TO RUN

- python3 json_dump.py
- python3 main.py

Each run you get only one generation, no.of times you run is no.of generations you run.Make generations==0 in line 138 from second run. (We did this to keep track of requests :p)

TEAM DETAILS

Number : 21
Name : Damn
Members : Veeral Agarwal (2019114009) , P.Sahithi Reddy (2020121011)

GENETIC ALGORITHMS

Genetic algorithms are often used in parameter selection and obtaining optimal solutions. We have been given coefficients of features(a vector of size 11) corresponding to an overfit model and our task is to apply a genetic algorithm in order to reduce the overfitting i.e. generalize the model so that the model performs better on unseen data.

SUMMARY OF ALGORITHM

The genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Five phases are considered in a genetic algorithm.

- Initial population
- Fitness function
- Selection
- Crossover
- Mutation

STEP WISE IMPLEMENTATION :

GENERATING INITIAL POPULATION

To start, we need few vectors of size 11 , we took **pop_size** no. of vectors.Now we have **pop_size** individuals each with 11 genes.We initialised them as **init_pop** to values zero for all genes.Then for each individual we randomly change few genes to value of gene at the position in the provided vector multiplied by some factor (rng) and then apply mutations to it using mutation() function as we wanted initial population to be more diverse.

In [ ]:
```
for i in range(pop_size):
        if generations==1:
                for j in range (chromosome_size):
                    tempp = random.randint(1,10)
                    if tempp < 11:
                        rng = np.random.uniform(low = 0.3, high = 0.90)
                        init_pop[i][j] = rng* temp_arr[i][j]
            else:
                for j in range (chromosome_size):
                    init_pop[i][j]=temp_arr[i][j]
            mutate(init_pop[i])
```

GETTING ERRORS AND FITNESS AND PROBABILITIES

To get train and validation errors we sent each individual to server and got back the errors.We calculated fitness from it and stored in array for the whole population.From fitness function we have calculated probabilities for each individual.We stored this fitness in fitness1 for parent population.we will get to know if its child or parent based on value of ind.

In [ ]:

```
def get_fitness(arr, ind):
        #getting errors on the data
    fitness=[0 for i in range(chromosome_size)]
    j=0
    for chromoso in arr:
        trainerr,validerr=get_errors(key,list(chromoso))
        #print(trainerr,validerr)
        fitness[j]=1/(train_factor*trainerr+validerr)
        j+=1

    if ind==1:
        for m in range(chromosome_size):
            fitness1[m]=fitness[m]
            #print(fitness)
    else:
        for m in range(chromosome_size):
            fitness2[m]=fitness[m]

    #calculate probabilities
    sum_fit=np.sum(fitness)
    for k in range(pop_size):
        probability[k]=fitness[k]/sum_fit
```

SELECTION AND CROSSOVER

Based on the probabilities calculated we have performed crossover **pop_size** no. of times.We have performed crossover at a random point.After crossover returned that child to new population.

In [ ]:

```
def selection(arr:np.ndarray):
    for i in range(pop_size):
        parent1ind=np.random.choice(pop_size,p=probability)
        parent1=arr[parent1ind]
        parent2ind=np.random.choice(pop_size,p=probability)
        parent2=arr[parent2ind]

        #printing parents after selection :
        print("printing parents after selection :")
        print("parent1:")
        print(parent1)
        print("parent2:")
        print(parent2)
        new_pop[i]=crossover(parent1,parent2)

def crossover(parent1,parent2):
    mid=random.randint(1,chromosome_size-1)
    child=np.ones(chromosome_size)
    for i in range(0,mid+1):
        child[i]=parent1[i]
    for j in range(mid,chromosome_size):
        child[j]=parent2[j]
    print(child)
    return child
```

GENERATING NEW POPULATION AND ITS FITNESS

From the above generated population we have mutated few genes and computed fitness for the new population by requesting errors from the server, using the same fitness function as above.We stored this in fitness2.

In [ ]:

```
for i in range(pop_size):
        mutate(new_pop[i])

    #printing population after mutation
    print("population after mutation :")
    for lol in new_pop:
        print(lol)

    #print(new_pop)
    get_fitness(new_pop,2)
```

GETTING NEXT GENERATION

We have merged our inital population with this new population and sorted in descending order based on fitness functions and picked top **pop_size** elements and made them as new_initial population.

In [ ]:

```
finaltup=[]
    for i in range(pop_size):
        finaltup.append((fitness1[i],init_pop[i]))

    for i in range(pop_size):
        finaltup.append((fitness2[i],new_pop[i]))
    print("final tuple:")
    print(finaltup)

    finaltup.sort(reverse=True , key=lambda x:x[0])

    print("final sorted tuple")
    print(finaltup)
    print("FFS , MIXED FITNESS FUNCTIONS IN ORDER")
    for i in range(pop_size):
        new_init_pop[i]=finaltup[i][1]
```

LOOP IT

Then we made new_init pop as initial population in the next run, therefore algorithm runs over specified generations.

