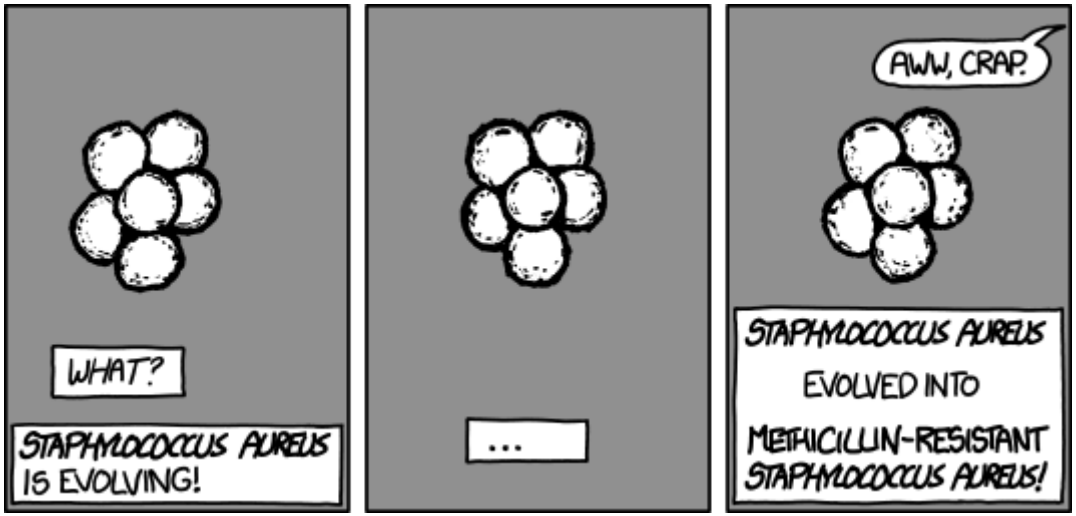
UNIVERSITY of WASHINGTON

Data Science UW Methods for Data Analysis

Introduction to Genetic Algorithms

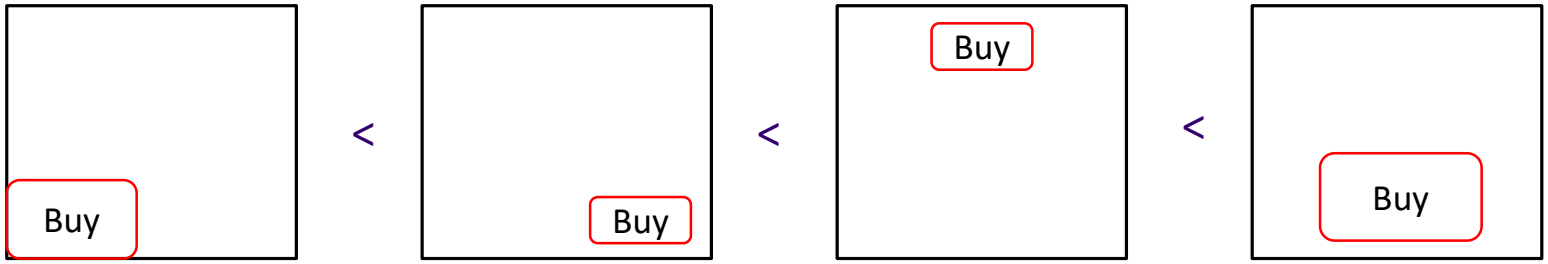
Nick McClure



"Scientists are playing reverse Pokemon, trying to avoid putting any one [bacteria] on the front lines long enough to accumulate experience to cause evolution." – Randall Munroe

Overview and Motivation

- Sometimes, we can't use linear optimization methods or gradient descent methods to optimize a solution.
  - Example: I want to know where to place the 'buy' button on my website (x,y), and how large it should be (size). It should not be assumed that the search space (x, y, size) is linear.



- Usually we can resort to a brute force method, but there are cases when the search space of parameters is too large.

Evolution and Genetic Algorithms

- Nature is a great optimizer.
  - Selection removes the least fit individuals.
  - The best fit parents create offspring with gene recombination.
  - Mutation increases the set of possible outcomes.

