Famine of Forte

This paper uses a search framework to prove several bounds concerning the proportion of good problems for a search algorithm.

The paper uses a finite **search space** Ω, and objective is to find an element of the **target set** $T \subset \Omega$. Given an order of the search space Ω, we can represent T as a binary vector of size Ω, because each element of the search space corresponds to one index in our vector. This vector is the **target function**.

Finally, the paper describes an information resource F. Similar to the No Free Lunch for Optimization paper, Famine of Forte treats F as an oracle from which our algorithm gets all information about a particular point $\omega \in \Omega$. $F(\omega)$ is the evaluation of ω. Interestingly, this paper abstracts this resource as a finite bit string.

Given a search space Ω, a target set T, and an information resource F, a search problem is defined as (Ω, F, T). Since we are abstracting F and T to be finite-length bit strings, we can do lots of fun things!

The paper describes a **Black-box Search Algorithm**, that attempts to find points ω in the target set T. The algorithm keeps track of its history H, which is a series of tuples $h_i = (\omega_i, F(\omega_i))$. Each tuple keeps track of the ith call to the oracle. At iteration i, the algorithm uses the history up to this point to compute a distribution P_i over Ω. The algorithm chooses some element w_i according to the distribution and consults the oracle. Note especially that this algorithm will perform queries $i_{\max}$ times - only after does it check if we have seen a target element.

Algorithm 1 Black-box Search Algorithm

Initialize $h_0 \leftarrow (\emptyset, F(\emptyset))$.
for all $i = 1, \ldots, i_{\max}$ **do**
  Using history $h_{0:i-1}$, compute P_i, the distribution over Ω.
  Sample element ω_i according to P_i.
  Set $h_i \leftarrow (\omega_i, F(\omega_i))$.
end for
if an element of T is contained in any tuple of h **then**
  Return *success*.
else
  Return *failure*.
end if

We care about the performance of our search algorithm, since it is sufficiently general to represent many ML algorithms.

Since the algorithm performs $i_{\max}$ queries to the oracle, but this number can change, we may measure its performance by the expected per-query probability of success $q(T|F)$. The paper goes on to discuss how since this expectation is over all sources of randomness, it is equal to the probability of success for samples drawn from an appropriately averaged distribution - I don't really understand this.

$$q(T, F) = \mathbb{E}_{\tilde{P},H}\left[\frac{1}{|\tilde{P}|}\sum_{i=1}^{|\tilde{P}|} P_i(\omega \in T) \,\middle|\, F\right]$$

The paper defines $p(T, F)$ to be the expected probability of success when performing uniform random sampling, and uses this a baseline for the performance of our algorithm. It may be worth checking out the following paper, since it is referenced by Famine of Forte when discussion $p(T, F)$.

W. Dembski and R. Marks II, "Conservation of information in search: Measuring the cost of success," Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 39, no. 5, pp. 1051 –1061, sept. 2009

Theorem 1. Famine of Forte

This result shows that the proportion of search problems with performance $q(T, F) \geq q_{\min}$ is bounded above. Formally, the paper defined a set τ_k, which is a set of k-hot target vectors. That is, τ_k contains all possible k-element target sets.

The paper restricts the potential information resources to be in the set B_m, where B_m is any set of binary strings such that each string has length m or less. The paper then defines sets R and $R_{q_{min}}$, where R contains all search problems with k-length target sets, and $R_{q_{min}}$ those problems that perform as well as q_{min}. Their definitions are below:

$$\begin{aligned} \tau_k &= \{T | T \subset Q, |T| = k \in \mathbb{N}\} \\ R &= \{(T, F) | T \in \tau_k, F \in B_m\} \\ R_{q_{min}} &= \{(T, F) | T \in \tau_k, F \in B_m, q(T, F) \geq q_{min}\} \end{aligned}$$

Then

$$\frac{|R_{q_{min}}|}{|R|} \leq \frac{p}{q_{min}} = \frac{k/|\Omega|}{q_{min}}$$

The same result holds as $m \to \infty$ - as the size of the information sets grows.

Essentially, we are given a search space Ω. In that space, the proportion of problems (Ω, F, T) in which our algorithm performs as well as q_{min} is bounded above by an exact quantity.