In [ ]:

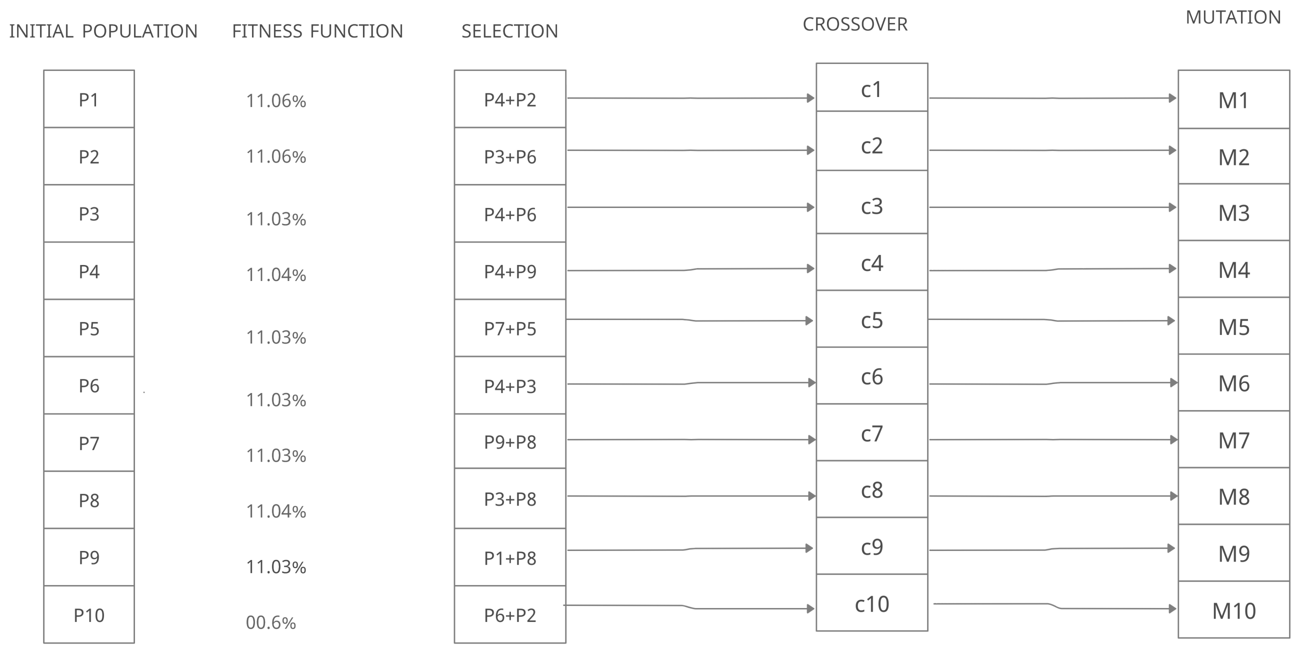
```
generations=1
total_generations=2
new_init_pop=np.zeros((pop_size,11))
while(generations!=total_generations):
#at last we can put newpop to init pop and start algo again
    init_pop=np.zeros((pop_size,11))
    new_pop=np.zeros((pop_size,11))
    fitness1=[0 for i in range(chromosome_size)]
    fitness2=[0 for i in range(chromosome_size)]
    probability=[0 for j in range(pop_size)]

    for i in range(pop_size):
        if generations==1:
            for j in range (chromosome_size):
                probab=11
                tempp = random.randint(1,10)
                if tempp < probab:
                    rng = np.random.uniform(low = 0.3, high = 0.90)
                    init_pop[i][j] = rng* temp_arr[i][j]

        else:
            for j in range (chromosome_size):
                init_pop[i][j]=temp_arr[i][j]
        mutate(init_pop[i])

    get_fitness(init_pop,1)
    selection(init_pop)
    get_fitness(new_pop,2)
    finaltup=[]
    for i in range(pop_size):
        finaltup.append((fitness1[i],init_pop[i]))

    for i in range(pop_size):
        finaltup.append((fitness2[i],new_pop[i]))

    finaltup.sort(reverse=True , key=lambda x:x[0])
    finaltup.sort(reverse=True, key=lambda x:x[0]) #made change here

    for i in range(pop_size):
        new_init_pop[i]=finaltup[i][1]
        print(finaltup[i][0])
    ret=submit(key,list(new_init_pop[0]))
    tr ,va = get_errors(key,list(new_init_pop[0]))
    update = []
    for i in range(len(new_init_pop)):
        update.append(list(new_init_pop[i]))
    generations+=1
```

GENERATION 1:

EXPLANATION OF DIAGRAM:

- We took an initial population of size 10, which are represented as P1 to P10 .
- The corresponding fitness function value is the probability of each getting selected.
- Selection refers to the parents which got selected in n_th run as n_th row.
- Crossover has children arised when the parents got crossover.
- Mutation has the mutated children.