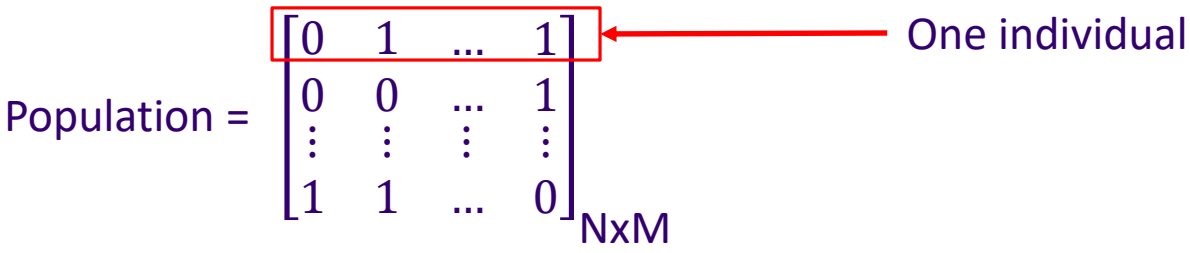
- An evolutionary algorithm copies these ideas to optimize a given problem.

Example: Have a training data set of M columns (and/or rows), and we need to identify a subset that predicts well.

Generate a Population of Individual Solutions

- First, we randomly create a population of solutions.
- You can also seed a portion of the population with a candidate solutions. (This will speed up convergence)

Example individual: [0, 1, 0, 0, 0, 1, 1, 0, 1, …]. This individual is specifying to use columns 2, 6, 7, 9, … .
Example population consists of N individuals:



Note: in R, you might create an individual by: ‘round(runif(M))’
And a population by: ‘lapply(1:N, function(x) round(runif(M)))’

Evaluate Fitness of Population

- This is the most expensive part of genetic algorithms.
- We have to evaluate each individual and see how ‘fit’ it is.

Example: Given an individual, [0, 1, 0, 0, 0, 1, 1, 0, 1, …], we have to evaluate how training just on columns 2, 6, 7, 9, … does.

We can choose to evaluate the individual outcome however we choose, but a good choice might be R^2 for a linear model, or MSE, etc…

Note, for R^2 we will want to maximize, but for MSE, we want to minimize.

Perform Selection

- We keep the top X fit of the population, and throw away the rest.

- Usual parameters are to keep between 5% and 50% of the population. 5% being very selective, and 50% being very relaxed.

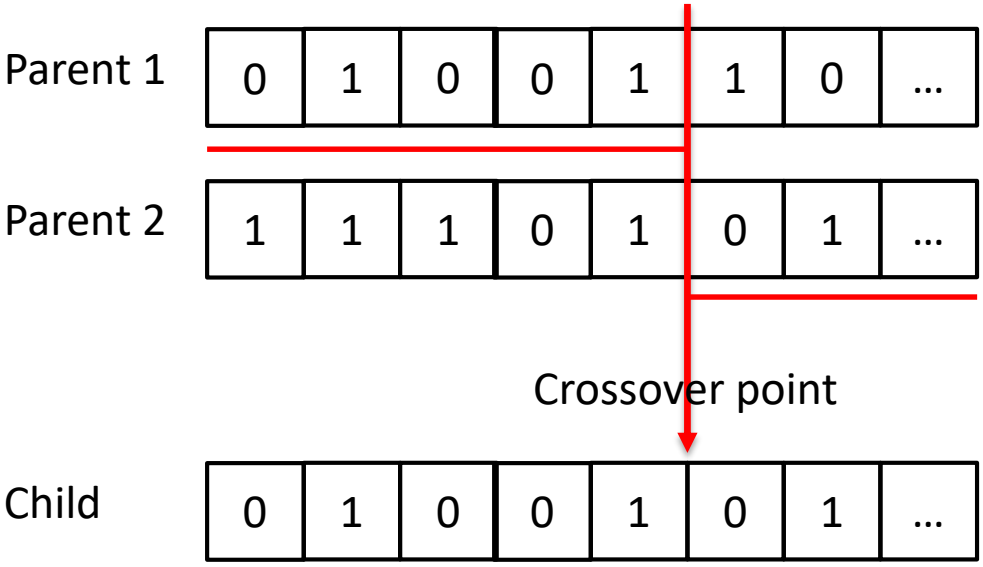
Example selection: We keep the top 10% of our population. If we had 100 individuals, our population is down to about 10 individuals.

