# MDL Project

Genetic Algorithms


- Dayitva Goel: 2019101005
- Prajneya Kumar: 2019114011


## 1 Overview of the Genetic Algorithm

A Genetic Algorithm is search-based optimization technique, based on Charles Darwin's Theory of Evolution. It employs the techniques of Natural Selection and Genetics to optimize a machine learning model. In this project, we try to optimize an overfit vector using genetic algorithms. We implemented different variations of said algorithm, each one of which includes the following steps:

1. Population Initiation
2. Fitness Determination
3. Continue Iterating until convergence
   (a) Parent Selection
   (b) Crossover parents
   (c) Mutate children

### 1.1 Approach 1: The Base Algorithm

1. **Population Initiation** The overfit vector provided to us is directly used with multiple instances as our initial population with a population size of 10.
2. **Fitness Determination** Fitness is determined by the formula:

   $$\frac{1}{(train\_error + validation\_error)}$$

   A probability factor for each vector is defined by the formula

   $$\frac{Fitness}{\sum Fitness}$$

3. **Start Iteration until convergence**
   (a) **Parent Selection** The same amount of individuals as the initial population are chosen as parents according to their probability factors.
   (b) **Crossover parents** Set of two parents are crossed with each other to produce two offsprings. This is done with the Single-Point Crossover Method.
   (c) **Mutate children** A randomly selected subset of the children are mutated with a randomly selected mutation factor that lies between (-1, 1).
   The mutated offsprings now form the new generation and the next iteration starts.

## 1.2 Approach 2: The Evolution

1. **Population Initiation** The overfit vector provided to us is mutated with a mutation factor randomly chosen between (0.9, 1.1) as many times as the population size, and is now treated as the initial population. The mutation function now mutates the individual based on a probability provided. Since in this step we are mutating the initial population, we keep this probability relatively high.

2. **Fitness Determination** Fitness is determined by the formula:

$$\frac{1}{(train\_error + validation\_error)}$$

   A probability factor for each vector is defined by the formula

$$\frac{Fitness}{\sum Fitness}$$

3. **Start Iteration until convergence**

   (a) **Parent Selection** The same amount of individuals as the initial population are chosen as parents according to their probability factors.

   (b) **Crossover parents** Set of two parents are crossed with each other to produce two offsprings. This is done with the Single-Point Crossover Method. After crossing over, each offspring is mutated with a crossover probability that is high in the initial iterations, and keeps on decreasing as the number of iterations increase. This is done to manipulate the Simulated Annealing Process.

   (c) **Mutate children** A randomly selected subset of the children are mutated with a randomly selected mutation factor that lies between (0.9, 1.1).

   The mutated offsprings now form the new generation and the next iteration starts.

## 1.3 Approach 3: God's Plan

1. **Population Initiation** The overfit vector provided to us is mutated with a mutation factor randomly chosen between (0.9, 1.1) as many times as the population size, and is now treated as the initial population. The mutation function now mutates the individual based on a probability provided. This probability aims to mutate 3 out of the 11 data points.

2. **Fitness Determination** Fitness is determined by the formula:

$$\frac{1}{((train\_error * 0.7) + validation\_error)}$$

   to give more weightage to the validation error.

   A probability factor is now not calculated. Instead, all potential parents are sorted based on their fitness.

3. **Start Iteration until convergence**

   (a) **Parent Selection** The same amount of individuals as the initial population are chosen as parents according to their fitness directly. This ensures that repetition of parents is reduced.

   (b) **Crossover parents** Set of two parents are crossed with each other to produce two offsprings. This is done with the Simulated Binary Crossover Method.

   (c) **Mutate children** A randomly selected subset of the children are mutated with a randomly selected mutation factor that lies between (0.9, 1.1), with a probability that aims to mutate 3 out of 11 data points in each individual.

   The mutated offsprings now form the new generation and the next iteration starts.

## 1.4 Approach 4: The Oblivion

1. **Population Initiation** Instead of using the overfit vector provided directly, an 11-D vector with all zeros is initialised and mutated at random places with a mutation factor corresponding to its index at the overfit vector. This is done as many times as the population size and is termed as the initial population.

2. **Fitness Determination** Fitness is determined by the formula:

$$\frac{1}{((train\_error*0.7)+validation\_error)}$$

   to give more weightage to the validation error.

3. **Start Iteration until convergence**

   (a) **Parent Selection** The same amount of individuals as the initial population are chosen as parents according to their fitness directly. This ensures that repetition of parents is reduced.

