_m^x} P(d_m | f, m, a)$$

$$\sum_f P\left(d_{m+1}^y | f, m+1, a\right) = \frac{1}{|Y|} \sum_f P(d_m | f, m, a)$$

By assumption, the right side is independent of a, so the proof is complete.

What does NFL tell you?

It's basically a counting argument

When you average (or sum) over *all possible* cost functions $f: X \to Y$, no algorithm is better than any other
- or better than random
- or better than picking the worst neighbor in local search

But most cost functions don't make sense
- Most problem domains have structure

One algorithm is only better than another in the context of a domain
- i.e. a restricted set (or uneven distribution) of *f*.

Geometric Intuition

Assume a non-even distribution over f
- e.g. any real problem domain

Let p be the vector of length $|F|$ that is the probability of each cost function

Let $v_{d_m^y, a, m}(f) \equiv P\left(d_m^y | m, a, f\right)$

$P\left(d_m^y | m, a\right) = \sum_f P\left(d_m^y | m, a, f\right) P(f) = v_{d_m^y, a, m} \cdot p$

Dot product is related to the cosine of the angle
- v is a vector describing the optimization/search algorithm
- p describes the problem domain
- The probability of success depends on the match (angle between) the algorithm and the domain