CS7641 Assignment 2: Randomized Optimization

Jack Chai Jie Feng
jchai41@gatech.edu

NEURAL NETWORK

We experimented with 3 different randomized search algorithms to find weights for our neural network. The 3 different algorithms are randomized hill climbing, simulated annealing and genetic algorithm.

NETWORK PARAMETERS

The neural network architecture is a multi-layer perceptron with 2 hidden layers of 64 and 16 nodes respectively. The activation function for each hidden layer is ReLU and each fit is given a maximum of 1000 iterations. The loss function used is binary cross-entropy loss that is offered as part of mlrose_hiive neural network package.

DATASET

The chosen income dataset is a binary classification problem that was made available on 4/30/1996. It contains 32k rows and 14 features, with 50% of the features being categorical and the other 50% numerical. The goal of this dataset is to predict whether an individual makes >50k annual income or not, with an imbalance class of ~24.1% making the former label and ~75.9% for the latter. This dataset is also interesting because the one hot encoding of the categorical features will create many more columns.

EXPERIMENTS AND ANALYSIS

Since we experimented with Adam optimizer for the neural network weights in assignment 1 we can use it as a benchmark against these 3 random optimization algorithms. We will be looking at >50k/f1 score since the dataset has an imbalance label class. After adjusting the hidden layers and nodes to be the same as assignment 2, we conducted a hyper parameters search of learning rate and regularization term alpha (see appendix A for adam optimizer hyper parameters) and got a f1 score of 0.681956.

RANDOMIZED HILL CLIMBING (RHC)

We explored 2 hyper parameters for RHC through 3 fold cross validation grid search. The parameters are learning rate and number of restarts during hill climbing. The learning rate will determine the rate of updating the weights at each iteration and number of restarts will determine how many times will the algorithm perform random hill climbs.

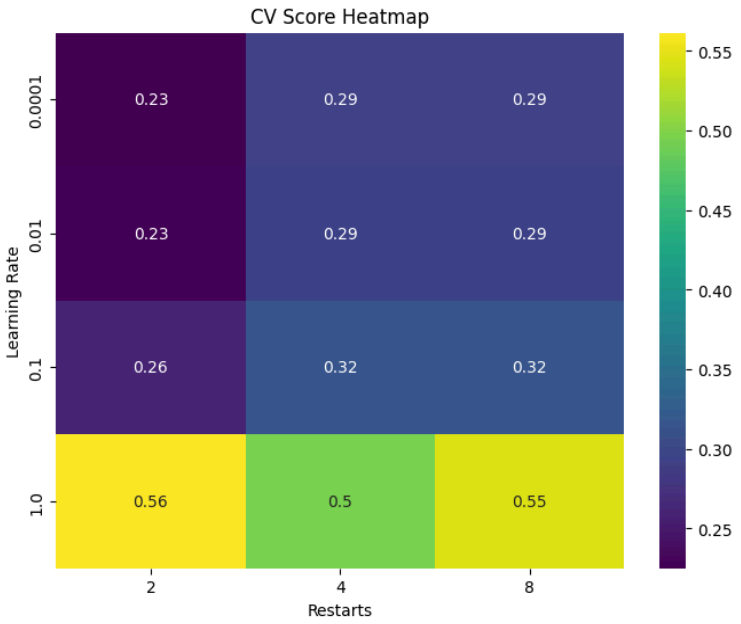


Figure 1.1.a: RHC CV f1 score heatmap

It is clear from figure 1.1.a that the algorithm performs best in terms of 3 CV f1 score when learning rate is 1.0 and number of restarts is 2.

Figure 1.1.b: Hyper parameter tuned loss curve

We can also observe that the loss curve is still decreasing and presumably will continue to decrease beyond 1000 iterations. This means that the algorithm will benefit with either a larger

learning rate or more iterations before reaching an optimum. Note that it is unsure whether the optimum is local or maxima.

SIMULATED ANNEALING (SA)

We explored 2 hyper parameters for SA through 3 fold cross validation grid search. The parameters are learning rate and initial temperature during simulated annealing. The learning rate will determine the rate of updating the weights at each iteration and initial temperature will determine how likely the algorithm will accept a worse solution in the name of exploration. The temperature decay schedule used is geometric decay with a rate of decay of 0.99.

