Jorge Betancourt

11/06/18

CS 4641

Assignment 2: Randomized Search

Dataset Description and Intro

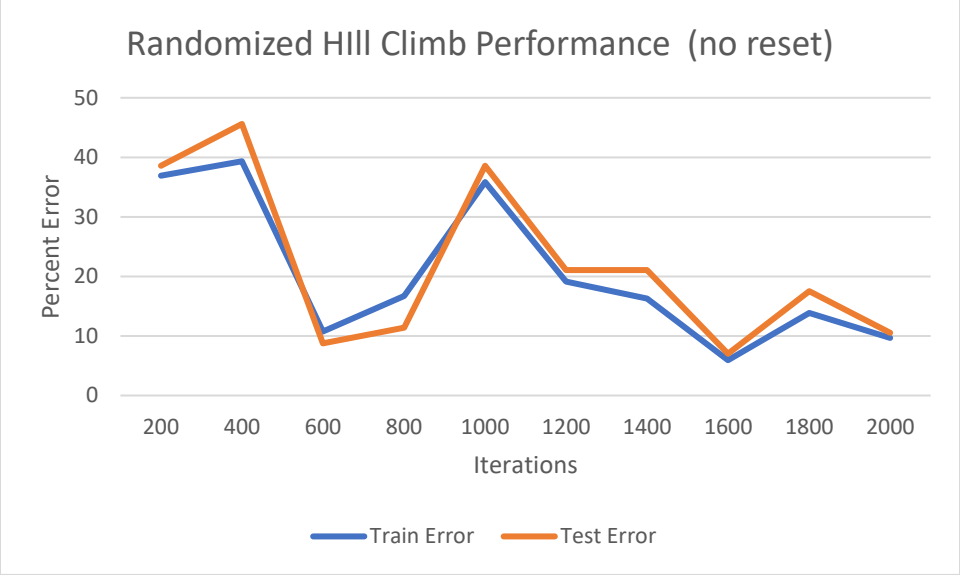
For this assignment, I am using the Wisconsin breast cancer dataset instead of the UFC fight data. I learned from the last assignment that having consistent features was very important in training models with low error. While I tested multiple methods of filling the missing values, in the end, there were too many inconsistencies for what features were collected after each fight, leading to a lot of noise and bad instances. I had assumed that the large number of instances would make up for the occasional missing value, but there were too many missing values in the dataset for it to be viable. Minimal preprocessing was required to utilize the cancer dataset. When I first ran the model, I received high error after many iterations for all the randomized search algorithms, but this was quickly fixed after removing the superfluous ‘id’ feature from the dataset.

The first problem I had to approach was the size of the neural net. The neural net I optimized contains only 1 input layer of size 20. After trying many different NN parameters, I settled on a single 20 node hidden layer because it provides generally good results while maintaining a small search space. This simple net allows the algorithm to quickly find local/global optima. For these experiments, I trained on 80% of the data with 10 folds for cross validation for a better sense of accuracy.