Corollary 1. (Conservation of Active Information of Expectations)

If we define $I_{q(T,F)} = -\log(p/q(T, F))$, and effectively use this metric as our performance (with minimum performance b), then we get the result

$$\frac{|R_b|}{|R|} \leq 2^{-b}$$

This uses the exact same set construction as theorem 1. Note that $I_{q(T,F)}$ looks very similar to the difference in surprisals of p and q - p and q are expected per-query probabilities of success. Our algorithm does better than uniform random sampling if I is large, since that indicates a higher per-query probability of success. Thus the proportion of problems that do as well as b under this metric is bounded above by 2^{-b}. The paper discusses how this shows that the improved search method is equivalent to a uniform random sampler on a reduced search space. (I'm not really sure how this works)

Theorem 2. Famine of Favorable Strategies

Theorem 2. *(Famine of Favorable Strategies) For any fixed search problem* (Ω, t, f), *set of probability mass functions* $\mathcal{P} = \{P : P \in \mathbb{R}^{|\Omega|}, \sum_j P_j = 1\}$, *and a fixed threshold* $q_{min} \in [0, 1]$,

$$\frac{\mu(\mathcal{G}_{t,q_{min}})}{\mu(\mathcal{P})} \leq \frac{p}{q_{min}},$$

where $\mathcal{G}_{t,q_{min}} = \{P : P \in \mathcal{P}, t^\top P \geq q_{min}\}$ *and* μ *is Lebesgue measure. Furthermore, the proportion of possible search strategies giving at least b bits of active information of expectations is no greater than* 2^{-b}.

To be honest, I am not sure what this theorem is saying at the moment. My guess is that over all possible sequences $\{P_i\}_i$ of probabilities we can bound the number of algorithms that perform well on a given search problem.

The paper discusses below how this places bounds on the number of favorable strategies on a given problem. This result and theorem 1 show that favorable problems for strategies and favorable strategies for problems are rare.

Theorem 3. Success Under Dependence

Theorem 3. *(Success Under Dependence) Define*

$$q = \mathbb{E}_{T,F}\left[q(T,F)\right]$$

and note that

$$q = \mathbb{E}_{T,F}\left[\,\overline{P}(\omega \in T|F)\right] = \Pr(\omega \in T; \mathcal{A}).$$

Then,

$$q \leq \frac{I(T;F) + D(P_T \| \mathcal{U}_T) + 1}{I_\Omega}$$

where $I_\Omega = -\log k/|\Omega|$, $D(P_T\|\mathcal{U}_T)$ is the Kullback-Leibler divergence between the marginal distribution on T and the uniform distribution on T, and $I(T;F)$ is the mutual information. Alternatively, we can write

$$q \leq \frac{H(\mathcal{U}_T) - H(T \mid F) + 1}{I_\Omega}$$

where $H(\mathcal{U}_T) = \log\binom{|\Omega|}{k}$.

This upper-bounds the expected performance of our algorithm on a search problem. The paper discusses how this bounds improves monotonically with the dependence between between our target set and information resources. That is, if our information is somewhat accurate, we can perform better!

Other than that statement, I am having a hard time understanding this theorem.

The paper then goes on to discuss examples of problems that fit under the framework, and proves theorem 1 for the space of hyperparameters for a given algorithm A. It then shows that there cannot be a fitness function (information resource) that works well for all target sets in a search space. In other words, there is no One-Size-Fits-All fitness function.

The paper then discusses why novel machine learning algorithms keep getting generated. Since the proportion of favorable strategies for a given problem is small and the proportion of favorable problems for a strategy is small, we must keep generating new strategies to deal with new problems.

Further Reading

J. Culberson, "On the futility of blind search: An algorithmic view of 'no free lunch'," Evolutionary Computation, vol. 6, no. 2, pp. 109–127, 1998

** Just want to see no free lunch in another context

Compression and machine learning: a new perspective on feature space vectors
https://ieeexplore.ieee.org/abstract/document/1607268

** Compression - ML direction.

An Introduction to Kolmogorov Complexity and Its Applications - skim chapters 2 and 3

** I read some stuff on Kolmogorov complexity, which is a way of measuring how random a string is / how hard a string is to generate. The complexity of a string is the number of characters needed to specify it (can use a programming language, so long as we include the specification in our length). Some strings are hard to generate, others are easy. There seems to already be a no-free-lunch theorem for compression based on the pigeonhold principle, but Kolmogorov complexity might help to give bounds on the number of strings that can be greatly compressed in a given language.