Randomized Optimization

Georgia Institute of Technology CS7641: Machine Learning Assignment 2

Weixu Chen

GT ID: wchen607

**Abstract**

This paper summarizes the works to explore random search. The implementation includes the four requested random search algorithms and they are 1) randomized hill climbing (RHC), 2) simulated annealing (SA), 3) genetic algorithm (GA) and the last one 4) MIMIC. The report includes two major portions, the first part which is “the problems you give us” including three problems and they are Travelling Salesman Problem (TSP), Continuous Peaks Problem (CPP), and Knapsack Problem (KP). The second part which is “the problems given to you” is to use RHC, SA, and GA to find good weights for a neural networks.

**Briefs of the four randomized optimization algorithms**

Randomized hill climbing (RHC) begins a random rough solution to the target problem and loops to search a better and better solution by increasing changes until no further improvement could be found. As the name suggested, the algorithm does hill climbing iteratively, every time begins with a random initial guess, traveling to increase fitness on each iteration. Usually, RHC needs little memory and good at the problems which is convex.

Simulated annealing (SA) is an interesting algorithm. It simulates metal annealing process from cooling to solidifying into the low energy crystalline structure. The algorithm models the characterizes of the process, that is not always accepting a fitness function and it gives a chance to avoid stucking the local optimum. One of key parameters in SA is the “temperature” T. The higher T, the model works towards random walk while decreasing T the model works towards hill climbing.

Genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. The advantage of GA is that it randomly discovers the big population and find optimal solutions which other randomized algorithm may not be able to. Another advantage of GA is that the search does not need to rely on the derivative of fitness function. This is difference as the local search family algorithms which need the derivatives of fitness functions. However, the disadvantage is the GA is not good at very complex problem more researches will be needed.

MIMIC is a novel algorithm which converges faster and reliably than above algorithms. It performs searching by successively approximating the conditional distribution of the inputs given a bound on the cost function. Another characters of MIMIC is that it directly passes information about the cost function from the early stages to the later stages of the search. MIMIC also explore the structure about optima by computing 2nd order statistics and sampling from a distribution consistent with those statistics.

**Part I: The Problems You Give Us**

The problems I have chosen and analyzed are Travelling Salesman Problem (TSP), Continuous Peaks Problem (CPP), and Knapsack Problem (KP). The analysis is using ABAGAIL package.

**Continuous Peaks Problem (CPP)**

CPP is a problem to search for global optima however there are existing may local optima. The importance to overcome the local optima is the key for this problem.

My CPP experiment setups are as following:

1. Number of local optima N = 80
2. Number of iterations iters = 5000
3. Number of test runs = 10
4. Randomized optimization setups

RHC:

[ContinuousPeaksEvaluationFunction, DiscreteUniformDistribution, DiscreteChangeOneNeighbor] 🡪 HillClimbingProblem 🡪 RandomizedHillClimbing

SA:

[ContinuousPeaksEvaluationFunction, DiscreteUniformDistribution, DiscreteChangeOneNeighbor] 🡪 HillClimbingProblem 🡪 SimulatedAnnealing

Temperature, t = 1E11

cooling = {0.15, 0.55, 0.85}

GA:

[ContinuousPeaksEvaluationFunction, DiscreteUniformDistribution, DiscreteChangeOneMutation, SingleCrossOver] 🡪 GeneticAlgorithmProblem 🡪 StandardGeneticAlgorithm

PopulationSize = 200

Mate = 100

Mutate = {10, 20, 30}

MIMIC:

[ContinuousPeaksEvaluationFunction, DiscreteUniformDistribution, DiscreteDependencyTree]