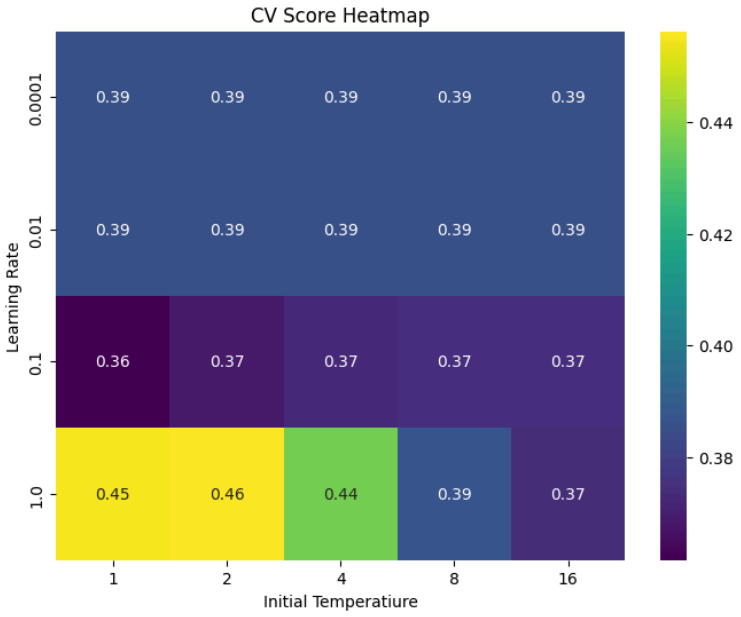


Figure 1.2.a: SA CV f1 score heatmap

It is clear from figure 1.2.a that the algorithm performs best in terms of CV f1 score when learning rate is 1.0 and initial temperature is 2.

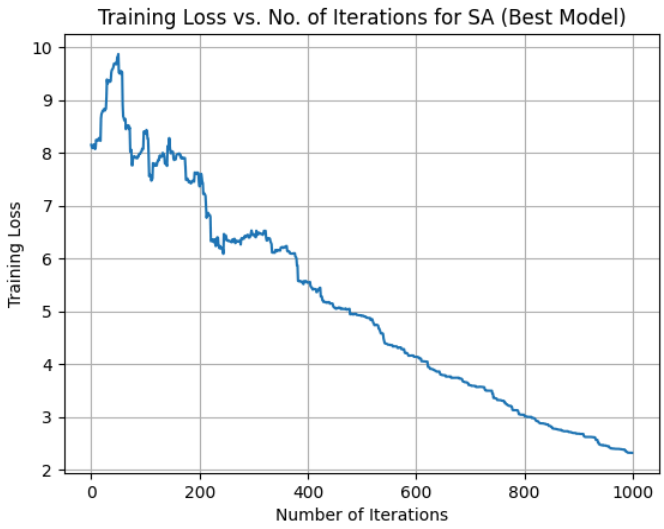


Figure 1.2.b: Hyper parameter tuned loss curve

Unlike randomized hill climbing, the loss curve of SA is not monotonically decreasing. This is due to the exploration phase of SA in the beginning where the algorithm is able to accept a worse solution with a higher probability (temperature). As it decays of iteration (temperature decreases), the SA algorithm will capitalize more on exploitation, hence we can see that after around iteration 400, the loss curve becomes monotonically decreasing.

GENETIC ALGORITHM (GA)

We explored 2 hyper parameters for GA through 3 fold cross validation grid search. The parameters are learning rate and population size. The learning rate will determine the rate of updating the weights at each iteration and the population size determines the number of different mutated weights at each iteration. Each set of weights in the population set is generated by a mixture of retention from previous iteration and crossover mutation of top weights from previous iteration. The mutation probability is tunable but in this experiment, we set it to 0.1 for all experiments.