Since all these vectors are large and can't fit the width, we documented them below.

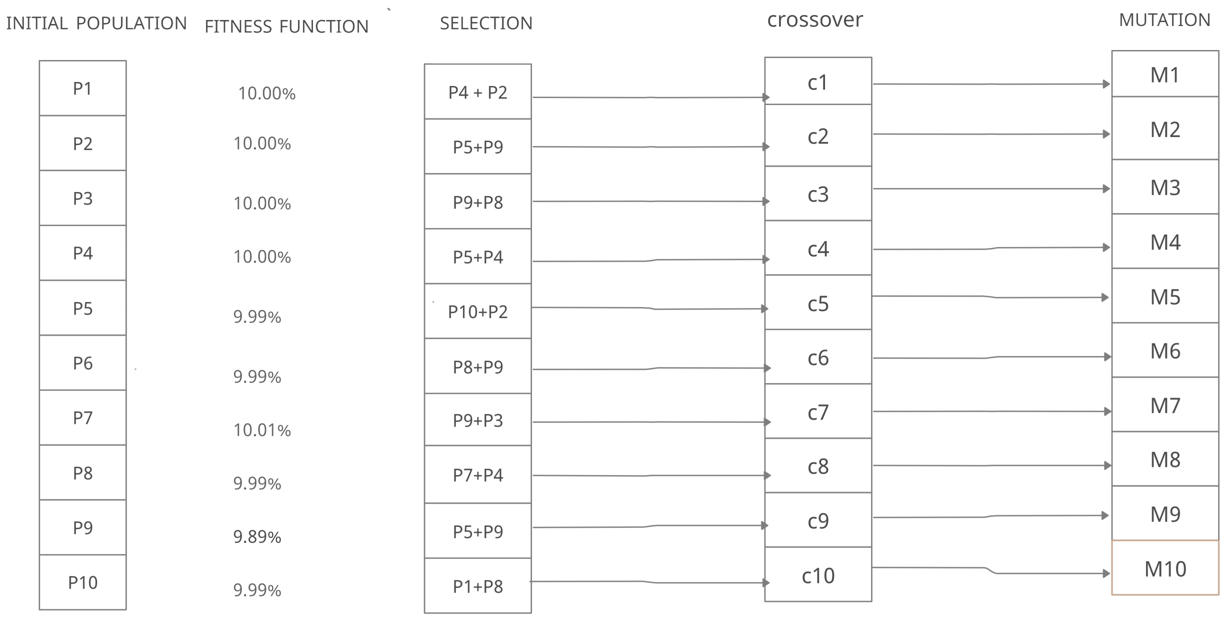


INDEX	population
P1	[0.00000000e+00 -8.58263092e-13 -1.83312975e-13 3.92122641e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46664817e-09 4.02140507e-10]
P2	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92103808e-11 -8.80325998e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46664817e-09 4.02140507e-10]
P3	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62382677e-05 -9.28162625e-07 -9.46664817e-09 4.02091438e-10]
P4	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92099448e-16 1.62370734e-05 -9.28162625e-07 -9.46664817e-09 4.02140507e-10]
P5	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92136468e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28162625e-07 -9.46664817e-09 4.02092953e-10]
P6	[0.00000000e+00 -8.58250702e-13 -1.83360980e-13 3.92173675e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28193952e-07 -9.46826489e-09 4.02140507e-10]
P7	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01322392e-16 2.92050263e-16 1.62370734e-05 -9.28162625e-07 -9.46664817e-09 4.02093556e-10]
P8	[0.00000000e+00 -8.58166496e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62361550e-05 -9.28148698e-07 -9.46664817e-09 4.02140507e-10]
P9	[0.00000000e+00 -8.58171786e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01469248e-16 2.92029940e-16 1.62370734e-05 -9.28162625e-07 -9.46793342e-09 4.02127443e-10]
P10	[0.00000000e+00 -7.56206795e-13 -9.37916682e-14 2.92221672e-11 -1.51459038e-10 -1.32767782e-15 3.32105849e-16 1.48210498e-05 -8.58797900e-07 -1.07149041e-08 5.04643778e-10]