   (b) **Crossover parents** Set of two parents are crossed with each other to produce two offsprings. This is done with the Simulated Binary Crossover Method.

   (c) **Mutate children** A randomly selected subset of the children are mutated with a randomly selected mutation factor that lies between (0.9, 1.1), with a probability that aims to mutate 3 out of 11 data points in each individual.

   The mutated offsprings now form the new generation and the next iteration starts.

## 1.5 Approach 5: The Predator and the Prey: Natural Selection

1. **Population Initiation** Instead of using the overfit vector provided directly, an 11-D vector with all zeros is initialised and mutated at random places with a mutation factor corresponding to its index at the overfit vector. This is done as many times as the population size and is termed as the initial population.

2. **Fitness Determination** Fitness is determined by the formula:

$$\frac{1}{((train\_error*0.7)+validation\_error)}$$

   to give more weightage to the validation error.

3. **Start Iteration until convergence**

   (a) **Parent Selection** In each iteration, the number of fertile parents is reduced by two. The least fit vectors are discarded, and the best fit vectors are included in the selection list, such that no parents are repeated in one iteration. This directly simulates the Predator Prey theory of Charles Darwin, where the weak individuals are killed naturally.

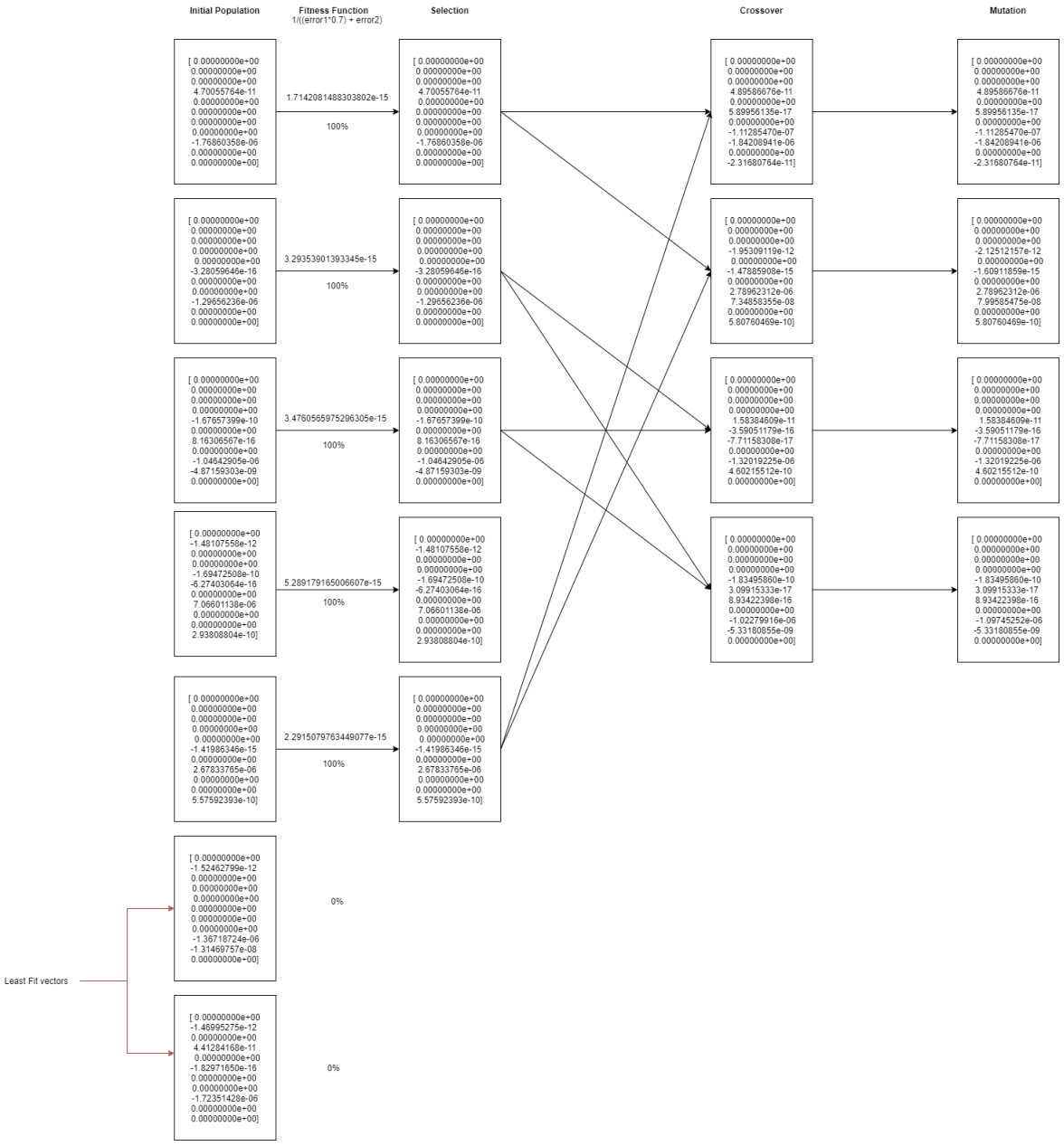
   (b) **Crossover parents** Set of two parents are crossed with each other to produce two offsprings. This is done with the Simulated Binary Crossover Method.

