Genetic Algorithm for Variable Selection

STAT 243 - Final Project

Fall 2023

Authors: Sean Zhou, Sun Moon, Frederik Stihler

Department of Statistics

Instructor: Christopher Paciorek

December 13th, 2023

1. Introduction

The goal of this project is to develop a genetic algorithm for variable selection, including both linear regression and GLMs. The results are consolidated in a Python package. The user can input a dataset (with covariates and corresponding responses), as well as the desired type of regression. The algortihm will perform the variable selection and tell the user which features to use.

In general, variable selection is the process of selecting a subset of relevant predictors for the model creation. Variable selection generally decreases the model complexity, which can make it easier to interpret and also reduce computational complexity.

Genetic algorithms (GA) are stochastic methods usually used for optimization or search problems. They utilize principles from biological evolution and natural selection, such as selection, crossover and mutation.

2. Programming approach

2.1 Overall structure

We identified the following main steps in the algorithm:

1. Population initialization
2. Evolution cycle for fixed number of iterations
    - Fitness assessment and ranking
    - Parent selection
    - Genetic operators
        - Crossover
        - Mutation
        - etc.
3. Ouput fittest individual of final population

Each (sub-)step was implemented as a modular component. We decided to use an object-oriented programming approach, which means that the individual steps are implemented as Python class methods. To keep the code organized in a logical order we applied inheritance so that each step is defined in its own class. Then, these classes and methods are inherited by the parent class "GA", which has a primary method "select" that carries out the overall algorithm outlined above.

2.2 Default parameter values and initialization

Based on the theory of Givens and Hoeting and the examples outlined below, we have decided on certain default parameters.

The theory suggests a population size P in the range of $C < P < 2C$, where C is the number of predictors (i.e. number of genes per chromosome). We chose the midpoint of this range as default value if no specific population size is given by the user: $P = 1.5 \times C$. As the population size should remain constant during the evolution of the algorithm and due to the crossover operation requiring an even number of chromosomes, an additional chromosome is added if the population size is odd (the user is informed about this modification). The initial population is fully randomized (random binary choice for all loci of all chromosomes).

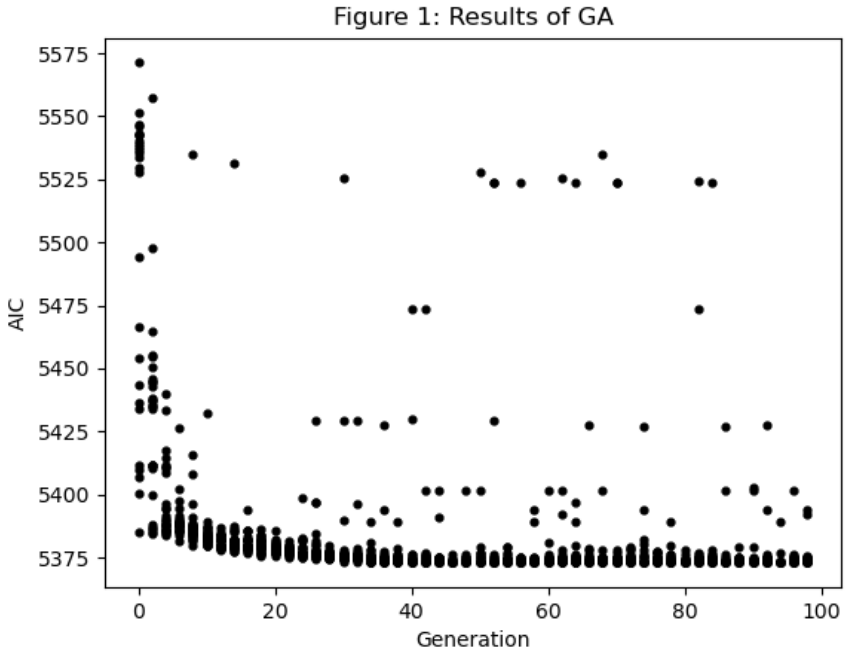
The default genetic operators applied per generation are single split crossover and standard mutation. However, the user can also provide his own genetic operators (implemented on the full population) or modify the order of execution. In addition to the default crossover, a version with random allel selection from the parents is included in the package. The commonly used mutation probability of 1% is set as the default.

3. Testing

4. Results

4.1 Example 1: Baseball data

We first test the algorithm on the same dataset as used by Givens and Hoeting, the baseball salaries.



One hundred generations of size 26 were used. The data set includes 27 predictors for the response variable "salary", so the individual chromosomes consist of 27 binary genes. The genetic operators applied were simple split crossover and a mutation rate of 1% applied independently for each chromosome and each gene position.

5. Contributions

Sun Moon Sun dedicated his efforts to the testing phase, crafting a comprehensive suite of tests to validate the algorithm. His contributions included developing test examples for the overall algorithm and implementing unit tests for individual functions using pytest. Additionally, Sun led the comparison with the Lasso method, offering valuable insights into the algorithm's performance.

Sean Zhou Sean was responsible for structuring and developing core components of the algorithm. His tasks included for example working on the overall GA class, the primary function "select", and parent selection. Moreover, Sean managed the group's GitHub repository, ensuring effective collaboration and organization.

Frederik Stihler Frederik was responsible for structuring and developing core components of the algorithm. He focused for example on essential functions within the algorithm's structure, such

as mutation, crossover, and fitness calculation. In addtion, Frederik took charge of overseeing the creation of the final report, ensuring clarity and coherence.

Finally, every team member performed code reviews of the other peers.