index	After crossover
c1	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92099448e-16 1.62370734e-05 -9.27980588e-07 -9.46664817e-09 4.02140507e-10]
c2	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92173675e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28193952e-07 -9.46826489e-09 4.02140507e-10]
c3	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92099448e-16 1.62370734e-05 -9.28162625e-07 -9.46664817e-09 4.02140507e-10]
c4	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92099448e-16 1.62370734e-05 -9.28162625e-07 -9.46793342e-09 4.02127443e-10]
c5	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01322392e-16 2.92050263e-16 1.62370734e-05 -9.28162625e-07 -9.46664817e-09 4.02092953e-10]
c6	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62382677e-05 -9.28162625e-07 -9.46664817e-09 4.02091438e-10]
c7	[0.00000000e+00 -8.58171786e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01469248e-16 2.92029940e-16 1.62361550e-05 -9.28148698e-07 -9.46664817e-09 4.02140507e-10]
c8	[0.00000000e+00 -8.58166496e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62361550e-05 -9.28148698e-07 -9.46664817e-09 4.02091438e-10]
c9	[0.00000000e+00 -8.58263092e-13 -1.83312975e-13 3.92122641e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62361550e-05 -9.28148698e-07 -9.46664817e-09 4.02140507e-10]
c10	[0.00000000e+00 -8.58250702e-13 -1.83360980e-13 3.92173675e-11 -8.80325998e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46664817e-09 4.02140507e-10]

index	After mutation
m1	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92099448e-16 1.62340014e-05 -9.27980588e-07 -9.46664817e-09 4.02140507e-10]
m2	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92173675e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28193952e-07 -9.46826489e-09 4.02140507e-10]
m3	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92056883e-16 1.62370734e-05 -9.28162625e-07 -9.46713956e-09 4.02193686e-10]
m4	[0.00000000e+00 -8.58250702e-13 -1.83358931e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92099448e-16 1.62370734e-05 -9.28162625e-07 -9.46793342e-09 4.02127443e-10]
m5	[0.00000000e+00 -8.58124507e-13 -1.83379060e-13 3.92122641e-11 -8.80409147e-11 -6.01322392e-16 2.92050263e-16 1.62370734e-05 -9.28162625e-07 -9.46664817e-09 4.02064114e-10]
m6	[0.00000000e+00 -8.58250702e-13 -1.83325284e-13 3.92082807e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62382677e-05 -9.28162625e-07 -9.46621357e-09 4.02091438e-10]
m7	[0.00000000e+00 -8.58171786e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01469248e-16 2.91981543e-16 1.62361550e-05 -9.28132759e-07 -9.46664817e-09 4.02140507e-10]
m8	[0.00000000e+00 -8.58166496e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01254027e-16 2.92029940e-16 1.62361550e-05 -9.28148698e-07 -9.46773251e-09 4.02091438e-10]
m9	[0.00000000e+00 -8.58263092e-13 -1.83312975e-13 3.92122641e-11 -8.80325998e-11 -6.01254027e-16 2.92066274e-16 1.62361550e-05 -9.28148698e-07 -9.46664817e-09 4.02140507e-10]
m10	[0.00000000e+00 -8.58305531e-13 -1.83360980e-13 3.92173675e-11 -8.80439557e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46664817e-09 4.02145764e-10]

GENERATION 2:



index	population
p1	[0.00000000e+00 -8.58305531e-13 -1.83360980e-13 3.92173675e-11 -8.80439557e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46664817e-09 4.02145764e-10]
p2	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92103808e-11 -8.80325998e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46664817e-09 4.02140507e-10]
p3	[0.00000000e+00 -8.58263092e-13 -1.83312975e-13 3.92122641e-11 -8.80181661e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46664817e-09 4.02140507e-10]
p4	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46713956e-09 4.02193686e-10]
p5	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11

	-8.80436365e-11 -6.01254027e-16 2.92099448e-16 1.62323049e-05 -9.27980588e-07 -9.46777446e-09 4.02140507e-10]
p6	[0.00000000e+00 -8.58250702e-13 -1.83314621e-13 3.92082807e-11 -8.80187974e-11 -6.01254027e-16 2.92099448e-16 1.62370734e-05 -9.28166643e-07 -9.46721203e-09 4.02140507e-10]
p7	[0.00000000e+00 -8.58166496e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01148749e-16 2.92029940e-16 1.62361550e-05 -9.28020876e-07 -9.46537974e-09 4.02140507e-10]
p8	[0.00000000e+00 -8.58263092e-13 -1.83281016e-13 3.92070241e-11 -8.80325998e-11 -6.01254027e-16 2.92066274e-16 1.62361550e-05 -9.28181336e-07 -9.46664817e-09 4.02140507e-10]
p9	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01469248e-16 2.91969747e-16 1.62361550e-05 -9.28132759e-07 -9.46709086e-09 4.02140507e-10]
p10	[0.00000000e+00 -8.58250702e-13 -1.83343334e-13 3.92082807e-11 -8.80426534e-11 -6.01254027e-16 2.92029940e-16 1.62410644e-05 -9.28109421e-07 -9.46621357e-09 4.02091438e-10]

