**CS 4641 – Machine Learning**

**Assignment 2 – Randomized Optimizations**

**Part 1: Experimenting with the weights of a Neural Net**

**Dataset**

For the first part of this assignment, I decided to use my nursery dataset from the previous assignment about supervised learning. The dataset contains instances of different applications of potential students to nursery schools in Ljubljana, Slovenia. The features such as number of siblings, financial/economical situations of the parents, social status of the families are given in different discrete categories/rankings. For the label, each instance has given a ranking from 1 to 5, ranging from ‘not recommended’ to ‘special priority’. There are total of 12960 instances that have 8 distinct discrete attributes in the dataset. There are no missing fields in the dataset. One important point about this dataset is the distribution of the labels, while three labels have around 4000 instances associated with them; the rest two contains much less instances. This is an import feature about the dataset that needs to be noted when running tests. Even though there isn’t a single dominant label, 2 of them seem weak compared to the other 3. However, the dominant labels are found at the both end of the spectrum of rankings, making this dataset still a valuable resource data-wise.

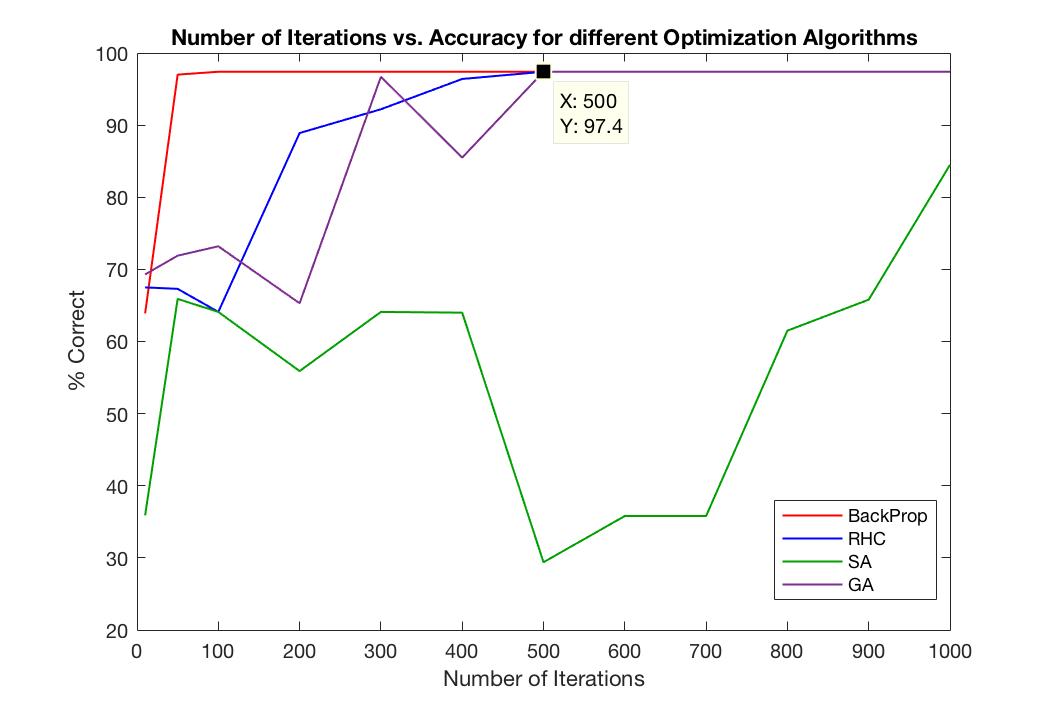
**Experiment**

After experimenting with ABAGAIL, the machine learning library, I was not able to create an accurate model of a multi-label classification. Instead, I decided to group the labels into two using the domain knowledge I had and since it was kind of a ranking system between 1-5, it made sense. The higher two labels were called ‘priority’ and ‘special priority’ applications. So in this experiment, I try to classify an application if it’s a ‘priority’ or ‘non-priority’ application. In the end, I created a binary classification problem out of the dataset I have used in the previous assignment.