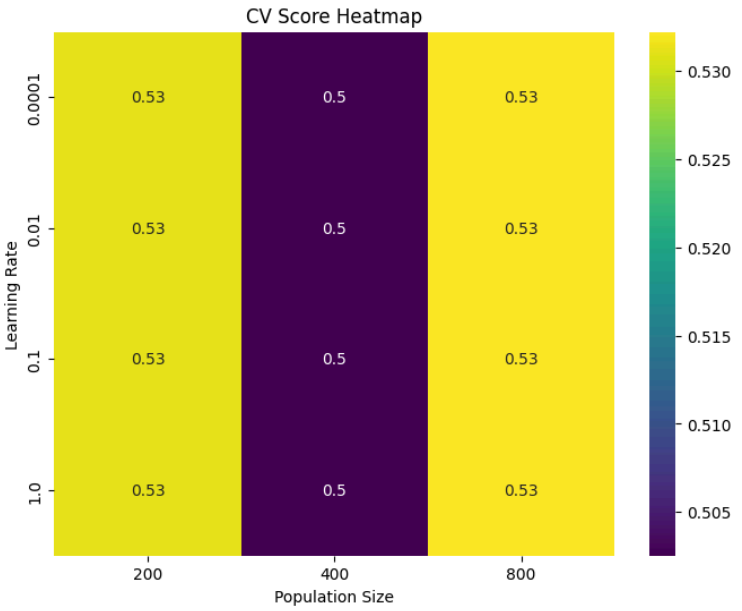


Figure 1.3.a: GA CV f1 score heatmap

From figure 1.3.a, the learning rate did not have any impact on CV f1 scores. This means that there are sufficient iterations for the algorithm to reach the optimum. The heatmap only shows up to 2 decimal places but based on the color, we can see that the population size of 800 has a slightly better result.

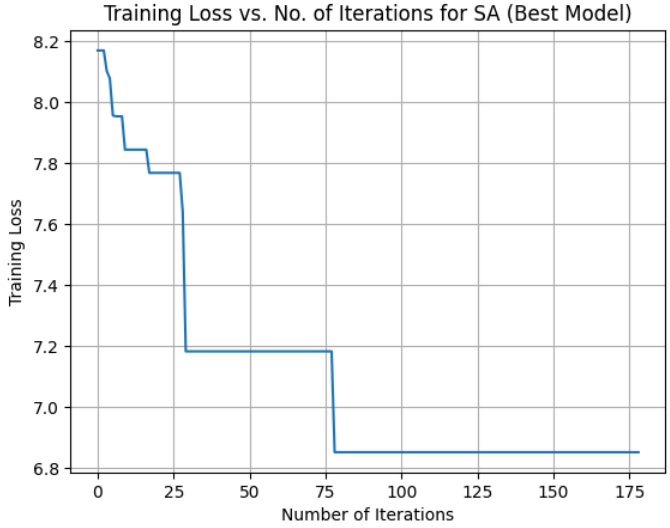


Figure 1.3.b: Hyper parameter tuned loss curve

As confirmed in the loss curve from figure 1.3.b, the model quickly converges on the optimum at around iteration 75, way below the max iteration we set at 1000.

ALGORITHM COMPARISON

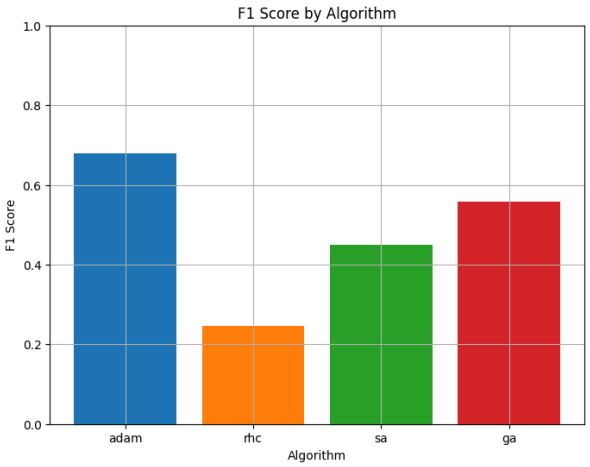


Figure 2.1.a: Algorithm f1 score comparison

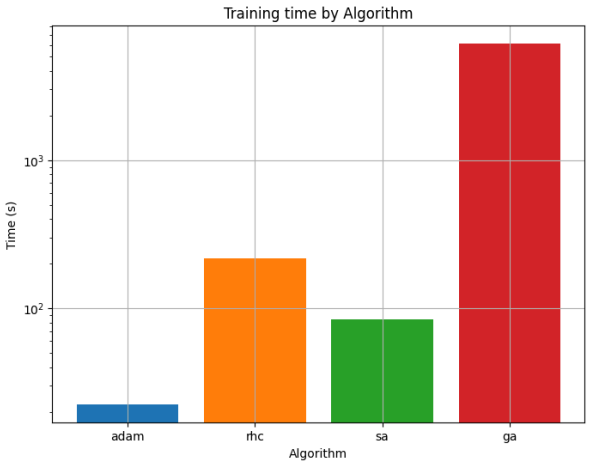


Figure 2.1.a: Algorithm search time comparison

As mentioned before, we will look at the f1 score of <50k label for the 3 different algorithms and a benchmark. Clearly, there is some room for improvement as all 3 randomized algorithms fell into a local optimum (shown in figure 2.1.a). Confusion matrix for the 4 algorithms can be found in appendix B.

