RandomizedOptimization

February 15, 2015

1 Randomized Optimization Algorithms

1.1 Introduction

In the previous assighnment we analysed various supervised algorithms, which were all solving some optimization problem in the form of minimizing the derivative of the error. What if the derivative does not exists, like in discrete problems which are not defined on continuous functions? Since discrete functions cannot be differentiated, then gradient descent tool cannot be used. Here we turn towards search and various way of optimizing it using randomized optimization algorithms.

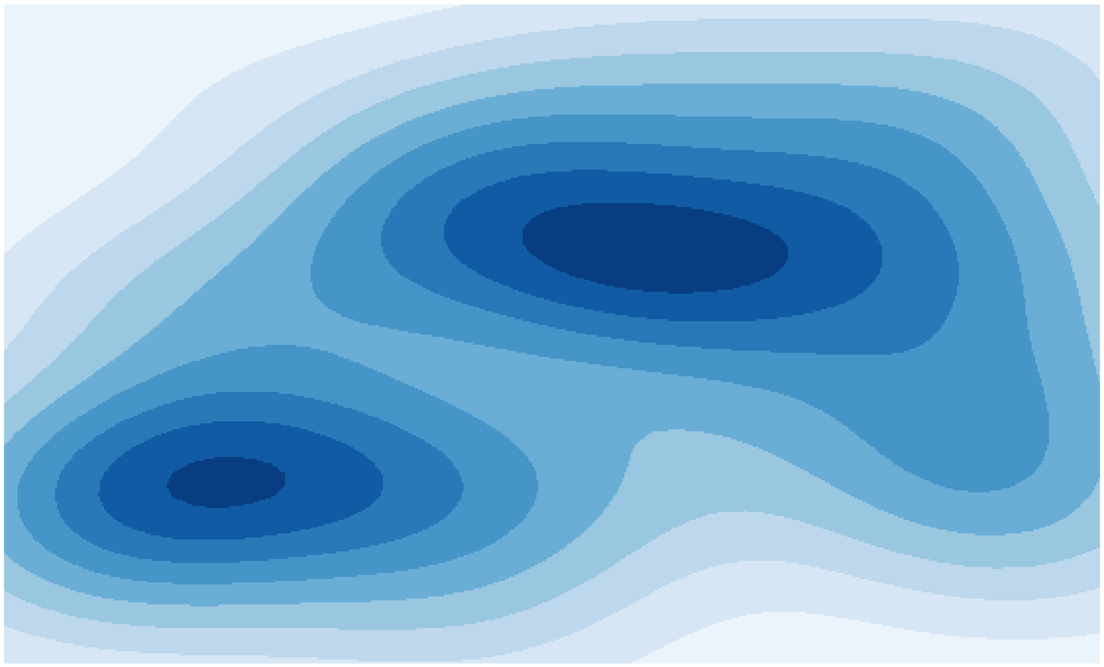
1.1.1 Genetic Algorithm on the problem "Where is Waldo?"

I only recently came across Waldo-spotting book series (since I grew up outside of US) through the blogpost of Randy Olson where he demonstrated optimal search using genetic algorithm. This problem fascinated me by being both fun and illuminating and here I tried to reproduce some of his results using different implementation of the genetic algorithm

```
In [1]: %matplotlib inline
        from algo_evaluation.datasets import *
        from algo_evaluation.plotting import plot_waldo_data as plt
```

```
In [2]: df = load_waldo_dataset()
```

```
In [7]: plt.plot_waldo_kde(df)
```



Fitness function We need to minimaize the distance between what waldo-looking solutions covers and the real waldo coordinates.

Genetic algorithm continuosly tinkers with the solution by slightly mutating the existing best solution until no better solution can be found.

1.2 Finding weights for ANN

In this section I will use optimization algorithms to search for the best weights in the Neural Network classification on the problem of Higgs detection from the previous assignment.

Quick review:

- given outcomes of particle decays, detect Higgs boson;
- most of the supervised algorithms gave acceptable accuracy ranging from 0.8 - 0.9;
- neural network gave the worse accuracy of around 0.5 (50%) across all expreriments with tunning paramaters;

Since backpropagation algorithm in the neural network did not provide satisfactory accuracy on the higgs dataset, the problem is higgs detection becomes a great candidate to apply randomized optimization algorithms weights learning.

From the previous assignment it was concluded that dataset size of 20k records will suffice the experiment while saving running time significantly:

```
In [45]: higgs_data = load_higgs_train(sample_size=20000)
         features, weights, labels = higgs_data
         print 'Size of the dataset:', features.shape[0]
         print 'Number of features:', features.shape[1]
         print 'Number of positives (signal):', labels.value_counts()['s']
         print 'Number of negatives (background):', labels.value_counts()['b']
```

```
Size of the dataset: 5477
Number of features: 13
Number of positives (signal): 2525
Number of negatives (background): 2952
```

Learning weights for the neural network will be achieved using pybrain library (same as in the previous assignment).