We need a way to generate 90 more individuals…

Create Children via Recombination

- To create one child solution, we randomly select two of the top parents.
- We then select a random position along the solution, and take the first part from parent 1 and the second part from parent 2 to create a child solution.

Example selection:



} Repeat this for all children.

Mutate Children

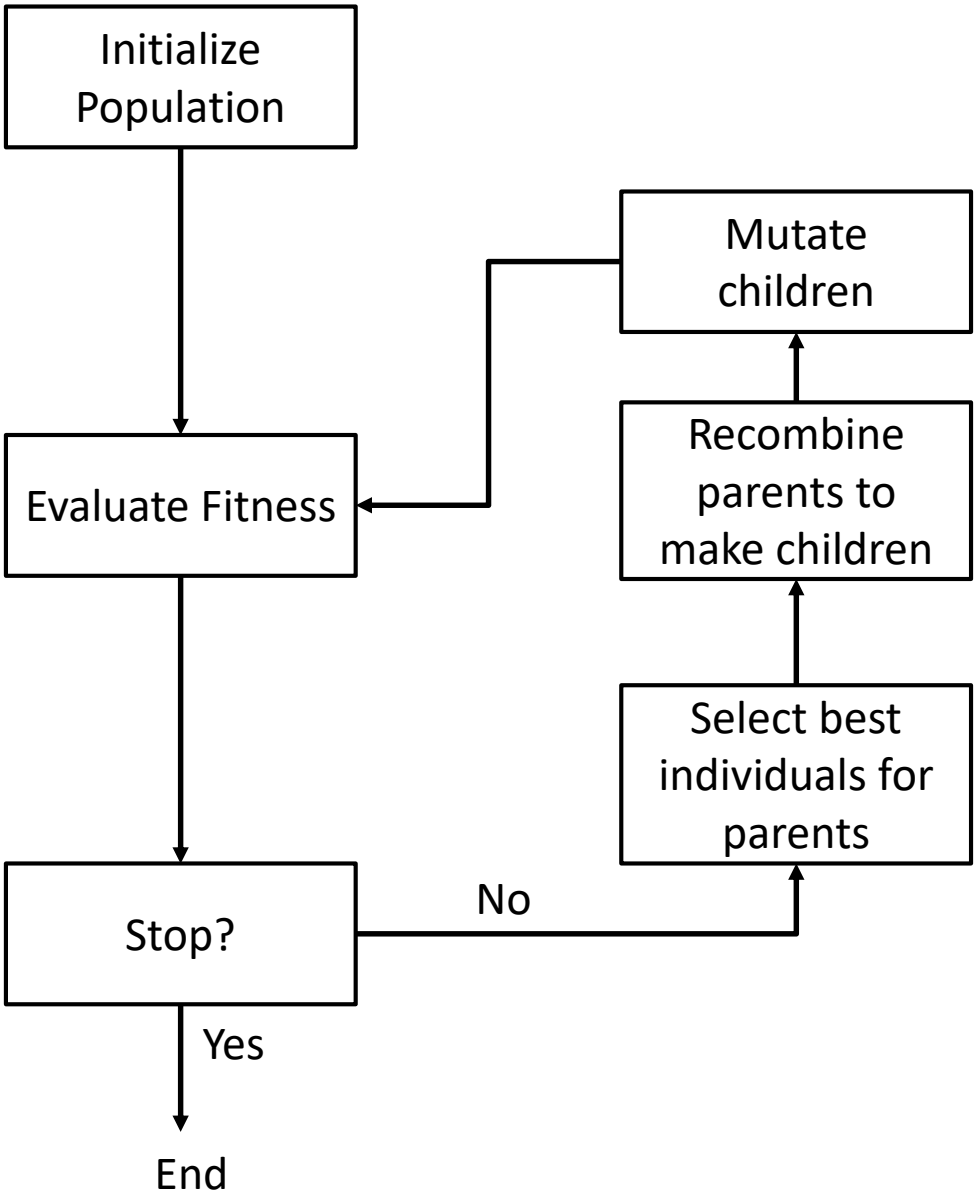
- Because our solution set of individuals might not have randomly generated enough solutions, we perform a mutation procedure on the children (not parents).
- Each parameter in each individual has the same probability of being mutated.
- Usually, p(mutation) ~ 1/length(individual), so that we expect each individual to only have one mutation each.
  - This can and should depend on the problem at hand.

