MDL ASSIGNMENT 3 - GENETIC ALGORITHMS REPORT

TEAM NUMBER - 19
TEAM MEMBERS - PRERNA GUPTA (2018101114)   RADHIKA RAO(2018111015)

SUMMARY OF GENETIC ALGORITHM WITH ALL STEPS

1. Creation of population 0
   a. Originally we generate a random numpy array with random numbers.
   b. Add the given overfit vector to achieve faster results.
2. Find fitness of parents
   a. Send request to server through client file and find the training and validation errors.
   b. Calculate the fitness = training_error^2 + validation_error^5
3. Choose mating pool made up of fittest parents
   a. Store indices of best vectors in an array (least fitness)
   b. Store them in a numpy array and return it.
4. Crossover of parents
   a. The crossover we used is a uniform crossover.
   b. Create offspring population.
5. Mutation of offspring
   a. For each offspring , each gene is mutated with probability 0.25
   b. Gene = gene + random.choice(+1,-1)*random.unifom(0.9,1.1)*gene
6. Creation of population of the next generation
   a. New population comprises parents and children
7. Repeat for the next generation

FITNESS FUNCTION

Fitness = a^5 * b^2 where a =validation_error and b=training_errror

Explanation:
We wished that the validation error should have a slightly higher weightage compared to the training_error, as the main aim of the assignment was to reduce overfitting. Upon considering functions of the form a+2b we found that they penalized the training error.

CROSSOVER FUNCTION

We have used a uniform non-weighted crossover mating of 2 parents.
Parents are chosen randomly from the set of strongest parents who are allowed to mate.
For each gene in the child , parent1 or parent2 may be chosen with equal probability. The child inherits the gene of the chosen parent.

MUTATIONS

Each gene of every child is mutated with probability 25%. In order to conserve the characteristics of the gene , we use the function

Gene = gene + random.choice(+1,-1)*random.unifom(0.9,1.1)*gene

Hence the change or mutation of the gene depends upon the value of the gene itself.
If the modified gene is less than -10, we make it equal to -10.
If the modified gene is greater than 10, we make it equal to 10.
This is done so that we satisfy the constraints on the vector in client.py.

HYPERPARAMETERS

- Number of vectors in the population = 40

  In case the population size was more than 40, there is a larger processing time and we can have a lesser number of generations. For population less than 40, we found that the algorithm got stuck in a local minima and could not move lower than a certain error. It also lead to overfitting.

- Number of parents allowed to mate = 30

  Lesser number of parents allowed to mate converged faster but did not give the least error. It got stuck at a point.
  Having a large number of parents helps get a variety of characteristics, so even if it takes more time it is a good approach.

- Number of children created = 30

  A large number of children created allows more diversity in the population. So there is more probability that stronger individuals are generated.

This makes the algorithm converge faster.

- Number of parents passed on to the next generation = 10

  Elitism is essential because we need the better characteristics of the previous generation to be retained.
  In case parents passed on becomes higher than 25%, we found that the convergence time became much longer.

STATISTICAL INFORMATION

- Number of iterations to converge: 40-45 generations

  After this point, there is very little change in the vectors.

- Best error : approx. 3*10^6(as shown on leaderboard)

- According to me, the best vector is
  [0.0, -0.07707041236910439, -0.016238240663019488, 0.03992617698656706, -7.813010480556999e-07, 1.7206482272010859e-06, 2.488933001716129e-12, 1.3895277783043574e-10, 6.957424115362021e-14, 9.539055467066884e-13, -3.228194319415809e-15]
  It has training and validation errors respectively as
  [251832.20903676175, 211141.3044746337].
  From here we see that the difference between training and validation errors is the least and they are low enough to not be underfit.
  Hence this vector is the fittest vector for an unseen dataset.

HEURISTICS

1. **PART 1 - CROSS OVER**

   - Uniform Crossover (Unweighted)

     Picking 2 parents at random and choosing the parent from which the child will inherit the gene with a probability of 0.5

   - Uniform Crossover (Weighted)

     Picking 2 parents at random and a random int x from 0 or 1
     If x==1
     new_gene= parent1_gene*0.4 + parent2_gene*0.6

Else
new_gene = parent2_gene*0.4 + parent1_gene*0.6

- Uniform Crossover with 3 parents

  Choose 3 random parents(parent1, parent2, parent3)
  Choose a number x at random from [0,1,2]
  If x==0:
  new_gene=0.4*parent1_gene + 0.3*parent2_gene + 0.3*parent3_gene
  If x==1:
  new_gene=0.3*parent1_gene + 0.4*parent2_gene + 0.3*parent3_gene
  If x==0:
  new_gene=0.3*parent1_gene + 0.3*parent2_gene + 0.4*parent3_gene

- One point Crossover
  Choosing a point at random and taking offspring genes before this point from parent1 and after the point from parent2.

- Uniform Crossover mating most different parents
  For this method, we calculated the absolute difference between parent vectors allowed to mate as the sum of differences between individual indices.
  We chose the most different mate for a vector and allowed a given vector to mate no more than 2 times.

- Uniform Crossover mating parents with difference greater than minimum difference
  Just like above, we calculated the difference between randomly chosen parent vectors. If it was greater than a given value(1), only then were the vectors allowed to mate.

2. PART 2 - MUTATIONS

- Random mutations

  Randomly change the value of a gene to a random number between -10 and 10.

- Change gene by +1 or -1

  Change value of a gene by +1 or -1.
  If gene>10:
  gene=10
  If gene<-10:
  gene=-10

- Change gene by random.choice(+1,-1)*random.unifom(0.9,1.1)*gene
  Presently used algorithm

TRACE OF OUTPUT

Enclosed in the file trace in this folder

DIAGRAMS

Enclosed in this folder