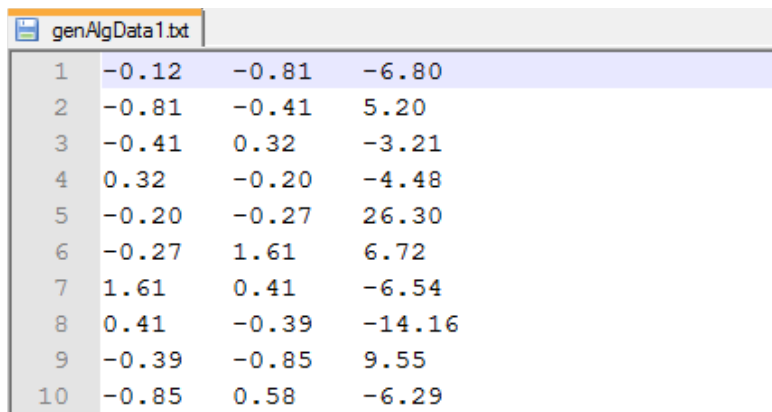


Project 2: Genetic Algorithms

Goal: To gain experience with implementing a genetic algorithm and with the design choices that accompany using one.

Description: Chartists are financial experts who look for recurring shape patterns in stock prices and then use those patterns to predict future stock movements. For this project you will be creating a genetic algorithm to detect 2-day chart patterns in financial data.

Training File Data: You will be provided with two files of data: genAlgData1.txt and genAlgData2.txt. The files each contain training data based on 30 years of price data for the S&P 500. For debugging purposes you are **required** to create small hand-designed data files to test for correctness – you will turn in the testing file(s) as part of your submission.



1	-0.12	-0.81	-6.80
2	-0.81	-0.41	5.20
3	-0.41	0.32	-3.21
4	0.32	-0.20	-4.48
5	-0.20	-0.27	26.30
6	-0.27	1.61	6.72
7	1.61	0.41	-6.54
8	0.41	-0.39	-14.16
9	-0.39	-0.85	9.55
10	-0.85	0.58	-6.29

Each line of the file contains 3 real numbers:

- The first number represents the percentage price change from one day to the next of the S&P 500 stock index.
- The second number represents the percentage price change on the following day.
- The third number represents the profit in dollars that you would have made if you had bought an ETF of the stock market index and held it for one day (negative numbers represent losing money).

Chromosome Encoding: You will build a system that has a population of chromosomes. Each chromosome will be formed from 5 numbers, an example is shown below:

-5.0	0.0	-0.9	3.1	1
------	-----	------	-----	---

- The first two digits, -5 and 0, represent a range. If the first day's price change is between -5% and 0%, then this chromosome will match. For example, on line 1 of the training file, the first value is -0.12%, which falls in the range -5 to 0, so it would be a match.
Note: The coding will be simpler if you enforce the requirement that the first digit be smaller than the second digit (and thus represents the lower bound to the second digit's upper bound). This means that your chromosome will be semi-ordered, so you will need to make sure to keep the ordering constraint satisfied.
- Assuming that the first range matched, then the next two digits, -0.9 and 3.1, represent a price change for the following day. So if the second price change is between -0.9% and 3.1%, then the chromosome fully matches.
- The final digit represents a BUY or SHORT recommendation: 1 means BUY the day after you see the pattern, 0 means SHORT the stock the day after you see it.

Fitness Function: When it comes to computing the fitness of a chromosome, you will sweep through all of the historical data in the data file you're using. You will check each line to see if it matches the pattern specified by the chromosome and then compute the total gain/loss, if the trader had followed the chromosome's recommendation. For example, using the chromosome above on the following data file:

-0.12	-0.81	-6.80
-0.81	-0.41	5.20
-0.41	0.32	-3.21
0.32	-0.20	-4.48
-0.20	-0.27	26.30

- Line 1 matches and the recommendation on the chromosome is 1, meaning BUY, which means we assume the user purchases the stock and loses \$6.80. Thus the total fitness is -6.8 so far. If the chromosome had ended in 0, meaning SHORT, then we assume that the user shorted the stock and made a profit of \$6.8 ($-1 * -6.8$).
- The second line matches and this time the BUY recommendation earns a score of 5.2. Thus the total fitness is $-6.8 + 5.2 = -1.6$.
- The third line matches, leading to a total fitness of -4.81.

- The fourth line doesn't match because 0.32 is outside the range -5 to 0, so this line has no effect on the total fitness.
- The fifth line matches and brings the total fitness to 21.49. (If the chromosome had recommended shorting the stock, then the total fitness would have been -21.49.)
- Notice we don't count anything against the chromosome for the days it doesn't match. However, if there is not a single match anywhere in the data file, assign the chromosome a fitness of -5000 – it's not useful to have a rule that never comes into play.

Note: Don't hardcode the size of the file into your program because the grader will test your code on different data files from the ones provided and they will vary in length.

