Robert Bradshaw

CS 4641 - Machine Learning

3/9/2017

Assignment 2 - Randomized Optimization

Part 1: The Waveform Dataset

The dataset I chose was the Waveform-5000 dataset (which deals with 3 classes of waves and 21 other attributes with continuous values between 0 and 6). In my first assignment, my trained Neural Network (or in WEKA I suppose the function is called the "Multilayer Perceptron") and tried to analyze how it did when certain parameters (the most prominent one in my report being the number of hidden layers to use when building the network) were changed. For Waveform-5000 it performed at its best when using 41 hidden layers. It was able to classify (using 10 fold cross validation) with an accuracy of 83.7% (meaning a root mean squared error of 0.3125). We are also going to use 41 hidden layers as the number of hidden layers (w/ 41 neurons per layer) for our randomized optimization algorithms. The three algorithms we will be running on this dataset are Randomized Hill Climbing, Simulated Annealing, and the Genetic Algorithm. Note that for the entirety of this report we'll be using 41 hidden layers for the neural network. Here's the data collected from the algorithms' performance on the Waveform-5000 dataset (you'll notice that the set of parameters that result in the highest classification accuracy relative to the others parameters of the same iteration number for each algorithm are bolded and red):

Randomized Hill Climbing Algorithm Results

Iterations	Layers	% Correctly Classified	Training Time (seconds)
1000	41	36.34	75.798
5000	41	56.82	415.872
15000	41	59.88	970.197
20000	41	61.18	1375.559
30000	41	62.22	2014.797
150000	**41**	**64.96**	**11692.187**

Simulated Annealing Algorithm Results

Iterations	Layers	Initial Temp	CF	% Correctly Classified	Training Time (seconds)
20000	41	13	95	36.66	1307.565
100000	41	13	95	38.66	6550.104
50000	41	10	90	37.48	3280.476
20000	41	10	90	36.34	1354.318
20000	41	10000	95	38.68	1428.933
20000	41	1000000000	95	29.6	1375.595
20000	41	50	50	39.22	1323.823
20000	41	50	90	37.46	1437.301
20000	41	50	0.9	60.78	1610.897
20000	41	1000	0.2	61.16	1275.91
20000	41	13	0.95	61.08	1438.426
20000	41	10000	0.9	60.62	1482.226
20000	41	1000	0.05	60.36	1399.293
20000	41	1000	0.8	61	1866.97
20000	41	1000	0.85	60.7	1866.97
20000	**41**	**1000**	**0.9**	**61.622381**	**1400.014**
20000	41	1000	0.95	61.18	1339.774
20000	41	1000	2	32.42	1413.825
20000	41	500	0.8	60.94	1580.789
20000	41	3000	0.8	60.6	1640.322
20000	41	3000	0.9	60.78	1335.288
20000	41	500	0.9	61	1544.913
20000	41	100000	0.9	60.96	1330.491
100000	41	1000	0.9	64.210403	6652.455

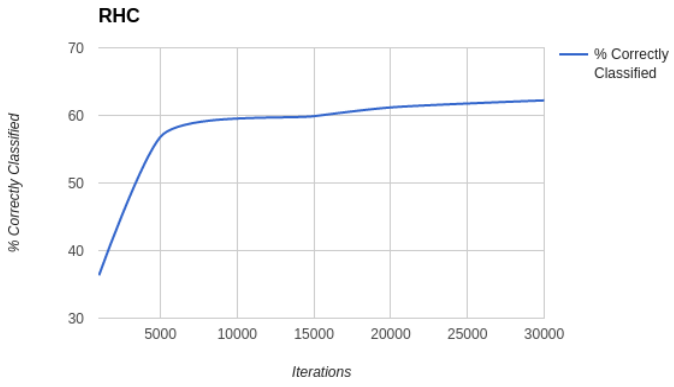
Iterations	Layers	Initial Temperature	CF	% Correctly Classified
10	41	1000	0.9	38.998
100	41	1000	0.9	40.7696
1000	41	1000	0.9	35.33568
2000	41	1000	0.9	46.84559
10000	41	1000	0.9	57.733104
20000	41	1000	0.9	61.622381
100000	41	1000	0.9	64.210403