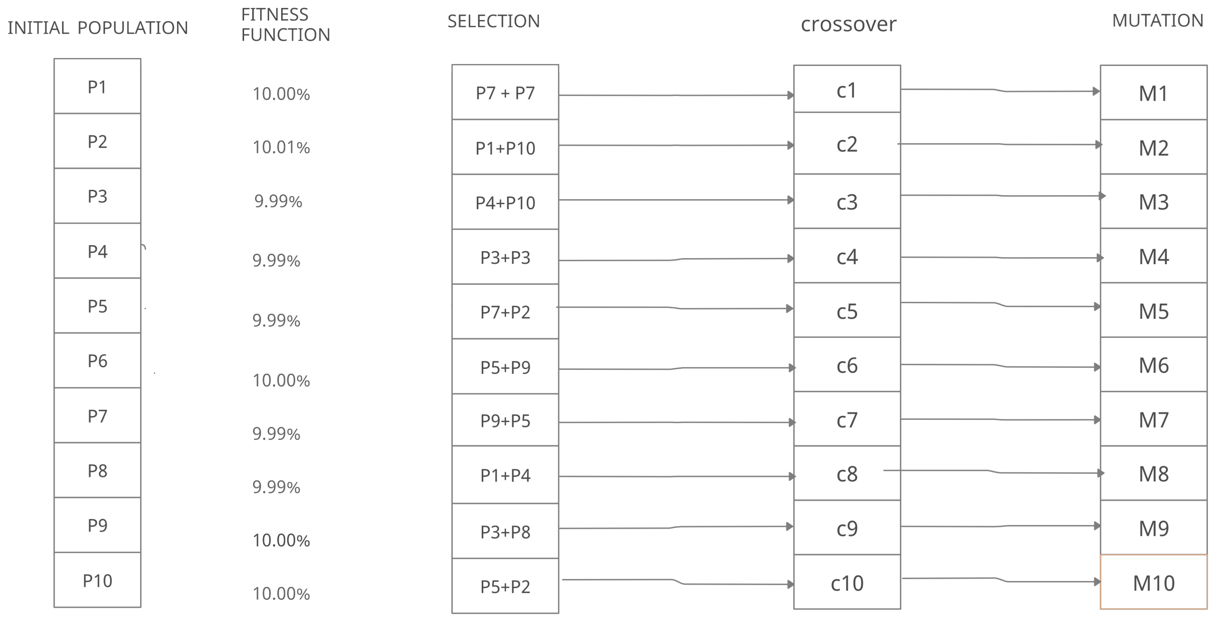
index	After crossover
c1	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46713956e-09 4.02193686e-10]
c2	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01469248e-16 2.91969747e-16 1.62361550e-05 -9.28132759e-07 -9.46709086e-09 4.02140507e-10]
c3	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01469248e-16 2.91969747e-16 1.62361550e-05 -9.28132759e-07 -9.46709086e-09 4.02140507e-10]
c4	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46713956e-09 4.02193686e-10]

c5	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92103808e-11 -8.80325998e-11 -6.01342450e-16 2.92029940e-16 1.62410644e-05 -9.28109421e-07 -9.46621357e-09 4.02091438e-10]
c6	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01469248e-16 2.92066274e-16 1.62361550e-05 -9.28181336e-07 -9.46664817e-09 4.02140507e-10]
c7	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80181661e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46664817e-09 4.02140507e-10]
c8	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46537974e-09 4.02140507e-10]
c9	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80436365e-11 -6.01254027e-16 2.92099448e-16 1.62323049e-05 -9.27980588e-07 -9.46777446e-09 4.02140507e-10]
c10	[0.00000000e+00 -8.58263092e-13 -1.83281016e-13 3.92070241e-11 -8.80325998e-11 -6.01254027e-16 2.92066274e-16 1.62361550e-05 -9.28181336e-07 -9.46664817e-09 4.02140507e-10]

index	After mutation
m1	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01242803e-16 2.92056883e-16 1.62393332e-05 -9.28330372e-07 -9.46713956e-09 4.02193686e-10]
m2	[0.00000000e+00 -8.58250702e-13 -1.83329568e-13 3.92122641e-11 -8.80240451e-11 -6.01566450e-16 2.91969747e-16 1.62361550e-05 -9.28132759e-07 -9.46709086e-09 4.02140507e-10]
m3	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01446473e-16 2.91969747e-16 1.62361550e-05 -9.28243937e-07 -9.46876836e-09 4.02140507e-10]
m4	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01246231e-16 2.92068653e-16 1.62393332e-05 -9.28162625e-07 -9.46713956e-09 4.02193686e-10]
m5	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92103808e-11