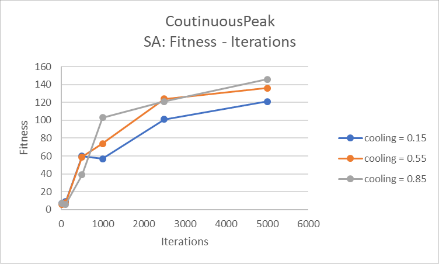
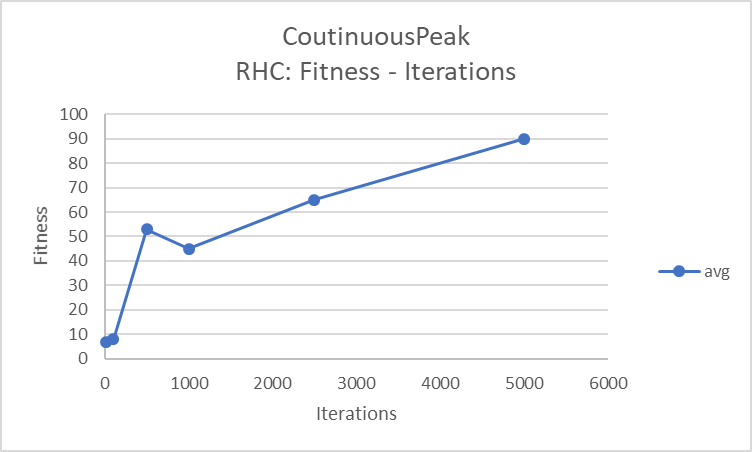
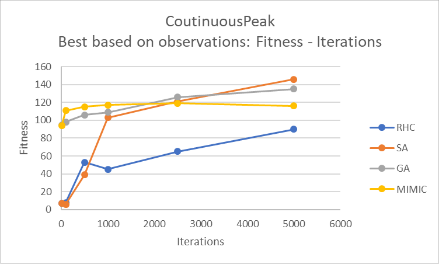
🡪 ProbabilisticOptimizationProblem 🡪 MIMIC

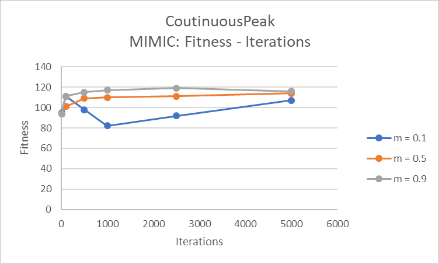
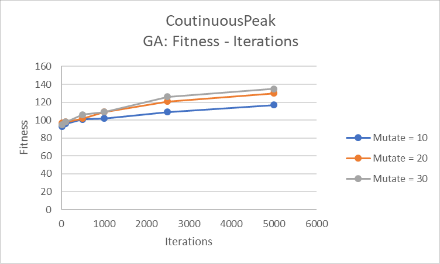
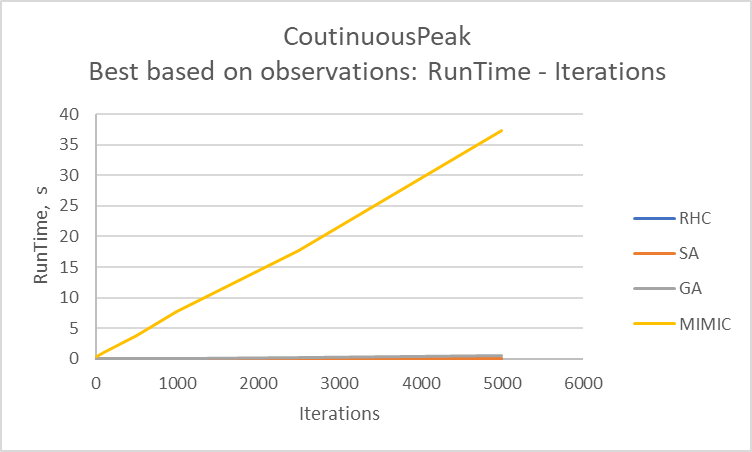
Samples = 200

Keep = 100

m = {0.1, 0.5, 0.9}

The outputs of experiment are shown in the following plots. As we can see the parameters have the influence of performance. A more systemic parameter grid search should be performed to find the optimal parameters for the problem.

