Project 7: Genetic Algorithms

Josh Bodah

Overview

- Used “adult” census income data set
- Wrote own Ruby code to implement genetic algorithm to learn the weights for a neural network
- Based on psuedocode in Russel and Norvig

Preprocessing

- Used Weka to convert nominal attributes into binary numeric ones

- Wrote code to normalize the attributes and remove any instances with missing rows

Design Decisions

- Hypothesis encoded: used an array instead of a binary string. n + 1 weights per node.
- Population: size varied (25, 50, 75, 100). Weights were initialized by randomly sampling [-1,1]
- Fitness function: used classification accuracy on the data (take sum of weights * attribute values and threshold)

Design Decisions

- Selection method: used fitness proportionate selection. Two parents were chosen to spawn two children during each reproduction phase.

- Crossover: used single point crossover. When parents reproduced, two paired children ([0, cross-1] of mother, [cross-1, size] of father and vice versa) were spawned.

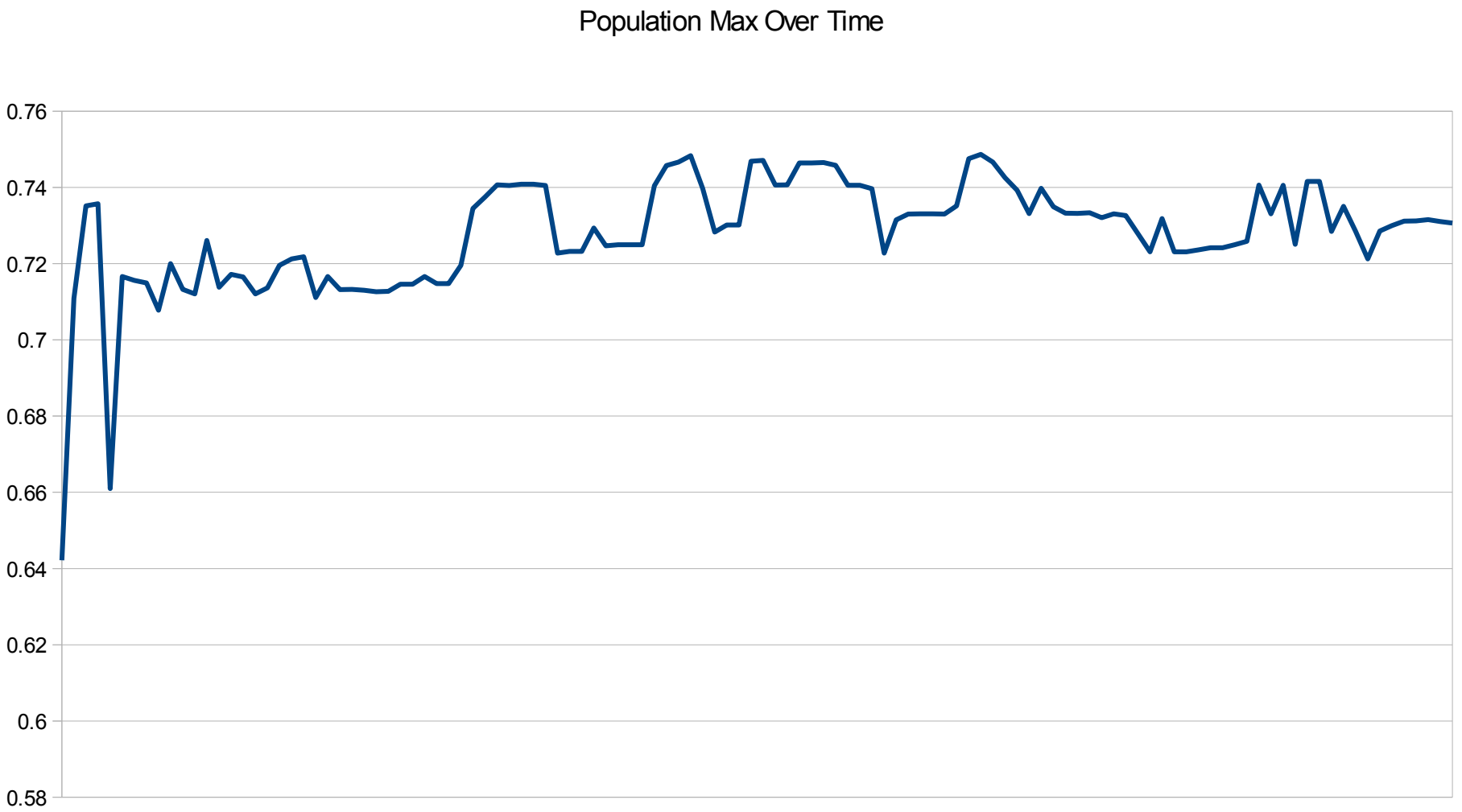
Design Decisions

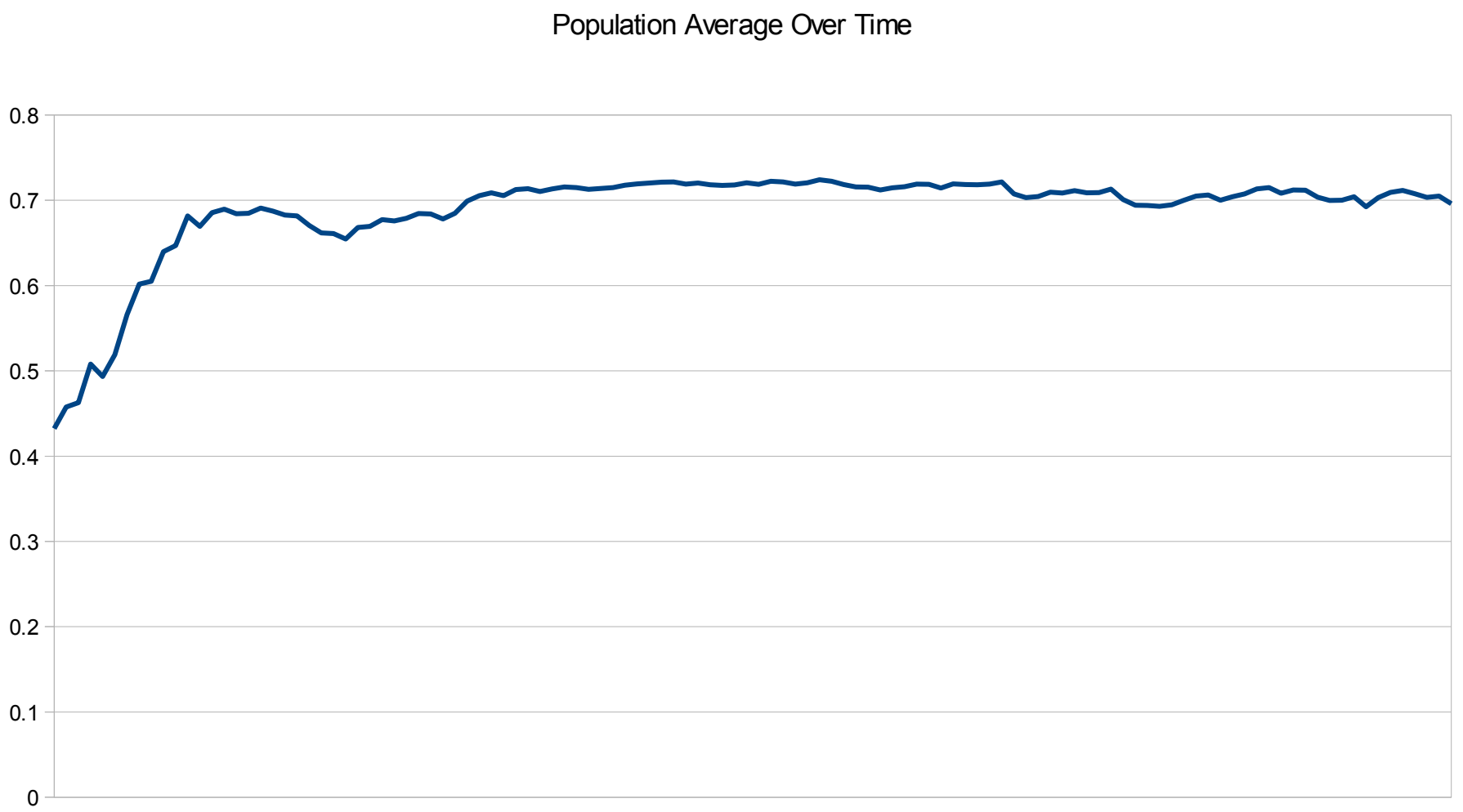
- Mutation method: based on mutation rate, randomly regenerated one random weight of the individual