One could suggest that we continue to push the randomized algorithms by searching on a larger hyper parameter space, such as increasing the learning rates and iterations for RHC and SA, or increasing the population size for GA. However, the training time for all 3 randomized algorithms is significantly more than the Adam optimizer (as illustrated in figure 2.1.a), making us believe that randomized algorithms are not suitable for this problem.

OPTIMIZATION PROBLEMS

Next, we will look at some other optimization problems that are suitable for different randomized algorithms. Each algorithm in our experiment uses 1000 max iterations and 100 max attempts and is tested on each problem 20 times for confidence. The average fitness value and search time is compared to evaluate which algorithm is most suitable for a given problem.

The 4 randomized algorithms in this experiment are randomized hill climbing, simulated annealing, genetic algorithm and MIMIC (De et al., 1996).

FLIP FLOP

In our Flip Flop problem, we are given a string that can only contain 3 different elements, 0, 1 and 2. The goal of this problem is to search for a string with the most consecutive same elements. There are 3 global optimum solutions where the solution is the same element repeating from start to end, one for each of the 3 different elements, i.e. 111111…. , 222222…., 333333…..

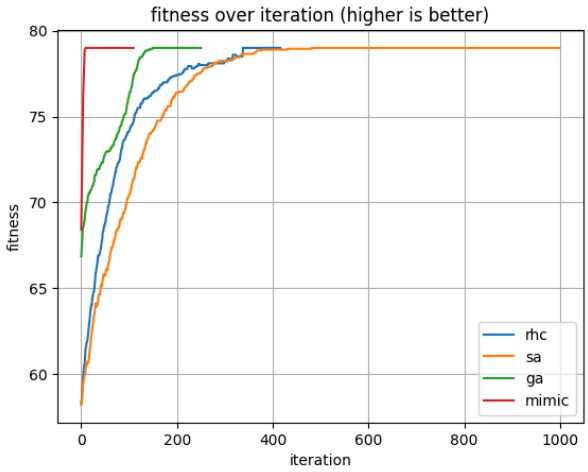


Figure 3.1.a: fitness over iteration

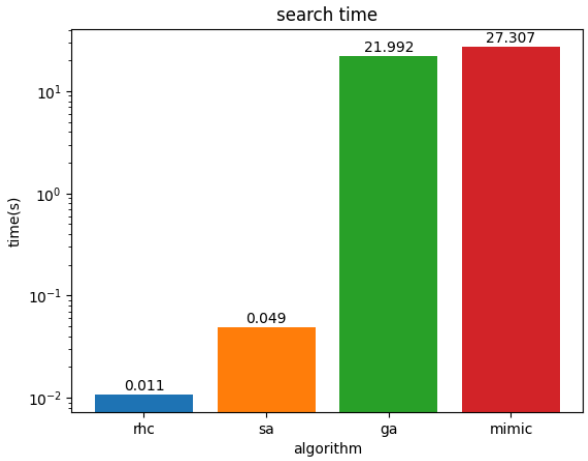


Figure 3.1.b: search time

This is a simple problem where any of the 4 randomized algorithms will be able to reach the global optimum given enough iterations and attempts. The idea here is to demonstrate that a simple problem only requires simple algorithms to tackle it. Both figure 3.1.a and 3.1.b shows that RHC and SA are able to solve it with much shorter time than GA and MIMIC.

KNAPSACK

In our version of knapsack problem, we are given a set of items, each with a weight and a value, and a maximum weight capacity the knapsack can hold. The task is to produce a binary string (1 and 0 for including or excluding the item in the knapsack) that maximizes the total value while keeping the total weight within the weight capacity.