To easy compare the result, I have picked the best curves from the experiment and plot them together. As we can see that SA gives us the best fitness for this CPP, and RHC gives the worst. MIMIC can provide the relative best fitness when less iterations being needed. But both MIMIC and GA flat even increasing iterations. However, the payback is that MIMIC needs more significant running time to perform computations. And it looks like it is linear time increment per iteration increment.

**Traveling Salesman Problem (TSP)**

Based on Wikipedia, the travelling salesman problem is "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" The problem is a NP-hard problem. The goal of TSP is that we can find the shortest possible path by visiting each vertex only once.

My TSP experiment setups are as following:

1. Number of local optima N = 30
2. Number of iterations iters = 2500
3. Number of test runs = 10
4. Randomized optimization setups

RHC:

[TravelingSalesmanRouteEvaluationFunction, DiscretePermutationDistribution, SwapNeighbor] 🡪 HillClimbingProblem 🡪 RandomizedHillClimbing

SA:

[TravelingSalesmanRouteEvaluationFunction, DiscretePermutationDistribution, SwapNeighbor] 🡪 HillClimbingProblem 🡪 SimulatedAnnealing

Temperature, t = 1E11

cooling = {0.15, 0.55, 0.85}

GA:

[TravelingSalesmanRouteEvaluationFunction, DiscretePermutationDistribution, SwapMutation, TravelingSalesmanCrossOver] 🡪 GeneticAlgorithmProblem 🡪 StandardGeneticAlgorithm

PopulationSize = 100

Mate = 50

Mutate = {10, 20, 30}

MIMIC:

[TravelingSalesmanRouteEvaluationFunction, DiscreteUniformDistribution,

DiscreteDependencyTree] 🡪 ProbabilisticOptimizationProblem 🡪 MIMIC

Samples = 100

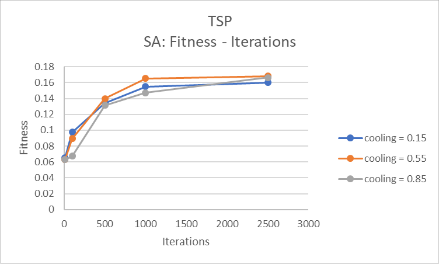
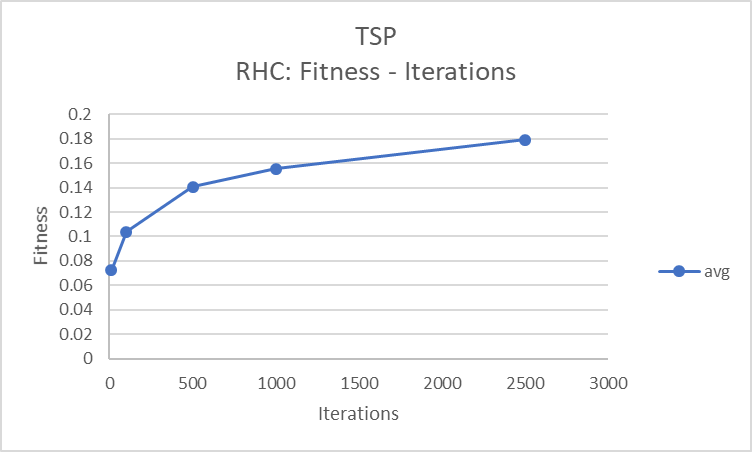
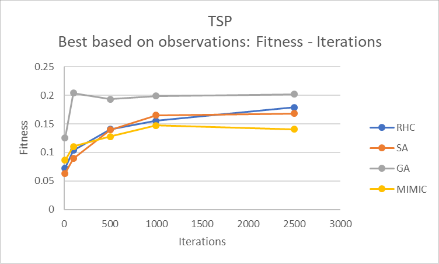
Keep = 50