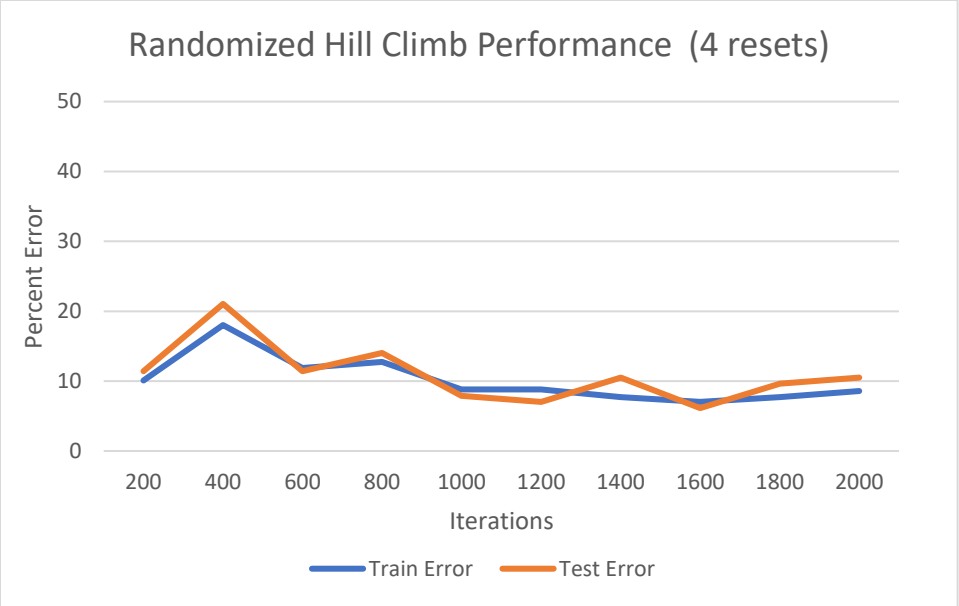
Random Search Algorithms

Randomized Hill Climbing

After determining good parameters for the neural net the algorithms will be optimizing, I recorded the percent error of the randomized hill climb algorithm when optimizing with varying iteration counts.



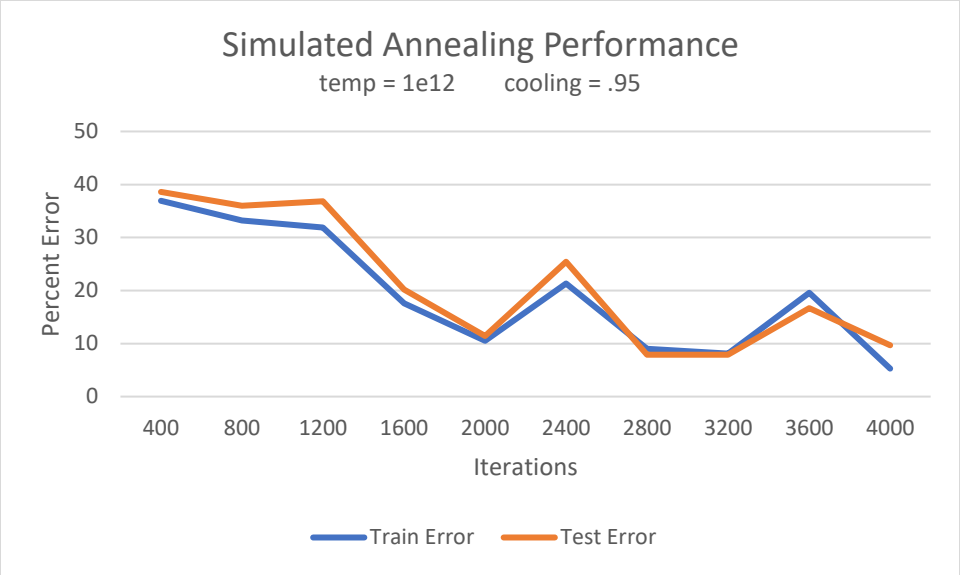
We see a general decrease in error as we increase the amount of iterations, but in some of these instances we see spikes in error (as seen when the number of iterations is at 1000). This is can most likely be explained by the algorithm getting caught in a local optimum. To analyze this further, I implemented the algorithm with random resets so more of the hypothesis space is searched. For the following data I used 4 resets.