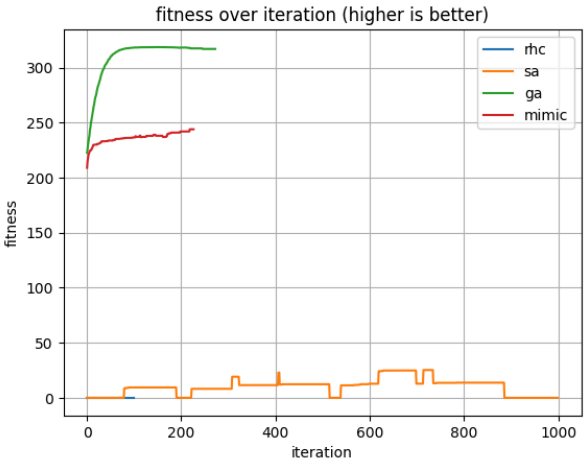


Figure 3.2.a: fitness over iteration

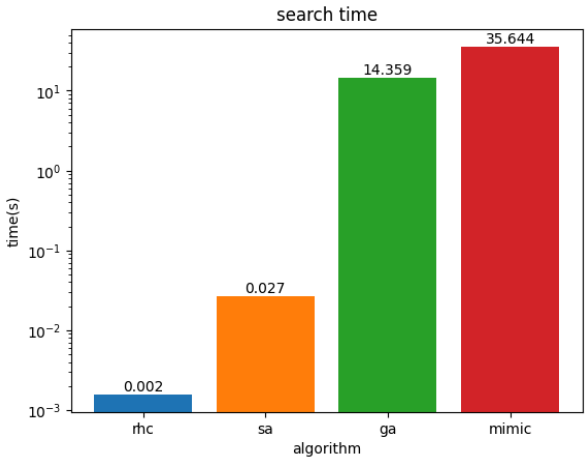


Figure 3.2.b: search time

This is a NP hard optimization problem commonly faced in various problems such as efficient resource allocation. From our experiments, RHC and SA are not able to make any meaningful progress while GA and MIMIC were able to produce some decent results. From figures 3.2.a and 3.2.b, GA is the best algorithm for the problem as it has got a better fitness and shorter search time as compared to MIMIC. GA's crossover mutation is suitable for this knapsack problem as solvers can easily be stuck in a local optimum when it selects an item with a huge weight, limiting the small weighted items to 'fill up the gaps'. At the same time, generating the probabilistic problem for MIMIC is a lot more complex for this problem, causing poor MIMIC scores.

K COLORS

In our version of k colors problem, we are given a sparse graph where the number of edges is only half of its complete graph. The task is to give each node 1 of 4 different colors while minimizing the number of adjacent nodes with the same color.

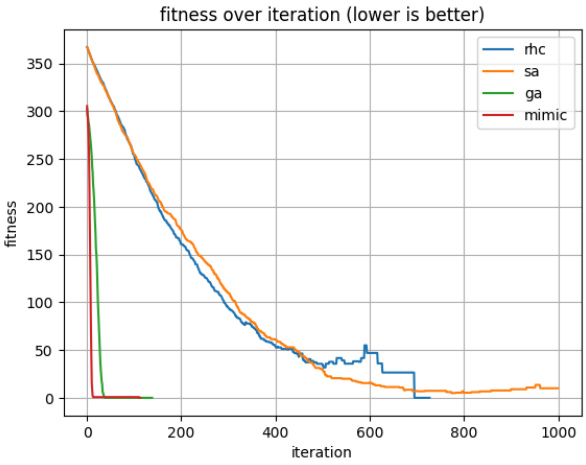


Figure 3.3.a: fitness over iteration

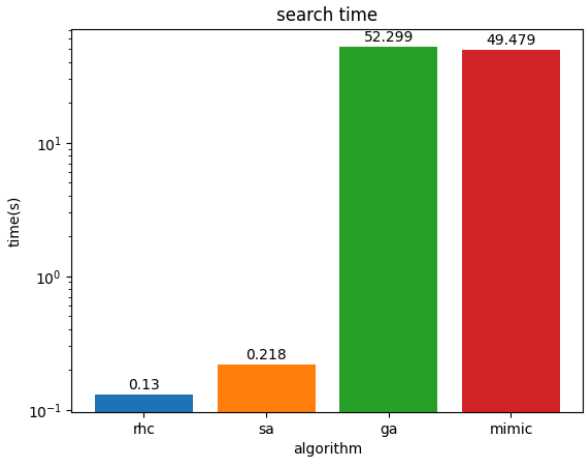


Figure 3.3.b: search time

From our experiments, MIMIC performed slightly better than GA in terms of search time and was also able to converge to the optimum solution with less iterations. This graph problem makes it easy to model it as a probabilistic model which MIMIC excels at solving. GA's crossover mutation allows the algorithm to escape from the local optimum while both RHC and SA are unable to converge on global optimum 100% of the time.