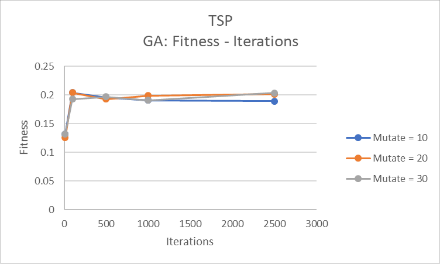
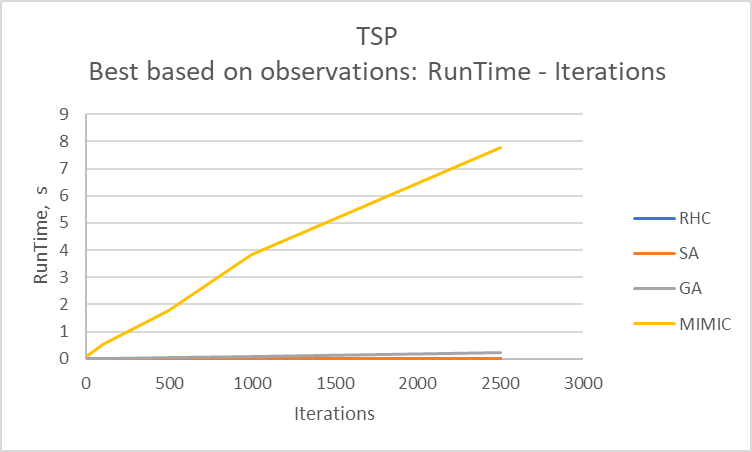
m = {0.1, 0.5, 0.9}

The outputs of experiment are shown in the following plots.

Same as above problem, I have picked the best curves from the experiment and plot them together.

It is interesting to see that GA gives us the best fitness for this TSP, and MIMIC gives the worst fitness.

Let us look at the run time, again MIMIC needs longer time to perform computations comparing with other three algorithms. Still it looks like it is linear time increment per iteration increment.  

**Knapsack Problem (KP)**

The Knapsack problem is defined as given a set of items, each having a weight and a value, the goal is to know the number of each item to include in a bag so that the total weight is less than or equal to a given limit and the total value is as large as possible.

My KP experiment setups are as following:

1. Number of items = 30
2. Number of copies each = 3
3. Maximum weight for a single element = 50
4. Maximum volume for a single element = 60
5. Number of iterations iters = 2500
6. Number of test runs = 10
7. Randomized optimization setups

RHC:

[KnapsackEvaluationFunction, DiscreteUniformDistribution, DiscreteChangeOneNeighbor] 🡪 HillClimbingProblem 🡪 RandomizedHillClimbing

SA:

[KnapsackEvaluationFunction, DiscreteUniformDistribution, DiscreteChangeOneNeighbor] 🡪 HillClimbingProblem 🡪 SimulatedAnnealing

Temperature, t = 1E11

cooling = {0.15, 0.55, 0.85}

GA:

[KnapsackEvaluationFunction, DiscreteUniformDistribution, DiscreteChangeOneMutation, UniformCrossOver] 🡪 GeneticAlgorithmProblem 🡪 StandardGeneticAlgorithm