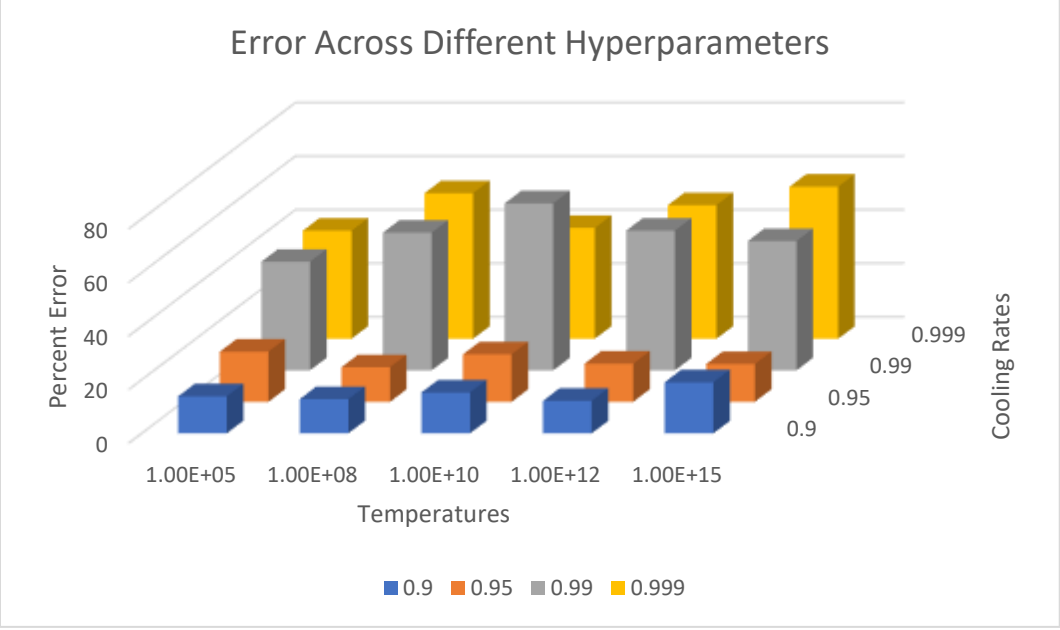
The results are clearly better when using random resets with hill climbing. This is because more of the search space is searched, leading to the discovery of more local optima in different neighborhoods. In the case where we used 1000 iterations, when we didn't use any resets we converged on a local optimum giving us 38.596% error. By resetting only 4 times, we were able to escape that optimum and find weights that gave us 7.895% error.

Simulated Annealing

This algorithm is another approach to getting caught in local optima. I started my analysis of simulated annealing by plotting percent error to the number of iterations. I intuitively selected parameters that I thought would work with the amount of weights we're trying to optimize, and right off the bat I realized that I needed more iterations to achieve an error close to the random hill climbing with random resets. This works because simulated annealing takes less time than random reset random hill climbing, this allows us to run more training iterations of the SA algorithm so we can get comparable results to RHC. The initial temp was 1e12 and the cooling rate was at .95.



Here, we see more consistent results compared to RHC. This is because SA explores more of the hypothesis space for optimal weights instead of getting caught on local maximums like in randomized hill climbing. After this I wanted to test which hyperparameters would work best with this problem, so I set a constant iteration count of 2000 and varied the hyper parameters.



We can see that the most important factor to increasing accuracy was the cooling rate. This makes sense based on the issues we've had before will local maximums; by having a lower cooling rate, the algorithm will explore more of the search space before converging. However that also means that the algorithm will take longer to run, so it is important to balance the two.

Genetic Algorithm

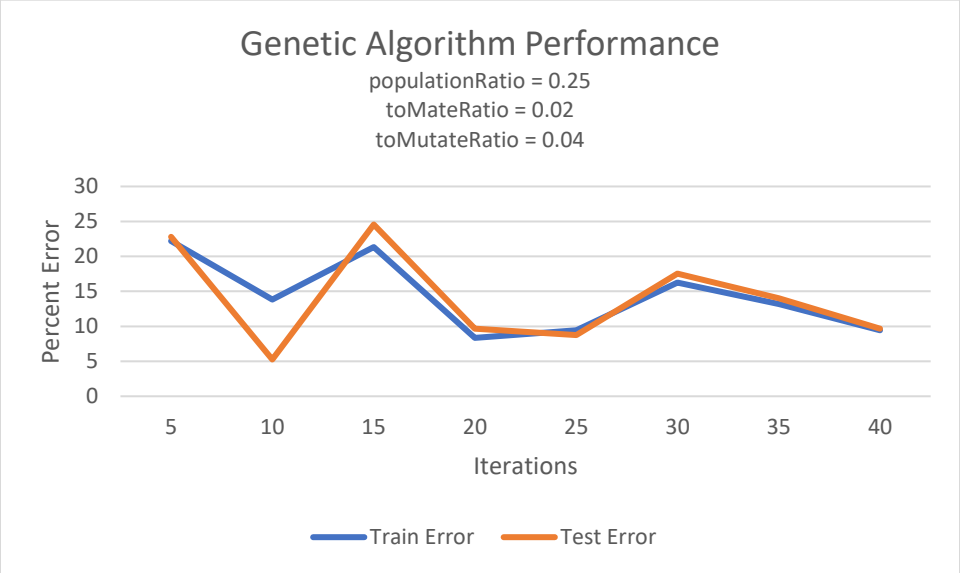
Due to the theoretical lengthiness of the GA's runtime, I decided to experiment with hyperparameters before running through possible iterations in order to see the best results. After running a similar hyperparameter tuning algorithm, I found that the best values for population ratio, mate ratio, and mutation ratio were:

populationRatio = 0.25

toMateRatio = 0.02

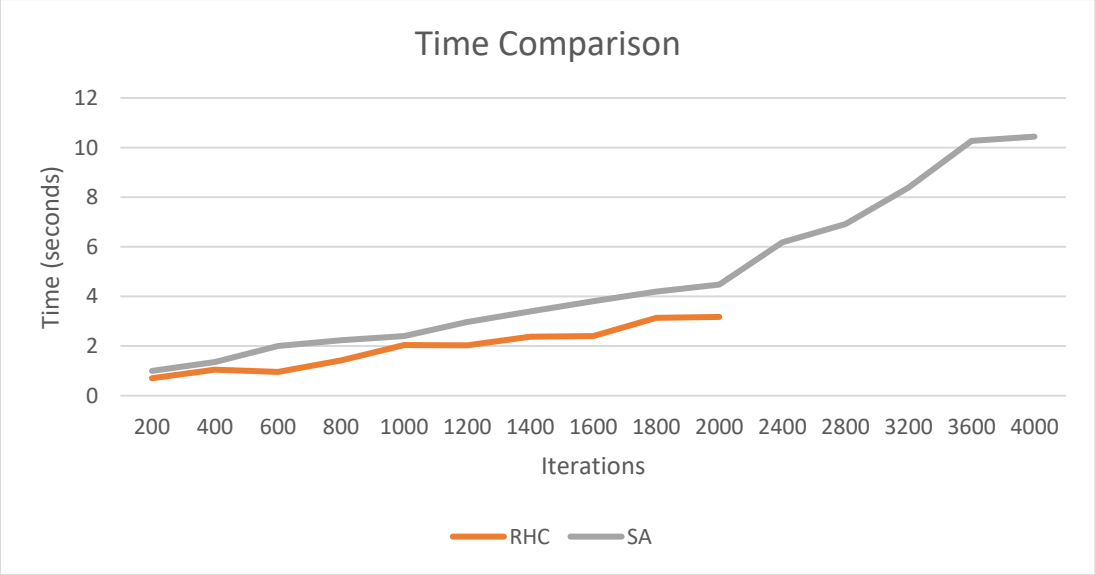
toMutateRatio = 0.04.

With these values I was able to achieve good results when running GA on with different iteration counts.



I was surprised with not only how accurate the algorithm was with such little iteration time, but also with how little time it took for the algorithm to converge. It was able to achieve competitive accuracy compared to the other algorithms in sub 2 seconds, whereas the other algorithms took around 8 seconds to achieve similar results. This is probably because the problem space is structured in a way that it can take advantage of the genetic algorithm's crossover. With the simplicity of the net, the GA can probably isolate features which are more important to the classification problem.