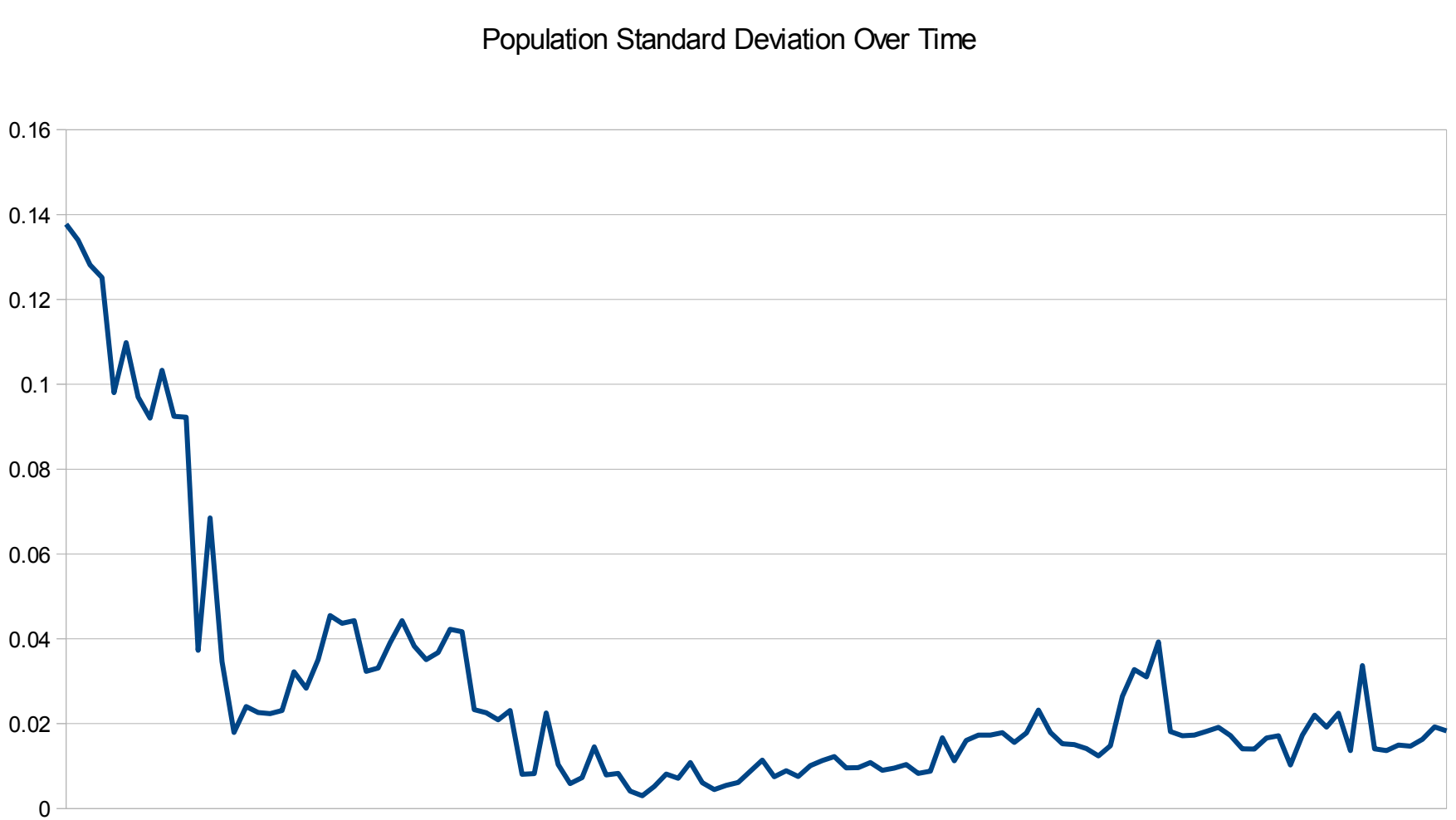
- Termination criteria:
  - Timeout (optional)
  - Fitness threshold

Experiment

- 1 hidden layer node solution using entire data set to train
- Weka → 84% accuracy in ~1m
- Ran genetic algorithm for 4.5 hours (pop_size = 20, mutation_rate = 0.3), best solution was 74%


Population Max Over Time
0.76
0.74
0.72
0.7
0.68
0.66
0.64
0.62
0.6
0.58


Population Average Over Time
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


Population Standard Deviation Over Time
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0

Parallelization

- Took a multithreaded approach
- Create separate threads when reproducing and when evaluating individual's fitness
- (Got a solution for threshold = 0.8 with pop_size = 100 and mut_rate = 0.3 after ~3-4hr but ran into an exception)

Time-limited Comparison

- Ran for 5 minutes (mut_rate = 0.15)

Parallel?	Pop_size	Individuals processed	Best Fit	Pop Fit Avg	Std Dev of Fit
N	24	284	75%	72%	.04
N	50	222	76%	60%	.11
N	74	198	75%	58%	.15
N	100	252	75%	54%	.15
Y	24	793	75%	74%	.005
Y	50	700	75%	68%	.07
Y	74	741	75%	66%	.10
Y	100	800	75%	69%	.06

Time-limited Comparison

- Parallel threads run ~3-4 times faster
  - Makes sense given run on a quad-core

Further Ideas

- Random restart when solutions converge to avoid getting “stuck”
- Would've liked some measure of how the individuals in the population were converging
- Would've worked better for me with a top-down approach (Experimenter-esque interface, more testing, etc)