APPENDIX

APPENDIX A

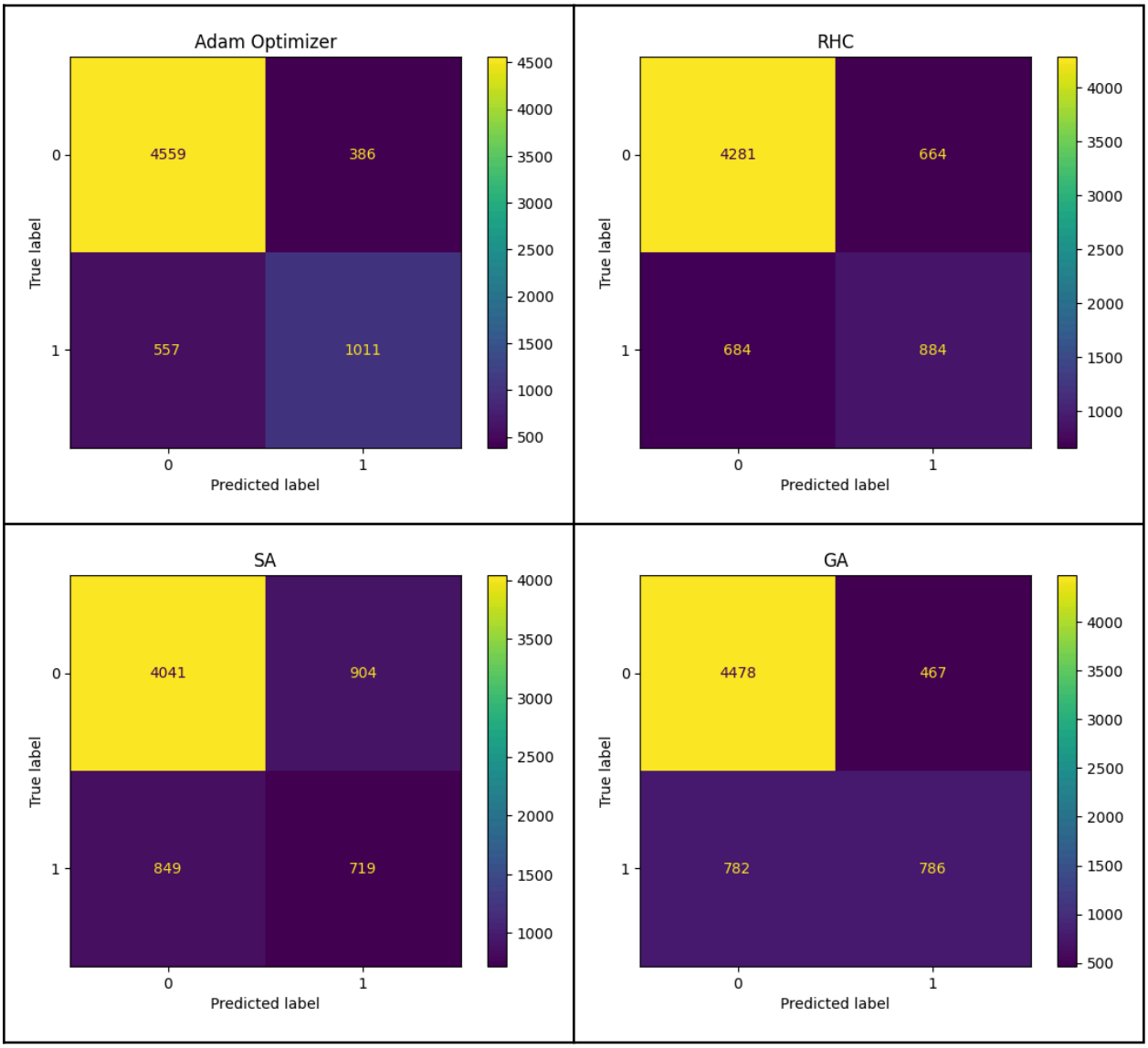
Adam optimizer hyper parameter (sklearn MLPClassifier)

```
clf = MLPClassifier(
 hidden_layer_sizes=[64, 16],
 early_stopping=True,
 max_iter=128,
 random_state=42,
 alpha=0,
 learning_rate_init=0.001,
```

)

APPENDIX B

Neural network confusion matrix



REFERENCES

De, J. S., Charles Lee Isbell, & Viola, P. A. (1996). MIMIC: Finding Optima by Estimating Probability Densities. Neural Information Processing Systems, 9, 424–430.