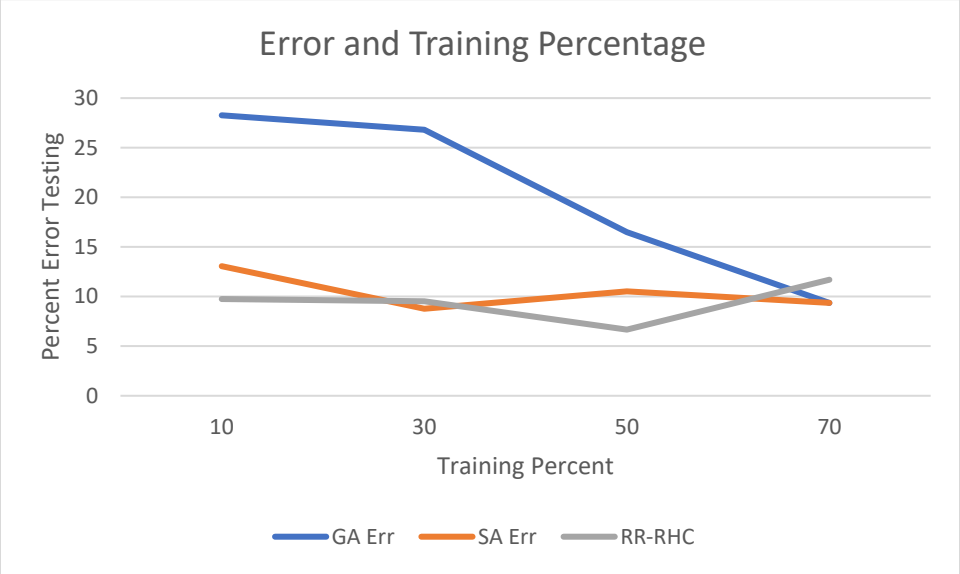
Comparison of Algorithms



Here we can see how long and how many iterations it took for the other algorithms to reach approximately the same error as the best GA run (10%). GA isn't plotted on this graph because it took less than 50 iterations and less than a second to achieve these results. This is probably because there are many local optima SA and RHC need to jump out of before landing on more accurate weights for the neural net.

I wanted to see how each algorithm would do when only provided a limited amount of data, so using the hyperparameters found to be the best for each algorithm (resets for RHC, temp

and cooling for SA, and population, mate, and mutate for GA), I ran the algorithms with an appropriate amount of iterations for each one to see how the testing error would compare.



Something interesting to see here is that SA and RHC with random resets outperformed GA with as little as 10% of the training data. What also is surprising is how accurate the models were given such a small amount of data. I believe this is due to the similarity of the features and the simplicity of the problem. Given enough iterations to converge on a maximum, the algorithms can find appropriate weights that apply to the small amount of training data that also work for the entire dataset. A reason as to why the genetic algorithm doesn't perform as well with a low training percent could be that the GA needs a large training percentage to create a diverse population.

Optimization Problems

Description

Now we turn to two optimization problems to demonstrate the pros and cons of genetic algorithms and simulated annealing in different problem domains. To demonstrate the benefits of

SA over GA, I chose to optimize 4 peaks. To show the benefits of GA over SA, I chose K-coloring.

I chose K-coloring to showcase GA because K-coloring is a clearly structured problem that can use crossover to its advantage. This is because the two parents chosen from crossover will represent/maintain the structure of the graph, so every crossover point will be representative as well. SA will fall short when solving K-coloring because the search space is too large.

I chose 4 peaks to showcase SA because the idea behind SA is that it should be able to easily break out of local maxima when given a low enough temperature. A GA will probably not work as well with this problem because it is not structured well. The piecewise nature of the fitness function might throw the algorithm off, and the fitness landscape is probably too complex for the GA to find a global maximum.

K-Coloring

K-coloring is an optimization problem that tries to minimize the amount of adjacent graph pairs that are colored the same.

When running K-coloring on the three algorithms, it is clear to see which one works best in this search space. RHC and SA were run with 20,000 iterations, and SA was given a cooling rate of .01 in order for the algorithm to explore more of the search space.

	RHC	**SA**	**GA**
Optimal Solution	340.0	340.0	350.0
Solved Problem?	Failed	Failed	Succeeded
Time to run	.165s	.173s	.094s

To ensure that this was not simply a case of poorly tuned hyper parameters, I ran SA again with a lower cooling rate and more iterations. However, the algorithm still converged on 340 and was unable to find the coloring without generating conflict. I would like to perform more experiments

where I see how many iterations it would take for SA to converge on a global maximum if possible. Also experimenting more with the hyperparameters could lead to a more thorough search of the search space. However, due to the lengthy nature of said experiment, I will have to delay that for another time.