PopulationSize = 100

Mate = 50

Mutate = {10, 20, 30}

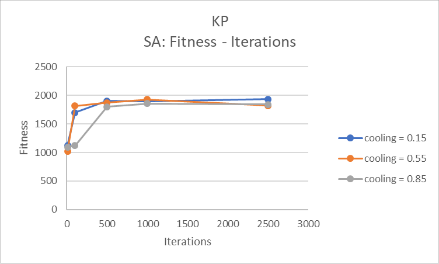
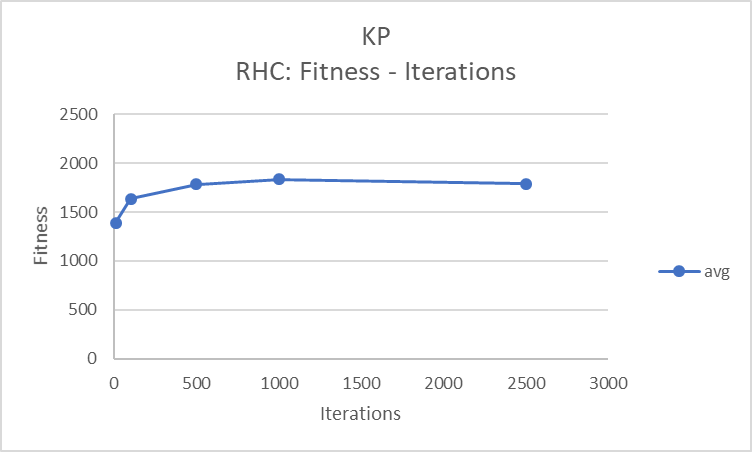
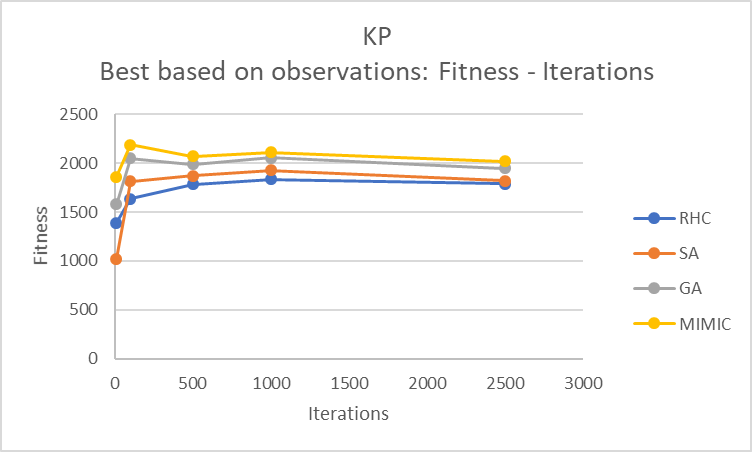
MIMIC:

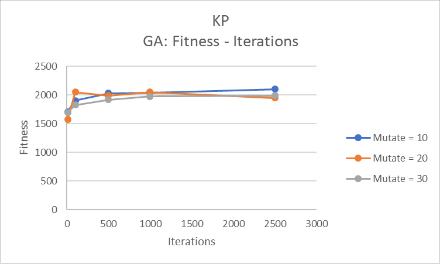
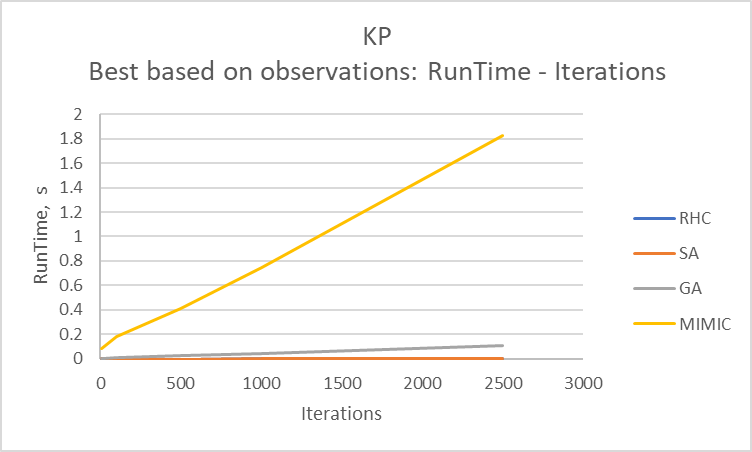
[KnapsackEvaluationFunction, DiscreteUniformDistribution, DiscreteDependencyTree] 🡪 ProbabilisticOptimizationProblem 🡪 MIMIC

Samples = 100

Keep = 50

m = {0.1, 0.5, 0.9}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Fitness | | | |
|  | RHC | SA | GA | MIMIC |
| CPP |  | best |  |  |
| TSP |  |  | best |  |
| KP |  |  |  | best |
|  | Fast | Fast | Medium | Slow |
|  | Run Time | | | |

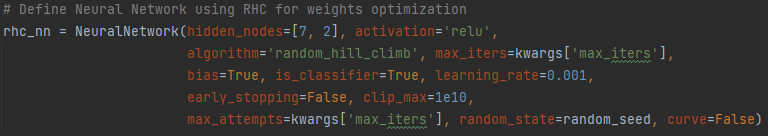
**Conclusion**

As we can observed from the above experimental outputs of best fitness, CPP has been performed best by using SA, TSP has been performed best by using GA and KP by using MIMIC. Except RHC, other three algorithm would need to select parameters through some tuning processes. And MIMIC is the most time-consuming algorithm. My observations have been tabulated in the following.