	-8.80325998e-11 -6.01342450e-16 2.92029940e-16 1.62410644e-05 -9.28109421e-07 -9.46621357e-09 4.02091438e-10]
m6	[0.00000000e+00 -8.58050924e-13 -1.83347451e-13 3.92122641e-11 -8.80325998e-11 -6.01469248e-16 2.92082473e-16 1.62361550e-05 -9.28181336e-07 -9.46664817e-09 4.02140507e-10]
m7	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80181661e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46534276e-09 4.02140507e-10]
m8	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92043353e-11 -8.80325998e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46537974e-09 4.02133684e-10]
m9	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80436365e-11 -6.01254027e-16 2.92099448e-16 1.62355356e-05 -9.27980588e-07 -9.46777446e-09 4.02140507e-10]
m10	[0.00000000e+00 -8.58263092e-13 -1.83281016e-13 3.92070241e-11 -8.80477025e-11 -6.01208348e-16 2.92032790e-16 1.62361550e-05 -9.28181336e-07 -9.46664817e-09 4.02140507e-10]

GENERATION 3:



index	population
p1	[0.00000000e+00 -8.58129147e-13 -1.83350958e-13 3.92122641e-11 -8.80181661e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46534276e-09 4.02140507e-10]
p2	[0.00000000e+00 -8.58166496e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01148749e-16 2.92029940e-16 1.62361550e-05 -9.28020876e-07 -9.46537974e-09 4.02140507e-10]
p3	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01246231e-16 2.92068653e-16 1.62393332e-05 -9.28162625e-07 -9.46678955e-09 4.02183429e-10]
p4	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46713956e-09 4.02170595e-10]
p5	[0.00000000e+00 -8.58305531e-13 -1.83360980e-13 3.92173675e-11 -8.80439557e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46664817e-09 4.02145764e-10]

p6	[0.00000000e+00 -8.58337545e-13 -1.83333358e-13 3.92103808e-11 -8.80325998e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27926807e-07 -9.46664817e-09 4.02140507e-10]
p7	[0.00000000e+00 -8.58263092e-13 -1.83312975e-13 3.92122641e-11 -8.80181661e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46664817e-09 4.02140507e-10]
p8	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92043353e-11 -8.80331352e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46443591e-09 4.02133684e-10]
p9	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80436365e-11 -6.01194521e-16 2.92145331e-16 1.62355356e-05 -9.27980588e-07 -9.46777446e-09 4.02215214e-10]
p10	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01302938e-16 2.92056883e-16 1.62393332e-05 -9.28330372e-07 -9.46713956e-09 4.02259204e-10]

index	After crossover
c1	[0.00000000e+00 -8.58263092e-13 -1.83312975e-13 3.92122641e-11 -8.80181661e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46664817e-09 4.02140507e-10]
c2	[0.00000000e+00 -8.58250702e-13 -1.83350958e-13 3.92122641e-11 -8.80181661e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46534276e-09 4.02140507e-10]
c3	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01302938e-16 2.92056883e-16 1.62393332e-05 -9.28330372e-07 -9.46713956e-09 4.02259204e-10]
c4	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01246231e-16 2.92068653e-16 1.62393332e-05 -9.28162625e-07 -9.46678955e-09 4.02183429e-10]
c5	[0.00000000e+00 -8.58166496e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01148749e-16 2.92029940e-16 1.62361550e-05 -9.28020876e-07 -9.46537974e-09 4.02140507e-10]
c6	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11

	-8.80436365e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46664817e-09 4.02145764e-10]
c7	[0.00000000e+00 -8.58305531e-13 -1.83360980e-13 3.92173675e-11 -8.80439557e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46777446e-09 4.02215214e-10]
c8	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46713956e-09 4.02170595e-10]
c9	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92043353e-11 -8.80331352e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46443591e-09 4.02183429e-10]
c10	[0.00000000e+00 -8.58166496e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01148749e-16 2.92029940e-16 1.62361550e-05 -9.28020876e-07 -9.46537974e-09 4.02140507e-10]

index	After mutation
m1	[0.00000000e+00 -8.58124045e-13 -1.83308446e-13 3.92122641e-11 -8.80181661e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46664817e-09 4.02140507e-10]
m2	[0.00000000e+00 -8.58250702e-13 -1.83314560e-13 3.92122641e-11 -8.80151434e-11 -6.01254027e-16 2.92029940e-16 1.62370734e-05 -9.28016446e-07 -9.46534276e-09 4.02140507e-10]
m3	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01273792e-16 2.92100871e-16 1.62393332e-05 -9.28330372e-07 -9.46713956e-09 4.02259204e-10]
m4	[0.00000000e+00 -8.58250702e-13 -1.83347451e-13 3.92082807e-11 -8.80325998e-11 -6.01246231e-16 2.92068653e-16 1.62393332e-05 -9.28162625e-07 -9.46678955e-09 4.02259633e-10]
m5	[0.00000000e+00 -8.58251157e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01148749e-16 2.92029940e-16 1.62361550e-05 -9.28020876e-07 -9.46537974e-09 4.02140507e-10]