4 peaks

4 peaks is an optimization problem that tries to find the Nth dimensional vector X that maximizes the fitness function described in Isbell's MIMIC tutorial (page 8).

$$f(\vec{X},T) = \max\left[tail(0,\vec{X}), head(1,\vec{X})\right] + R(\vec{X},T)$$

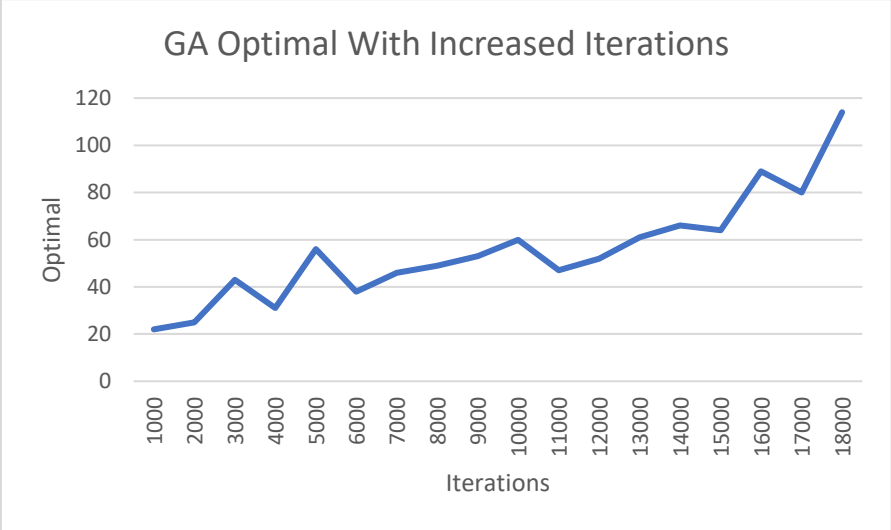
$$tail(b,\vec{X}) = \text{number of trailing } b\text{'s in } \vec{X}$$
$$head(b,\vec{X}) = \text{number of leading } b\text{'s in } \vec{X}$$
$$R(\vec{X},T) = \begin{cases} N & \text{if } tail(0,\vec{X}) > T \text{ and } head(1,\vec{X}) > T \\ 0 & \text{otherwise} \end{cases}$$

For these experiments, T will be a fifth of N. For the first run, N is set to 200. RHC and SA go through 20000 iterations, GA goes through After running each algorithm on this problem, we see where there is a clear advantage SA (and by proxy RHC) have over GA.

	RHC	**SA**	**GA**
Optimal	200	200	14
Found Global Optimum	Succeeded	Succeeded	Failed
Time to run	.165s	.338s	.456s

GA performed horribly compared to the other algorithms. This is most likely because this is an unstructured problem and not every crossover point leads to an improved result due to the piecewise nature of the fitness function. Again, to ensure that this isn't due to a hyperparameter being mis measured, I attempted to increase the iterations until GA found the global maximum. However, as I increased the iterations, it became clear that time is a huge restraint when trying to

force a genetic algorithm to work for a problem it can't solve. Because of this, I stopped when it passed 100.



Conclusion

When finding weight for the simple neural net, the random search algorithms did a good job in finding accurate models. It became clear over time how random these algorithms are, and how after repeating experiments, sometimes outliers would pop up because of something simple like RHC's starting location. I believe my hypothesis was correct when I decided to make a simpler neural net with less weights to optimize, however it would be nice to see the effects of different net shapes on algorithm run time.

Whereas with the neural net, sometimes increasing the amount of iterations was enough to reduce error, the same cannot be said for all optimization problems. At a certain point, time and computing power become too spent for the improvement seen from repeated iterations such as in 4 peaks. By utilizing all three algorithms in different ways, we can see the pros and cons of each one and when they should be used for different types of optimization problems.