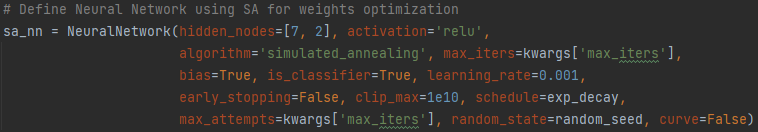
**Part II: The Problems Given to You**

The goal of this part is using the first three algorithms which are 1) randomized hill climbing (RHC), 2) simulated annealing (SA), 3) genetic algorithm (GA) to find weights of the neural network (NN) for the wine quality problem solved in #1 assignment. Same as #1 assignment, a cut-off of 6 was chosen. Values higher than 6 were relabeled as ‘1’/good, and lower as ‘0’/bad. The dataset has 66.5% ‘1’s examples and 33.5 ‘0’s examples. In #1 assignment based on my analysis, the setup for wine quality problem was that the split data percentage was set as 70% training data and 30% testing data. The NN was set up with 2 hidden layers which include 7 and 2 nodes, activation function was ReLU and learning rate was 0.001. In order to keep the consistent structure, I am using the same neural network setup. The computation has been implemented by using mlrose package. The accuracy is still be used the measurement to look at the performance of each algorithm.

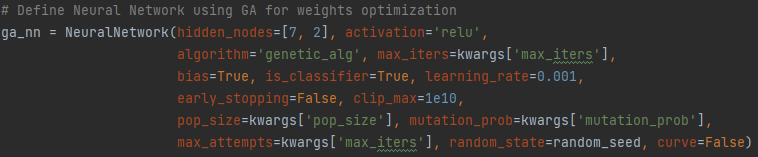
* The implementation of NN using RHC is set up as the following screenshot.



* The implementation of NN using SA is set up as the following screenshot.