Genetic Algorithm Results

Iterations	Layers	Starting Population	Mates Per Iteration	Mutations Per Iteration	% Correctly Classified	Training Time (seconds)
3000	41	100	100	10	42.42	7658.225
3000	41	100	50	10	38.66	3773.909
3000	41	200	50	10	43.64	3841.951
3000	41	200	100	10	45.7	2579.117
3000	41	200	150	10	42.7	11753.801
3000	41	300	100	10	48.48	7459.924
3000	41	300	100	50	50.74	7548.938
3000	41	300	150	30	49.08	11719.306
3000	41	300	150	50	49.26	11040.95
3000	41	300	150	100	50.18	11194.397
3000	41	1000	100	50	52.82	7945.812
3000	**41**	**1000**	**100**	**150**	**53.7**	**13207.051**
3000	41	10000	100	150	33.8	11085.274
10000	41	1000	100	150	62.13	41034.221

Randomized Hill Climbing (RHC)

This is the simplest algorithm we use here, which makes it a good baseline to talk about first. Basically, on each iteration we take a look at the current cost of the current “position on the hill” we are, pick a random neighbor to journey to, and decide whether to move towards that neighbor or to pick another neighbor.

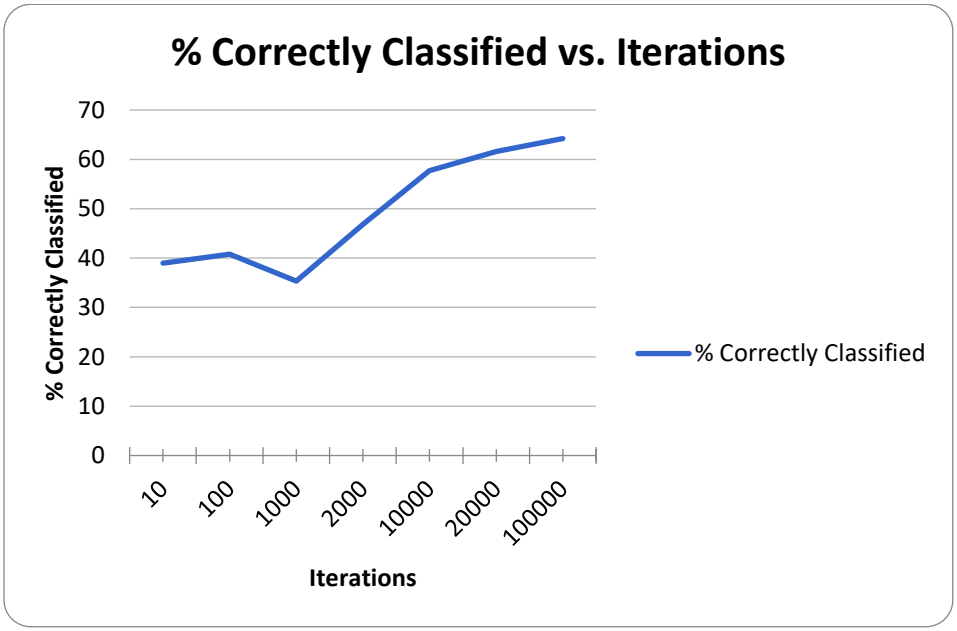


When run on Waveform-5000, the randomized hill climbing didn’t do too shabby. As the number of iterations increased, the classification accuracy of the algorithm increased logarithmically.

Simulated Annealing

This algorithm is interesting in that it attempts to emulate annealing in the metallurgical sense (I imagine the blacksmithing of a katana like the weeaboo trash I am when I hear the name of this algorithm). Basically, when metal is heated to a high temperature, it is soft and the shape can be dramatically changed very easily. With blacksmithing, you cool this extremely hot soon-to-be blade slowly as you work on it so the features become more and more structurally sound and refined as the process continues. It is easier to make dramatic changes to the shape of the blade when it is hotter, but it is difficult to be precise while it is extremely hot, which is why you cool the blade as you make progress on the shape of the blade and generally want the blade to be cooler while you're adding more precise detail to the blade after the general shape is determined.

This algorithm probabilistically determines neighbors to journey to while simultaneously attempting to move the overall "physical system" from a point of initial energy (set by the initial temperature parameter) to a point of minimum energy through cooling. By cooling the temperature slowly from a point of initial high energy you can find the global maximum of an optimization problem more easily since you're ignoring the local maxima (unlike with RHC -- which innately gravitates towards local maxima). Simulated Annealing essentially "scans" for the most likely area for the global maximum to appear.