m6	[0.00000000e+00 -8.58129147e-13 -1.83347451e-13 3.92122641e-11 -8.80485565e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.28131674e-07 -9.46664817e-09 4.02145764e-10]
m7	[0.00000000e+00 -8.58305531e-13 -1.83378492e-13 3.92099956e-11 -8.80439557e-11 -6.01342450e-16 2.92021597e-16 1.62370734e-05 -9.27980588e-07 -9.46777446e-09 4.02215214e-10]
m8	[0.00000000e+00 -8.58100161e-13 -1.83347451e-13 3.92007850e-11 -8.80325998e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46713956e-09 4.02170595e-10]
m9	[0.00000000e+00 -8.58100161e-13 -1.83334054e-13 3.92043353e-11 -8.80331352e-11 -6.01246231e-16 2.92056883e-16 1.62393332e-05 -9.28162625e-07 -9.46266968e-09 4.02183429e-10]
m10	[0.00000000e+00 -8.58223655e-13 -1.83367371e-13 3.92122641e-11 -8.80325998e-11 -6.01148749e-16 2.92004660e-16 1.62373814e-05 -9.28020876e-07 -9.46499396e-09 4.02140507e-10]

FITNESS FUNCTION

Fitness function determines how fit an individual is. Lower the error more fit the individual is.Hence fitness is inversely proportional to MSE.We defined fitness as:

fitness= 1/(train_factor*trainerror + validationerror)

where trainerror and validationerror are mse of train and validation datasets for given vector. **train_factor** is used for controlling the effect(weightage) of trainerr and validationerr on fitness function.

The probability that an individual will be selected for reproduction is based on its fitness score.

PROBABILITY

Since the probability of a individual getting picked up is directly proportional to fitness, we have calculated probabilities for each individual.Probability of the i_{th} individual is defined as follows:

probability(i) = fitness(i)/total_fitness

where fitness(i) is fitness of the i_{th} individual and total fitness is sum of all fitness of all individuals in the population.

In [ ]:

```
def get_fitness(arr, ind):
        #getting errors on the data
    fitness=[0 for i in range(chromosome_size)]
    j=0
    for chromoso in arr:
        testerr,validerr=get_errors(key,list(chromoso))
        print(testerr,validerr)
        fitness[j]=1/(train_factor*testerr+validerr)
        j+=1

    if ind==1:
        for m in range(chromosome_size):
            fitness1[m]=fitness[m]
            #print(fitness)
    else:
        for m in range(chromosome_size):
            fitness2[m]=fitness[m]

    #calculate probabilities
    sum_fit=np.sum(fitness)
    for k in range(pop_size):
        probability[k]=fitness[k]/sum_fit
```

SELECTION FUNCTION

The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation. Two pairs of individuals (parents) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

From the above calculated probabilites we chose two parents using **np.random.choice** and we have done crossover between them, and added child to new population (initially new population is empty).We have repeated this process **pop_size** no. of times to generate complete new population of same size.

In [ ]:
```
def selection(arr:np.ndarray):
    for i in range(pop_size):
        parent1ind=np.random.choice(pop_size,p=probability)
        parent1=arr[parent1ind]
        parent2ind=np.random.choice(pop_size,p=probability)
        parent2=arr[parent2ind]

        #printing parents after selection :
        print("printing parents after selection :")
        print("parent1:")
        print(parent1)
        print("parent2:")
        print(parent2)

        new_pop[i]=crossover(parent1,parent2)
```

CROSSOVER FUNCTION

We chose a point randomly between 1 and 10 (since length is 11). All the genes in the child before that point are from parent1 and after that are from parent2. Hence new child is generated from crossover at that point(mid).

In [ ]:
```
def crossover(parent1,parent2):
    mid=random.randint(1,chromosome_size-1)
    child=np.ones(chromosome_size)
    for i in range(0,mid+1):
        child[i]=parent1[i]
    for j in range(mid,chromosome_size):
        child[j]=parent2[j]
    print(child)
    return child
```

MUTATION FUNCTION

Some of the genes in the child can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped.

We have defined the probability as mutation_probability.And we changed the value as follows.

1. We took a variable whose value is randomly choosen from [-abs(actualvalue)/5000, abs(actualvalue)/5000], abs stands for absolute. 2.We added that variable to actualvalue if a randomly generated value between (0,1) called k is less than mutation probability.