   (c) **Mutate children** A randomly selected subset of the children are mutated with a randomly selected mutation factor that lies between (0.9, 1.1), with a probability that aims to mutate 3 out of 11 data points in each individual.

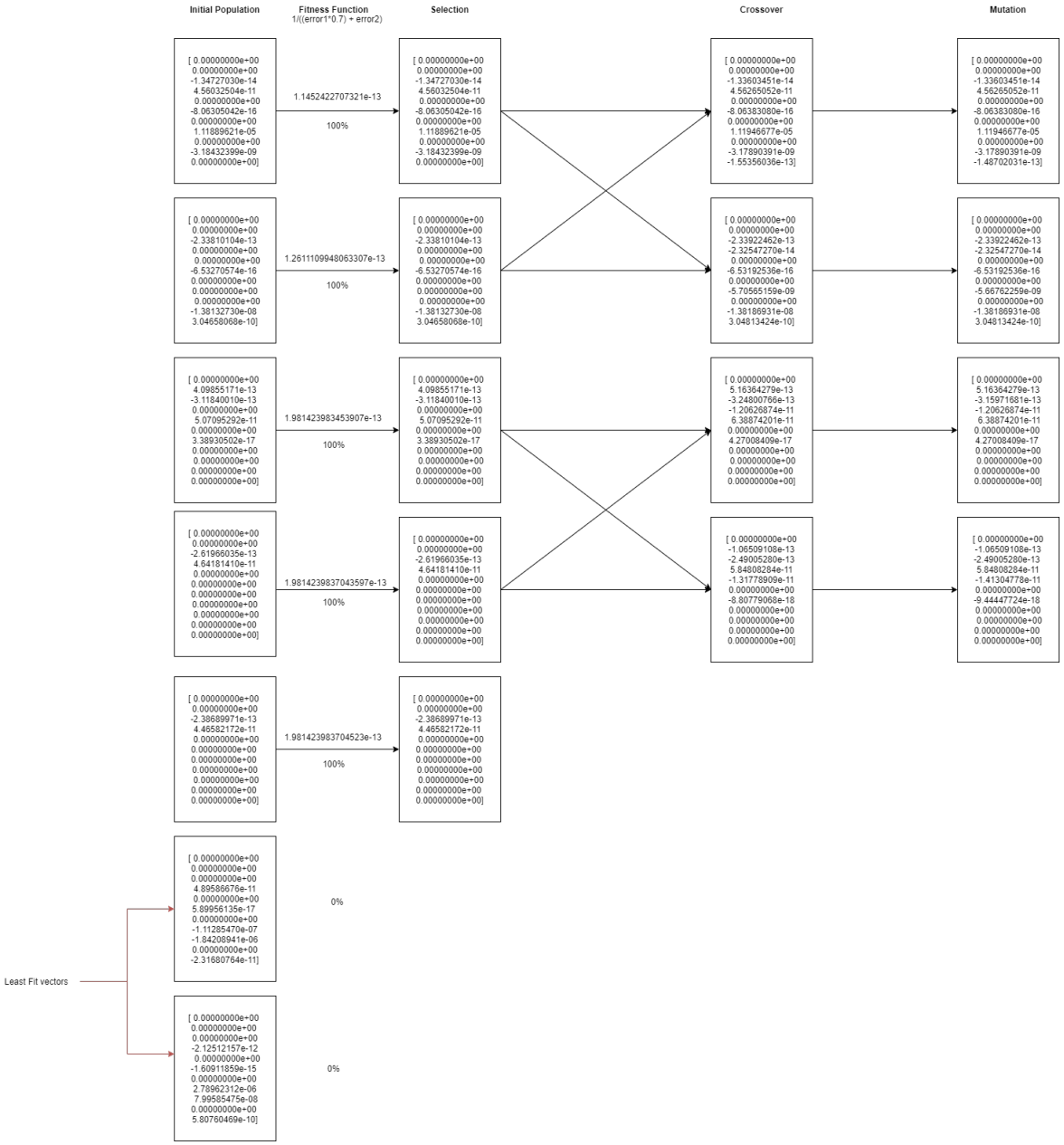
   The mutated offsprings now form the new generation and the next iteration starts.