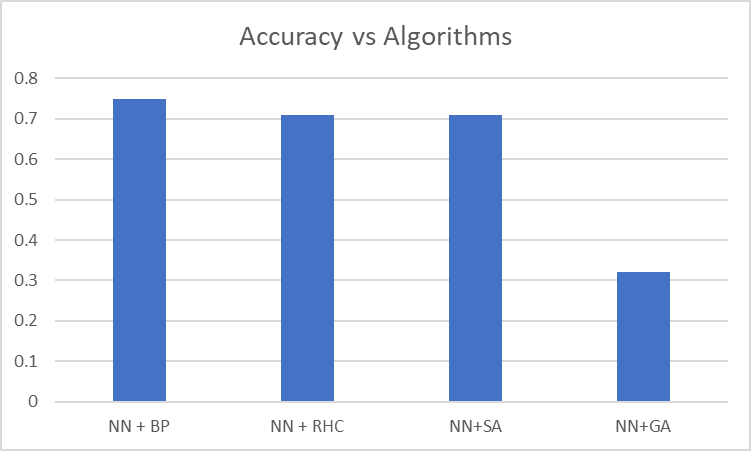
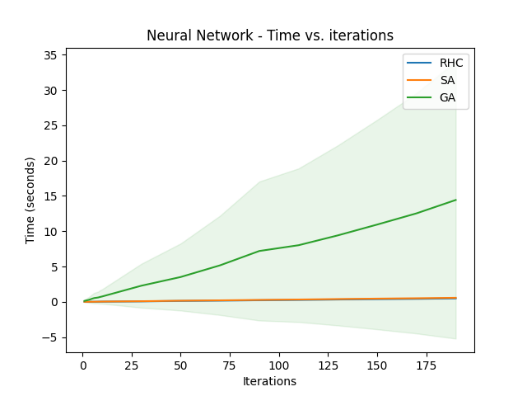
* The implementation of NN using GA is set up as the following screenshot.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NN -> Backpropagation (BP) | | | | |
|  | precision | recall | f1-score | support |
|  |  |  |  |  |
| 0 | 0.78 | 0.3 | 0.43 | 460 |
| 1 | 0.75 | 0.96 | 0.84 | 1010 |
|  |  |  |  |  |
| accuracy |  |  | 0.75 | 1470 |
| macro | 0.76 | 0.63 | 0.64 | 1470 |
| weighted | 0.76 | 0.75 | 0.71 | 1470 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NN -> Randomized optimization --> RHC | | | | |
|  | precision | recall | f1-score | support |
|  |  |  |  |  |
| 0 | 0.67 | 0.12 | 0.2 | 460 |
| 1 | 0.71 | 0.97 | 0.82 | 1010 |
|  |  |  |  |  |
| accuracy |  |  | 0.71 | 1470 |
| macro | 0.69 | 0.55 | 0.51 | 1470 |
| weighted | 0.7 | 0.71 | 0.63 | 1470 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NN -> Randomized optimization --> SA | | | | |
|  | precision | recall | f1-score | support |
|  |  |  |  |  |
| 0 | 0.67 | 0.12 | 0.2 | 460 |
| 1 | 0.71 | 0.97 | 0.82 | 1010 |
|  |  |  |  |  |
| accuracy |  |  | 0.71 | 1470 |
| macro | 0.69 | 0.55 | 0.51 | 1470 |
| weighted | 0.7 | 0.71 | 0.63 | 1470 |
| NN -> Randomized optimization --> GA | | | | |
|  | precision | recall | f1-score | support |
|  |  |  |  |  |
| 0 | 0.31 | 1 | 0.48 | 460 |
| 1 | 1 | 0.01 | 0.01 | 1010 |
|  |  |  |  |  |
| accuracy |  |  | 0.32 | 1470 |
| macro | 0.66 | 0.5 | 0.25 | 1470 |
| weighted | 0.79 | 0.32 | 0.16 | 1470 |



**Observations and Conclusion**

For the above reports of output results in this analysis, we can RHC, SA and GA are all worse than the built-in NN backpropagation for the measurement of accuracy stand point. Backpropagation gives us 75% accuracy, both RHC and SA give us 71% accuracy, and GA gives us the lowest accuracy only 32%. I have plotted the run time vs iterations as well, as we can see GA takes much longer time to run. Looking at F1 score even it was not targeted as a measurement, it still shows that the built-in NN provides the better learning than the other three randomized optimization approaches. It seems following same insights of accuracy measurement. So combined the accuracy and runtime, GA is not fitted for the wine quality problem. The original NN with back propagation is the good approach to perform learning.

**Reference**

**Reference for report:**

<https://medium.com/@duoduoyunnini/introduction-implementation-and-comparison-of-four-randomized-optimization-algorithms-fc4d96f9feea>

mlrose Documentation: <https://readthedocs.org/projects/mlrose/downloads/pdf/stable/>

Genetic algorithm: <https://en.wikipedia.org/wiki/Genetic_algorithm>

MIMIC: <https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf>

Travelling salesman problem: <https://en.wikipedia.org/wiki/Travelling_salesman_problem>

Knapsack problem: <https://en.wikipedia.org/wiki/Knapsack_problem>

**Reference for program:**

<https://github.com/danielcy715/CS7641-Machine-Learning/tree/master/Assignment2>

<https://github.com/kylewest520/CS-7641---Machine-Learning/tree/master/Assignment%202%20Randomized%20Optimization>

<https://github.com/ezerilli/Machine_Learning/tree/master/Randomized_Optimization>