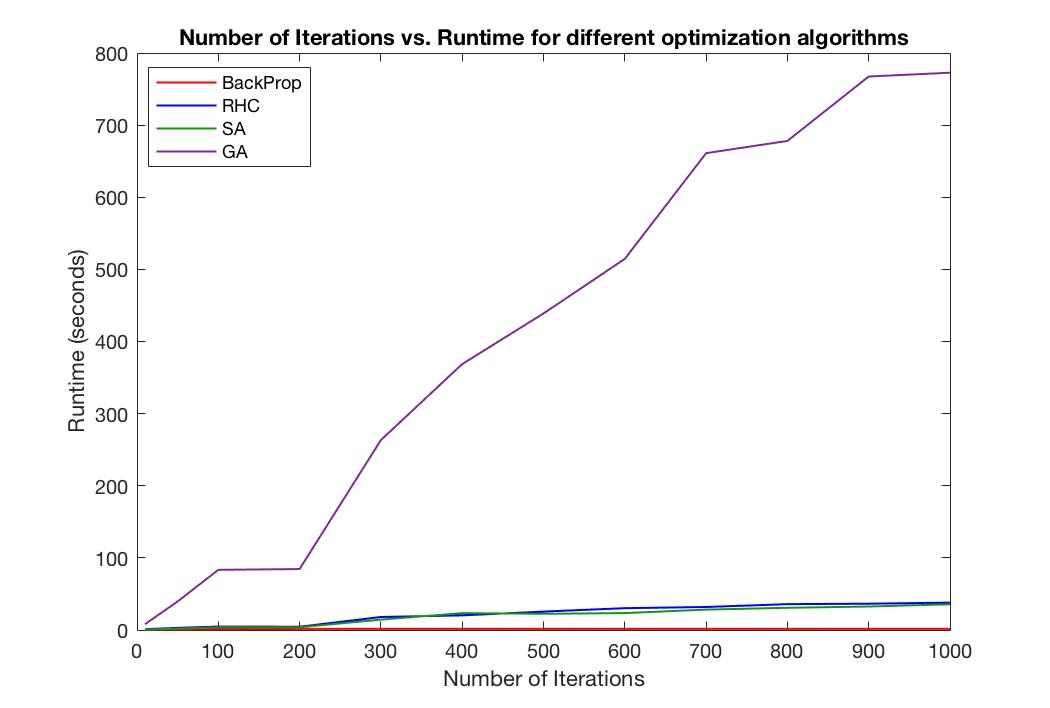
In order to be able to compare different randomized optimization algorithms and their effects on the rearrangement of the weights of a neural network, I re-implemented the neural network from the previous assignment to work with the new binary classification problem I created for this assignment. I used python’s *sklearn* library and the generic neural net classifier that comes with it. The default back-propagation algorithm was stochastic gradient descent, which is a very optimized algorithm. Because of that, I was not expecting any of the randomized optimization algorithms to surpass that both in accuracy and runtime.

To see how algorithms behave, I tested those 4 (backprop and 3 randomized optimization algorithms) on a neural net that has 5 hidden layers with 2 neurons on each layer. I compared both the accuracies and runtime values over increasing number of iterations.

**Plots**

****

**Figure 1:** Number of iterations versus testing accuracy on different optimizations for neural network weight change.

****

**Figure 2:** Number of iterations versus runtime of different optimizations for neural network weight change.

**Results**

The independent variable of this experiment was the increasing number of iterations. I have tested different algorithms on iterations of 10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 and 3 out of 4 algorithms managed to reach to the same accuracy of 97.4% after different number of iterations. As expected, the neural net from the previous homework, the one with the stochastic gradient descent, reached to the maximum accuracy the fastest with around 50 iterations as can be seen in figure 1. On the other hand, simulated annealing performed very poorly compared to the other algorithms and the highest it reached was around 84% with 1000 iterations.

RHC takes around 500 iterations to reach its maximum potential. With the random starts it has, RHC is the simplest algorithm of all. 500 iterations is relatively too much compared to the backprop, suggesting that there are lots of local minima that it gets stuck in.

SA is similar to RHC but it has an improved exploration feature that I will be talking more about in the later sections. However, SA performs the poorest in this case, suggesting that it fails to lower its temperature for the right extreme and exploit more in the global minimum. Looking at the dataset, we see that almost all feature/attribute combinations are covered and a possible equal distribution of labels could be causing more of a flat shaped graph with many shallow local minima, preventing SA to find the global one.

We see that GA also reaches the maximum accuracy of 97.4% with around 500 iterations. The accuracy curve of GA in figure 1 looks similar to RHC’s curve. Like RHC, GA choses points randomly and mutates a new population of solutions to choose the best and generate more generations. Taking around 500 iterations, just like RHC does, GA shows us that there are lots of local extremes in the dataset.

Looking at the figure 2, GA obviously take much more time compared to any other algorithms and as expected, backprop takes the least time. The reason for that is manages to converge at the right weight values, causing the algorithm return early when the perfect classifier model is created. The reason GA takes the longest training time is because it generates an entire population for each data point for the number of iterations and this takes up the longest processing time.

**Part 2: Random Optimization Problems**

**Problem 1:**

**Problem 2:**

**Problem 3:**