# 2 Iteration Diagrams

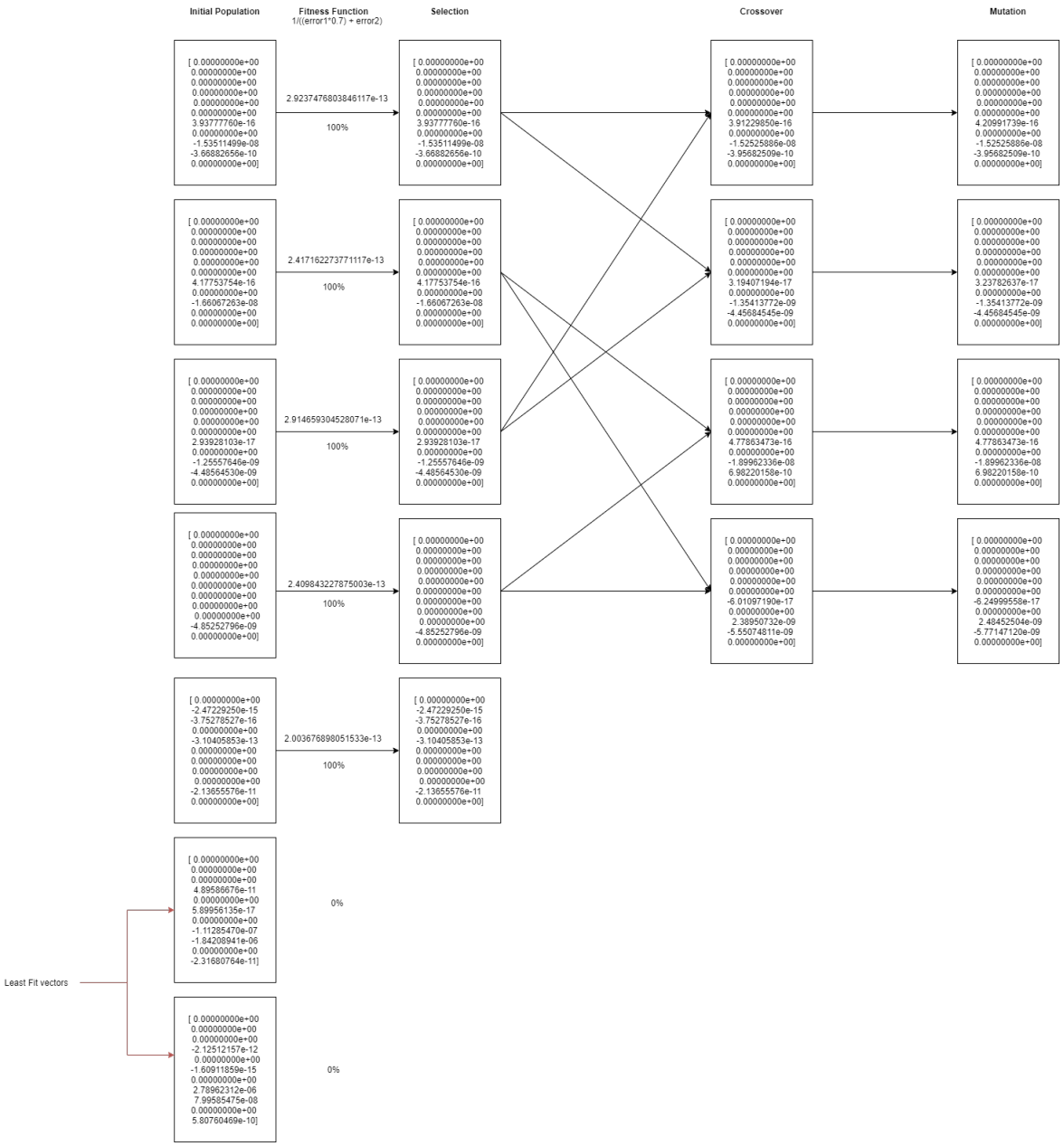
## 2.1 Iteration 1



## 2.2 Iteration 2



## 2.3 Iteration 3




# 3 Fitness Function

The fitness function returns the fitness of an individual which determines its chance of selection. It does so by calculating a linear combination of the train and validation error. A lower weight is assigned to the train error as the validation error is quite high compared to it and needs to be brought down significantly.