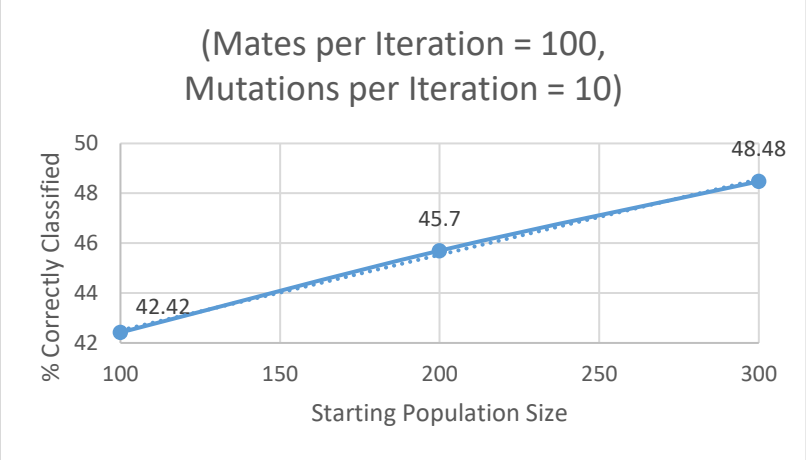
A similar logarithmic trend in classification accuracy relative to the number of iterations that is present in the RHC graph can be seen here as well. However, you'll notice a remarkable dip in accuracy at 1000 iterations. Since as compared to the performance at 100 iterations. Being a probabilistic model, simulated annealing will make incorrect assumptions early in the iteration sequence (while the temperature is hot). Since we iterated for barely any time here, it isn't exactly impossible for our model to finish the iteration process before it gets close to finding the probable global optimum range. Simulated Annealing classified with 61.622% accuracy with 20000 iterations – Randomized Hill Climbing classified with a 61.18% accuracy. Simulated Annealing only took 25 more seconds to run than RHC at 20000 iterations. As such, it is clear to us that Simulated Annealing resulted in a better solution than Randomized Hill Climbing for the Waveform-5000 dataset. This makes sense because simulated annealing essentially is probabilistic hill climbing where you can go both up and down with varying degrees of stride (it better do better than randomized hill climbing if it's innately more complex). While running all of these algorithms, I found that a moderately high Initial Temperature (in the thousands) paired with a Cooling Factor that is a real positive number less than 1 yielded higher classification accuracy than low temperatures with large Cooling Factors and extremely high Initial Temperatures with a low Cooling Factor. The best parameters I found were Initial Temperature of 1000 and a Cooling Factor of 0.9. As indicated in my data, I ran trials where I tweaked one of these parameters ever so slightly while keeping the other parameter constant only to find that this combination does the best overall with Waveform.

Genetic Algorithm

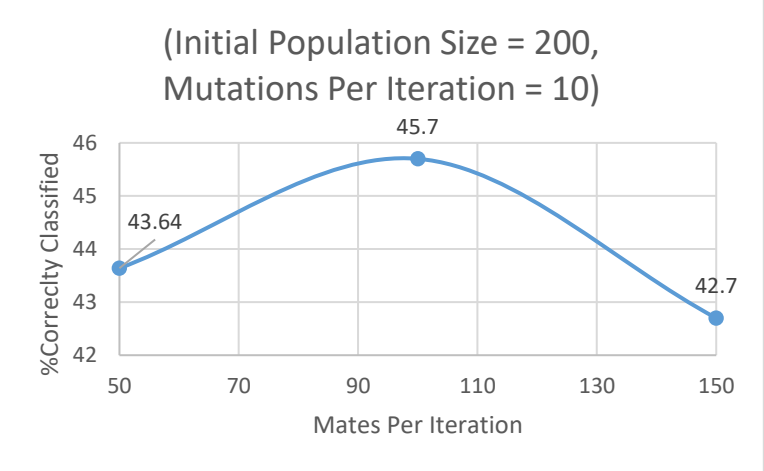
This algorithm attempts to model the biological process of evolution by creating a population of randomly initialized weights for a neural network, taking the ones that perform the best in a generation, breeding them with one another (by crossing over partial solutions to the network optimization problem -- i.e. a subset of the weights), and mutating some of the weights (which helps the algorithm avoid getting stuck at a local maxima). Mutation aids in escaping local maxima since eventually a random change in a given set of weights for the network will cause a member of the population to classify slightly better than other members of the generation. This slightly more optimal change in the network eventually gets propagated to the successive generations (this pretty much emulates the process of natural selection). The main downside of this extremely cool-sounding algorithm is that it has no idea where to search and is way more computationally intensive than the rest of the algorithms due to the crossover operations within a given population size. As such, finding the optimal solution for a given

problem with genetic algorithms takes forever (just like how evolution from apes to humans took eons, due to the inherent stochastic and nondeterministic nature of the algorithm).