Example mutation: Loop through each individual and each parameter and randomly reassign it at probability = p(mutation).

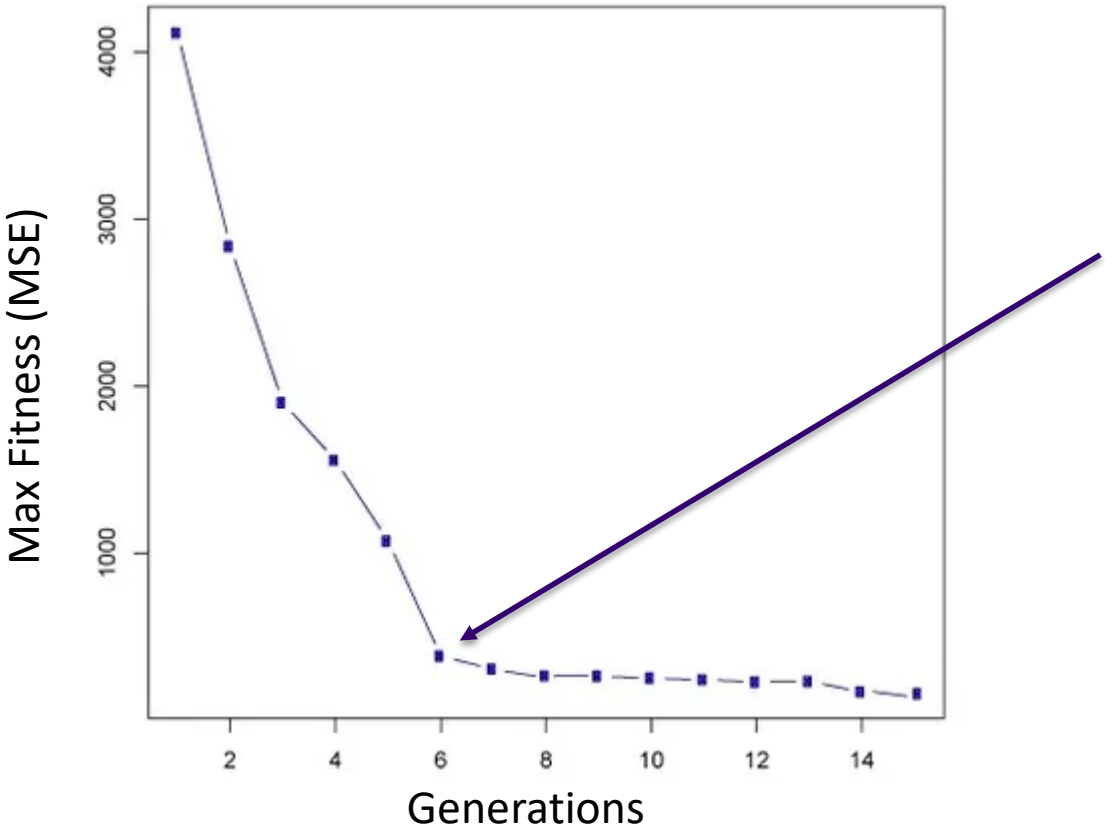
Recombine Children and Parents and Repeat…

- We repeat this process until some stopping criteria is met.
- The most common stopping criteria is to stop on the number of generations.
- We can also stop if:
  - Max fit individual doesn't change after X generations.
  - Average fitness of parents changes less than X amount.
  - …

In Summary


Initialize Population
Evaluate Fitness
Stop?
Yes
End
No
Select best individuals for parents
Recombine parents to make children
Mutate children

Evaluating Performance



After this point, you get diminishing returns for every computationally hard generation.

Look at R Example Code.