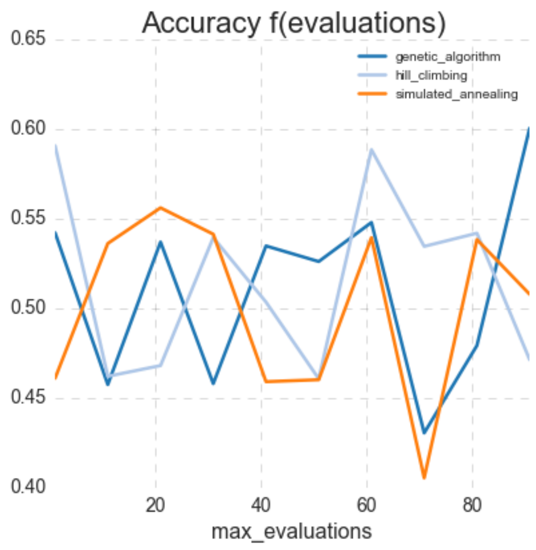
The **fitness function** here (called evaluator) will be **MSE** (maximum squared error) and the goal is to minimize it using three different optimization algorithms: HillClimber, Genetic Algorithm and Simulated Anealing.

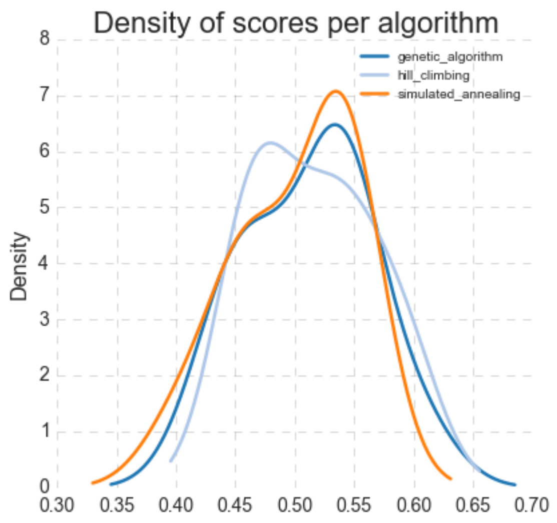
```
In [56]: from algo_evaluation.algos import neural_network as nn
```

```
In [57]: df = nn.compare_weight_learning_optimized(higgs_data)
```

Overall, comparing to Backpropagation algorithm, learning weights by means of randomized optimization algorithms performed slightly beter and significantly faster.

```
In [58]: learn_plot = nn.plot_weight_learning_accuracy(df)
```

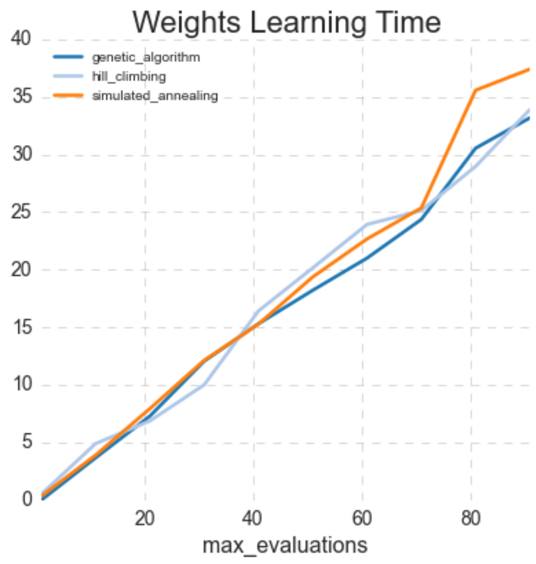


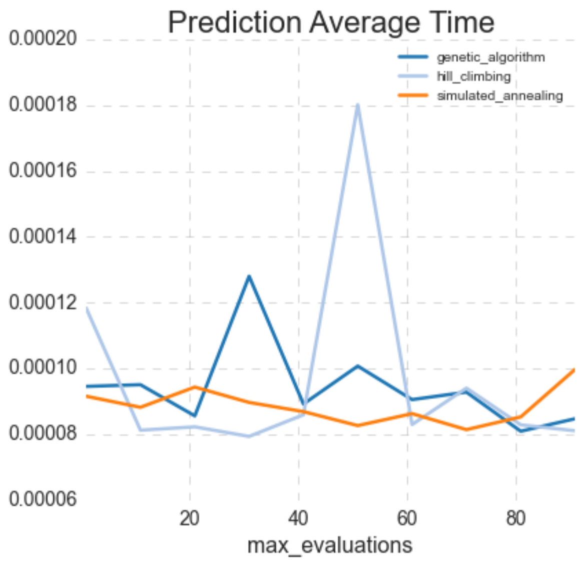


Accuracy: On the left side, learning accuracy curve is presented as a function of max_evaluations and since there is a lot of volatility, I also presented the KDE plot of the accuracy scores.

KDE plot showed that *simulated annealing* and *genetic algorithm* on average performed better that *hill climber*.

```
In [59]: time_plot = nn.plot_weight_learning_time(df)
```





Running Time: Evaluation three randomized algorithms for learning optimal weights did not give much variability for running time. All algorithms are comparable and time grows linearly as number of evaluations increases. I also showed the prediction times which are averaged across dataset size - again comparable results.

1.2.1 References

[1] Pybrain Optimization Documentation, Online Available, at http://pybrain.org/docs/tutorial/optimization.html

[2] http://www.cc.gatech.edu/~isbell/papers/isbell-mimic-nips-1997.pdf

[3] http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.optimize.anneal.html

[4] http://www.randalolson.com/2015/02/03/heres-waldo-computing-the-optimal-search-strategy-for-finding-waldo/