   We put conditions to ensure that genes value doesn't go out of bounds.(It was given that gene value must be from (-10,10) for case of simplicity)

In [ ]:
```
def mutate(chromosome:np.ndarray):
    mutation_probability = 0.2
    for i in range(chromosome_size):
        l = abs(chromosome[i])/5000
        r = -1*l
        temp = random.uniform(r,l)
        k = random.uniform(0,1)
        if k <= mutation_probability:
            chromosome[i]+=temp
        if chromosome[i]>10:
            chromosome[i]=10
        elif chromosome[i]<-10:
            chromosome[i]=-10
```

HYPERPARAMETERS

- key = "F1hP7PePw62PZ8iABBDNb2zqmkX7nbVrz8328hJ3ySZLvyQ88o"
- pop_size = 10
- choromosome_size = 11
- train_factor = 0.5
- mutation_probability = 0.2
- mid = random.randint(1,chromosome_size-1)
- total_generations = 2
- probab = 11
- rng = np.random.uniform(low = 0.3, high = 0.90)

key : secret key, team name :Damn, team number:21

pop_size: population size of the algorithm, we took as 4 initially , we were getting same vectors for few generations , we decided to try with 10, this gave better results and felt like ideal numbers not using too many of server requests.

choromosome_size: no. of features in the given initial vector.

train_factor: for initial given vector train error was in range 1.3e10 and valid error was in range 3.6e11. To adjust the affect of these values on fitness function we have used a parameter train_factor.Value 0.5 was giving the best possible relation.

mutation_probability: the probability of each gene/feature mutating is mutation_probability.We didn't want many genes to get mutated, we tried out with few values like 0.4, 0.6. value 0.2 gave stable vectors and improvised errors.

mid: mid is the crossover point.All the genes before mid are from parent1 and after are from parent2.We have tried with few values like 4,5.Random values gave better results.

total_generations: no.of generations we are running, we controlled this as we didn't have many requests.We rredirected output to file and took as input for next generation.

probab: we initialised our initial population to zeros, then we decided to change few values from it.probabilityof each gene changing is porbab.We tried with 8 it was giving decent results, we tried with 11.i.e, all genes are to be changed, we tried this for more diversity, this seemed to give better results as compared to 8.

rng: the factor which we multiplied the initially given gene with, so as to generate initial population(population for first run).

STATISTICS :

pop_size	Iterations to converge	Mating Pool size	Binary crossover	Mutation probability(of children)	Initpop from
4	13	4	No	0.6	Mutate on given vector
4	11	4	No	0.5	Mutate on given vector
10	29	10	No	0.7	Mutate on given vector
10	21	10	No	0.5	Mutate on Zeros p=0.7
10	15	10	Yes	0.5	Mutate on Zeros p=0.7
10	21	10	Yes	0.6	Mutate on Given vector
10	17	10	No	0.4	Mutate on Given vector
10	26	10	No	0.4	Mutate on zeros p=0.8
10	23	10	No	0.3	Mutate on Zeros p=1

HEURISTICS

While constructing the Genetic Algorithm, the heuristics that we applied include :

- initial population generation : We took the initial given vector as the whole population and applied few mutations on it with high probability.This didnt improve error much.Running for 6-7 generations still gave very little difference in error, which implies we were struck in a local minima. So we tried initialising to zeros and with some probability change few values and then mutating few again.We did this as to diversify initial population which would help us to come out of local minima.
- fitness function : for the fitness we used fitness=1/(train_factor*trainerr+validerr), tried with different combination with fitness expression(variation in train_factor) and 0.5 one giving the best relation.
- crossover function : We have tried binary crossover. We chose a point randomly , for child 1 all genes before that point are from parent 1 and all genes after that are from parent 2, and for child2 all genes before that point are from parent2 and after that are from parent 1.But this was giving high MSE than normal crossover.Therefore we sticked to normal crossover.
- mutation : We have used the below mentioned function initially. 1) We tried subtracting the mean of genes over population. 2) Tried replacing it with other chromosome

In [ ]:
```
def mutate(chromosome:np.ndarray):
    mean=np.mean(init_pop,axis=0)
    for i in range(chromosome_size):
        #chromosome[i]=np.random.choice(chromosome) bad idea
        temp=np.random.choice(mean)
        #temp=temp+random.uniform(-0.005,0.005)
        #chromosome[i]=temp
        if mod(temp)<mod(chromosome[i]):
            chromosome[i]-=temp
```

Both of these performed poorly on the data.Then we have tried current mutation function , with varying parameters, the finalised parameters were giving better results.

TRAIN AND VALIDATION ERRORS

The train error we were able to achieve was around 2e11 (269197209518.2254) and validation error was around 6e11 (607848571571.9812) .Most of the vectors that gave better position on leader board has a sum of 10e11.Below listed are reasons why we think our algorithm would perform better on unseen data.

- We have taken diverse initial population.We didnt completely startout random, we have taken overfit vector and from that we tried to reach the best fit vector.

- We modified that algorithm such that it performs well on train error, validation error and as well as better position on leader board.That means we are't overfitting out vector on train data.
- We aren't restricting pool size of parents. We are choosing best parents based on probability(fitness) and performing crossover on them.At the end we are choosing best individuals from set of parents and children.That means we are choosing best individuals which performed well on the data.