The run time of this algorithm is substantially longer than both randomized hill climbing and simulated annealing; 100000 iterations of RHC takes half as long to complete as the majority of the genetic algorithm trials do with only 3000 iterations (as evidenced by the data table). I can see why Isbell says utilizing this algorithm in the real world is both impractical and "stupid". I tested a myriad of parameter combinations to see what parameters yielded the most accurate classifier. Increasing the population size was shown to linearly increase the classification accuracy of the neural network optimization problem to an extent.

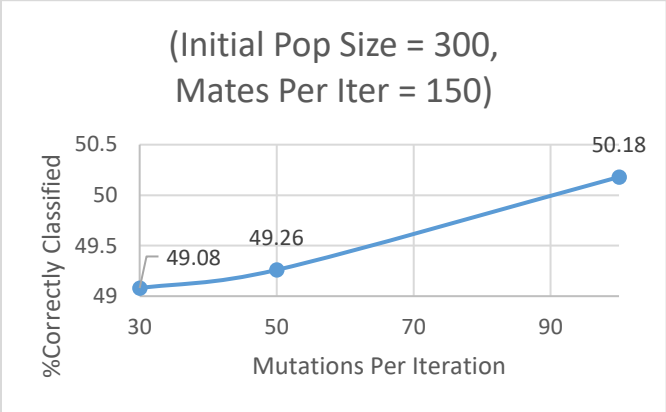


Mates Per Iteration was observed to have a "sweet spot" within the range we tested for.



Increasing the mutations per iteration was observed to increase classification accuracy linearly within the range we tested for. Since more mutation allows us to escape local maxima by adding more entropy to the algorithm, it also logically impedes convergence if the good mutations aren't propagated through to the next generation. This is why increasing the mutation rate in the

following example where half of the population is mating works well. I imagine this success drops off if the entire population is mutating constantly.



The genetic algorithm takes 3000 iterations to classify with 53.7% accuracy. RHC takes approximately 4000 iterations to reach around 53% classification accuracy and Simulated Annealing takes approximately 10000 iterations to classify with a 57.731% accuracy. "Throughput"-wise, the genetic algorithm performs better early on with the Waveform-5000 dataset than both simulated annealing and RHC (since it takes way less iterations). However, actual timewise, simulated annealing comes to answer faster since the crossover operations take so long with the genetic algorithm.

Final Notes on Part I: The Waveform Dataset

Overall, I'd say that Simulated Annealing did the best for Waveform. All three algorithms were capable of arriving at similar levels of classification accuracy eventually, but Simulated Annealing does it slightly more accurately than the next best algorithm (which is RHC). Genetic Algorithms are not practical for neural network optimization obviously due to how complex these classification problems get due to the curse of dimensionality. It isn't practical to stochastically determine what minor change in weights caused the improvement in fitness for a generation when there are so many attributes to check through and against one another. I was surprised that RHC did almost as well as Simulated Annealing – but it makes sense if the function's values are shaped like a bell curve (which I'm inclined to believe is the case for Waveform-5000).

Training the Waveform-5000 dataset using WEKA's ZeroR algorithm w/ 10-fold cross validation (the simplest classification method possible) produces a classification accuracy of **33.84% in under a second**. Training the Waveform-5000 dataset using WEKA's Multilayer Perceptron algorithm (i.e. backpropagation) w/ 10-fold cross validation produces a classification accuracy of **83.7% in under 5 minutes**.

The randomized optimization algorithms were only capable of achieving a maximum of approximately **64% classification accuracy** for the neural network optimization problem **in around 190 minutes for RHC, 110 minutes for Simulated Annealing, and at least 683 minutes for the Genetic Algorithm**. This sheer discrepancy between the accuracy of supervised learning (backpropagation) and randomized optimization makes sense considering these stochastic algorithms are only useful/relevant due to their simplicity. It's a lot easier to randomly search for the correct answer than it is to come up with a concrete approach towards solving a complex classification problem (supervised learning) – it's just orders of magnitude more computationally intensive and not guaranteed to find the optimal solution at all depending on the dimensionality of the problem. Randomized Optimization gives us insight to the idiosyncrasies of the data we're working with prior to running more realistic learning algorithms on it.