Precisely, the fitness function returns

$$\frac{1}{((train\_error*0.7)+validation\_error)}$$

# 4 Crossover Function

The crossover function takes two individuals and crosses over their genes to produce an offspring. This is similar to reproduction in biology. In our model, we have used **Simulated Binary Crossover** to achieve this.

Precisely, the crossover function follows the following steps:

1. Selection of two parents $x_1$ and $x_2$
2. Generation of Random number $u \in [0, 1)$
3. Calculate the cross-over point $\beta$ using the following formula:

$$\beta = \begin{cases} (2u)^{\frac{1}{\eta_c+1}}, & \text{if } u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise} \end{cases}$$

where $\eta_c$ is the distribution index

The offsprings are calculated finally with the following formula:

$$x_1^{new} = 0.5[(1+\beta)x_1 + (1-\beta)x_2]$$
$$x_2^{new} = 0.5[(1-\beta)x_1 + (1+\beta)x_2]$$

# 5 Mutation Function

The mutation function introduces a small change in the gene of an individual. It does so by multiplying the vector with a number from a given range at a random index with some probability.

The function takes arguments population, probability, mutate_from and mutate_till. The first two arguments are self-evident and the last two indicate the range.

# 6 Hyperparameters

1. **Population Size** is 30. We experimented with values of 10 and 20 as well but ultimately set on 30 as 10 and 20 were quite low for the algorithm to converge.
2. **Mutation Probability** is 0.272727. This mutates the individual at 3 out of 11 indexes on average. We found it low enough to retain a lot of the parental features but high enough to bring a change for the next generation.
3. **Mutation Range** is 0.9 to 1.1. We kept the mutation range quite small and around 1 so that the offspring does not change heavily and has fitness close to its parents.
4. **Iterations** are 15. We tried different values for iterations and for the last model as the population size is 30 which keeps decreasing by 2, the model can only be run 15 times after which there are no vectors left.

# 7 Heuristics

1. **Parent Repetition** Initially, we let the parents repeat in selection process so that the vector with the best fitness could propagate more. However, a single vector would then take over the whole population and we would get stuck at a local minima. We, therefore, did not let any vector repeat and picked the best top n vectors across all generations. This led to much better results.

2. **Zero Vector Initialisation** We ran the algorithm multiple times with the overfit vector as the initial population. However, that only took us to a local minima each time as the vector has already overfit on the training data. Therefore, we switched to a zero vector as the starting position with the overfit vector dictating the initial mutation. This reduced the errors by a huge amount.

3. **Simulated Binary Crossover** Single-point crossover at the center worked well only to a certain extent. Switching to simulated binary crossover helped us tune the amount of variation we wanted in the offspring from the parent.

4. **Best Fitness Determination** As stated above, the fitness of an individual is calculated using a linear combination of the train and validation error. We started out by giving both the errors the same weight but that did not bring down the validation error by much. We tried 0.5 as the weight for train error but then it increased very slowly. The weight 0.7 was seen as optimal after multiple iterations.

# 8 Conclusion

The best vector we achieved was [ **0.00000000e+00, 1.20951187e-13, 6.52112527e-14, -3.22055327e-12, 5.31828014e-11, 0.00000000e+00, 4.78276501e-16, 1.34109177e-07, -9.72867237e-07, 0.00000000e+00, 3.60315293e-10**]

It has a train error of 133152433593.64453 and validation error of 21611353658.14957 with fitness 8.709431434209443e-12.
The overfit vector has a train error of 13510723304.19212 and validation error of 368296592820.6967 with fitness 2.6472247e-12.

As we can see, the train error has increased by a magnitude of 10 which shows that overfitting has reduced on the train set. The validation error, on the other hand, has dropped by a magnitude of 10 indicating that the model performs well on unseen data. The fitness of our vector is also more than 3 times the fitness of the overfit vector.

We can therefore conclude that this vector generalizes better than the overfit vector and will perform better on the hidden dataset.

# 9 Others

Sources: Simulated Binary Crossover