Initializing the Population: You should randomly initialize the chromosomes as follows:

- The first 4 cells should have their values pulled from a **normal distribution** with a mean of 0 and standard deviation of 1.15. In Python you can use the NumPy library to generate the numbers. Since the first cell has to be smaller than the second cell and the third cell needs to be smaller than the fourth cell, you should check to see whether the randomly generated values need to be swapped to create a valid chromosome.
- The final cell should be assigned 0 or 1 with a **uniform** 50-50 probability.

Be sure to seed your random number generator.

Creating the Next Generation: You will create the next generation using a combination of selection, crossover and mutation. First you will select X of the chromosomes from the current generation to be cloned into the next generation. Then you will use crossover to create the remaining (PopulationSize – X) of the chromosomes for the next generation. Once the new chromosomes have been created, you should iterate over them and with a Z% probability, trigger a mutation on each gene. The values of X and Z will be determined by parameters that should be adjustable – see the later section on parameters.

- **Selection:** You should create code to perform elitist selection and tournament selection – the user will use a parameter to specify which selection algorithm to use. Elitist selection means simply copying the X highest fitness chromosomes from the current generation into the next generation. Tournament selection means holding a tournament X times. In each tournament you will randomly select two chromosomes from the current generation and whichever one has a higher fitness score will be copied into the next generation. You do not need to prevent chromosomes from being selected more than once.
- **Crossover:** You should create code to perform uniform crossover and 1-point crossover – the user will use a parameter to specify which crossover algorithm to use. Assuming you need to create Y chromosomes using crossover (where $Y = \text{PopulationSize} - \text{NumberOfSelectedChromosomes}$), then you will need to do the following Y times: first, randomly select 2 chromosomes from among those that were selected (again you do not need to prevent a chromosome from being used more than once), then ...

- **Uniform:** Iterate over each of the 5 genes and randomly select whether to use the value from the first parent chromosome or the second parent chromosome. Note, that it is possible that the child chromosome could be poorly structured, e.g.

Parent 1	3.5	4.1	-0.9	0.65	0
----------	-----	-----	------	------	---

Parent 2	0.3	1.9	-2.8	-1.7	1
----------	-----	-----	------	------	---

Child	3.5	1.9	-2.8	0.65	0
-------	-----	-----	------	------	---

In this case, the random selections led to the first two genes being in the wrong order relative to each other. In such a situation, you should swap the mis-ordered genes to create a valid child chromosome. (Note that the third and fourth genes happened to come out as a valid range and don't need swapping.)

Child	1.9	3.5	-2.8	0.65	0
-------	-----	-----	------	------	---

- **1-Point:** Take the first 2 genes from the first parent chromosome and the last 3 genes from the second parent chromosome to form a child chromosome. In this case you don't need to worry about creating an invalid chromosome.
- **Mutation:** Once the new chromosomes have been created, you should iterate over each gene in each one of them and with a Z% probability, trigger a mutation. A mutation means that you generate a new random value for that gene, using the same rules as for generating the initial population. Like when you created the initial population, a mutation on one of the first 4 genes could create an invalid chromosome, so you should check whether swapping is required after the mutation.

Parameters: Your code should provide a convenient interface for setting the various parameters that go into your genetic algorithm. Some options for the interface would be: a set of constants at the top of your source code file, a GUI form, or a config file. Whichever interface you use, your README file should clearly explain how to use each parameter and what values are appropriate for it. Your code should validate that the parameter values are appropriate before running the genetic algorithm. The following parameters should be controllable:

- The filename containing the training data.
- The number of chromosomes in each generation.
- The number of generations that your algorithm will run before terminating.
- Which selection algorithm to use (elitist or tournament).

- What percentage of the next generation should be formed using selection – the remainder will be created using crossover. For example, if the parameter is set to 40% and your population size is 150, then you will select 60 chromosomes ($0.4 \times 150 = 60$) to be cloned into the next generation. You will use those 60 chromosomes to create a further 90 chromosomes using crossover, for a total of 150 chromosomes. Mutation will then be applied to randomly selected genes from within those 150 chromosomes.
- Which crossover algorithm to use (uniform or kpoint).
- The initial mutation rate and how much to change the mutation rate each generation – it should be possible to have a fixed mutation rate or to have a mutation rate that gradually decreases (without reaching zero).

In the README file, you should list which combination of parameters achieves the best performance on the training files that you have been provided with.

Output: After each 10 generations your program should display the max, min and average (mean or median is fine) fitness of the chromosomes in the population (which should be straightforward if you sorted your population as part of the selection process). At the end of running all the generations, your code should find the highest fitness chromosome from the final generation and display the chromosome to the screen.

Please provide a clean, easy-to-use interface on your program. It can be command line or GUI, whichever you prefer. It should be easy for the grader to understand how to run and test your code. If you are concerned that any part of your interface may be confusing the grader, then please provide a README file explaining how to use your program.

Coding Requirements: You are expected to choose Python as the programming language for this project.

Your code is expected to follow all good programming practices, i.e. well-commented, broken into functions, thoroughly tested, etc.

Submission: Please zip up the entire contents of your project and submit the zip file via Canvas. At a minimum, the zip file should contain:

- Your source code
- Your README file
- Your hand-created debug training files