Part II: Optimization Problems

The three optimization problems I ran with were FourPeaks, the Traveling Salesman Problem, and FlipFlop. For FourPeaks we are maximizing the following function:

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

where

$$\begin{aligned} tail(b, \vec{X}) &= \text{number of trailing } b\text{'s in } \vec{X} \\ head(b, \vec{X}) &= \text{number of leading } b\text{'s in } \vec{X} \end{aligned}$$

$$R(\vec{X}, T) = \begin{cases} N & \text{if } tail(0, \vec{X}) > T \text{ and } head(1, \vec{X}) > T \\ 0 & \text{otherwise} \end{cases}$$

For the Traveling Salesman Problem we are maximizing the inverse of the distance of the longest path found (i.e. finding the shortest path). For FlipFlop we are maximizing the number of alternating bits found in a bitstring. This time we're comparing four algorithms: the three we utilized to analyze randomized optimization on Waveform-5000, and one more named MIMIC,

which stands for Mutual-Information-Maximizing Input Clustering (what kind of scrub would give their algorithm such a ridiculous name???). MIMIC produces optimal values by sampling parts of the input spaces that are most likely to contain optimal values, running density estimation on those spaces, and then propagating useful information about the function to the next iteration.

Here's how all four algorithms (the parameters used for each algorithm are in the project2 Java files in the src) performed on these three optimization problems – the algorithm that performed the best (as in the algorithm that provided the relative maximal value) is highlighted in red. These data tables contain the averages of 3000 trials:

TSP (N = 20)

Algorithm	Average Optimal Value	Average Time (seconds)
RHC	0.237765763	0.02088869
Simulated Annealing	0.238818764	0.187931257
Genetic Algorithm	**0.289911567**	**0.118970181**
MIMIC	0.282619659	0.985744227

Traveling Salesman is a well-known NP-hard problem. Given N cities, we are to find the shortest possible path that visits each city once and returns us back to the city we started in. Since we know this problem is NP-hard, it makes sense that the genetic algorithm performed the best on it given its seemingly infinite scope of its search space. Probabilistic modeling and mindless climbing isn't enough to solve a problem as difficult as TSP, which is why MIMIC, Simulated Annealing, and RHC fall short here. Funnily enough, the genetic algorithm runs faster than MIMIC and Simulated Annealing here. Since we're not working with neural networks or complex classification problems here, it is entirely feasible to use the Genetic Algorithm here since runtime isn't as much of a hindrance.

FourPeaks (N = 100, T = 6)

Algorithm	Average Optimal Value	Average Time (seconds)
RHC	102.418	0.030791732
Simulated Annealing	180.507	0.125253959
Genetic Algorithm	118.5476667	0.052751607
MIMIC	**188.7046667**	**2.810756808**

This problem essentially has two local maxima of N (where the bitstrings passed in contain entirely 1's or entirely 0's), and two global maximums of 2N – T – 1 (which means we're trying to get as close to the integer 193 as possible in this example). Randomized Hill Climbing clings to the local maximum of 100 for this problem and as such – performs the worst. Simulated Annealing is great at finding global maxima in situations where there exists a myriad of local

maximums, which is why I thought it was going to do the best on this optimization problem. However, surprisingly enough, MIMIC comes in first place here. MIMIC analyzes the likelihood that an optimal value exists in a certain range via sampling, which gives it a slight edge over the somewhat myopic probabilistic algorithm of Simulated Annealing since we have the trigger T to search around.

FlipFlop (N = 100)

Algorithm	Average Optimal Value	Average Time (seconds)
RHC	81.10866667	0.040189909
Simulated Annealing	**98.69866667**	**0.135008086**
Genetic Algorithm	87.882	0.064014728
MIMIC	86.80933333	5.712593933

The function to maximize is the number of alternating bits in a bitstring. As such, the greatest value possible is the value of N. Since so many bitstrings such as “01011” and “10100” have the same return value, this function has a decent number of local maxima to maneuver around. RHC is just dumb and falls into these ruts. Simulated Annealing works very well with large numbers of local maxima due to the nature of heating and cooling to find the general range of the optimal value that I’ve explained a million times already.

It is worth noting that Randomized Hill Climbing does the worst on all of these optimization problems due to its propensity to cling to local maxima. It is the fastest because it is computationally the cheapest – but it’s imperative to recognize that you’re getting what you pay for here.

MIMIC pretty much always takes the longest to run since it has to sample a changing probability distribution multiple times after complex density approximation operation.

References & Things I Used

- http://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf
- https://github.com/james7132/GTCourseWork/tree/master/CS%204641%20-%20Machine%20Learning/Randomized%20Optimization%20Assignment
- WEKA
- The single perceptron I have in my head