CS 680 Project

A Bootstrap Aggregation Approach to Feature Selection using Genetic Algorithms

William Pugsley - 20824015

Dec. 2023

1 Introduction

A crucial step in the machine learning process is feature selection; determining which parts of a dataset are important in training a given model and discarding the ones that may reduce accuracy or extend training time without contributing to a model's learning. This experiment seeks to explore the application of genetic algorithms to feature selection.

Genetic algorithms (GA) are a type of optimization method inspired by the process of Darwinian evolution, survival of the fittest, and reproduction. All these procedures will arise in a typical GA [GH05]. GAs have been used as a heuristic method to solve linear optimization [JHC13] and discrete optimization problems [CT96]. They also provide a way to search the solution spaces of combinatorial optimization problems or other problems of intractable size [Wol98]. Beyond just being a theoretical curiosity or toy algorithm, GAs have been applied to real-world problems such as antenna design in electromagnetism [WM97].

The application of GAs to feature selection has been explored at length in the past and has seen uses in the medical field [TEC13; Alt+23; LH06]. The majority of a GA's time complexity lies in the 'survival of the fittest' part of the Darwinian analogy, that is, determining which candidate solutions are more optimal [Ank91]. When performing feature selection, this refers to evaluating which subset of features leads to training a more accurate model. Implementations include simple k-fold cross-validation [GC17], introducing a second optimization problem (with its own GA) to reduce the amount of training [Alt+23], or ranking aggregation [TDA22]. This paper will explore implementing a bootstrap aggregating approach to evaluating the fitness of a candidate solution.

2 Genetic Algorithms

Given a dataset $X \in \mathbb{R}^{n \times d}$, we seek a subset of the most important of the d features. Begin by considering a list $L = (L_1,\ L_2, \ldots,\ L_d)$ where the entries, L_i, can only take on binary values indicating whether or not the corresponding feature is to be used. L is equivalent to a chromosome in the Darwinian analogy, encoding all the information that defines an organism's characteristics. The fitness of an organism depends on its genes and is evaluated by the fitness function, f.

At each generation, there are a given number, P, of candidates in our population, Γ. The fittest of these candidates will reproduce to create offsprings with combinations of their parents' genes. As the fittest reproduce, their genes, which are theoretically correlated with better solutions, will be passed onto the next generation. Additionally, there is a small chance that an offspring's genes will randomly mutate, introducing diversity to the candidate population's genes. This step allows GAs to escape local optima. As the final step, a selection of the fittest parents and children will form the population of the next generation. The specific implementation will vary depending on the problem the GA is applied to [GH05]. These procedures

are done at each generation until a maximum number of iterations, t, is reached. The terms generation and iteration may be used interchangeably.

2.1 Fitness & Selection Algorithms

Our fitness function will perform bagging on a subset of X using a machine learning model specified as a hyperparameter of the GA. Given a candidate solution from the population, extract the corresponding features of X, then train a bagging model using m estimators and s samples. The final number that f returns is the bagging model's accuracy on all samples in X, still only testing on a subset of features.

Bagging is a promising implementation of the fitness function because we can control the hyperparameters m and s. Since the time complexity of f will be on the order of m times the ML model's training time complexity, when using models whose training time is of an order higher than $\mathcal{O}(n)$ we can reduce the evaluation time by up-scaling and down-scaling m and s, respectively.

The process of selecting individuals from Γ has multiple possible implementations [IC19]. At its core, it is a function that takes Γ and returns a subset of that population, so any function that accomplishes this is viable. However, there are some functions which are more suitable than others. The individuals in this subset are the parents creating offspring to repopulate Γ at the next generation. Ideally, we want these children to carry on the best genes from their parents. There are five selection algorithms of interest we take from the literature [IC19; GH05; NK97]:

- Random selection: A subset of Γ is selected at random
- Random selection with replacement: A subset of Γ is selected at random with duplicate entries allowed
- Rank-based: Parents are randomly selected with probability proportional to their fitness ranking. Replacement is not allowed.
- Tournament: A random subset of Γ is selected and the most fit member of this subset is selected for reproduction.
- Truncation: Parents are picked in order of most fit.

2.2 Reproduction & Mutation

After selecting the set of parents, two parents are paired at random to create two children by combining their chromosomes. This step requires an additional hyperparameter, c. Each chromosome is split into $c+1$ substrips which are shuffled and recombined (maintaining ordering) [NK97]. Uniform crossover is when $c = d$, at which point every entry of the child's chromosome is randomly selected from either of its parents [IC19]. Fig. 1 is a visualization of this process with chromosomes of length three crossing over at one point.

Our alphabet being binary greatly simplifies the mutation process. Each entry of a chromosome is flipped with probability λ. Some literature defines λ as the probability that *any* entry in a chromosome is altered while others refer to the probability of a single entry [NK97; SD08]. We use the latter definition. This convention is purely notational and only scales λ by d.

This procedure is repeated until we create g children, which will replace the g worst performing individuals in the current generation.

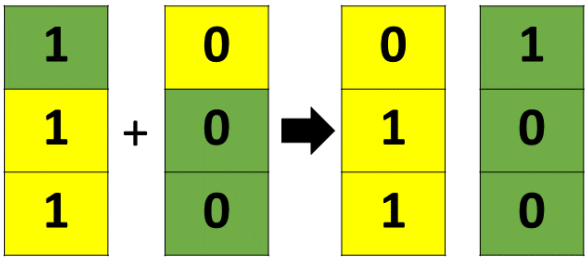


Figure 1: Two parent chromosomes (left) reproduce with 1-point crossover to create two children (right).

2.3 Summary

When combining these processes we are left with several hyperparameters. This section summarizes their definitions and discusses viable values. To begin, P is the population size, the number of individuals in each generation. This is the parameter that will most severely lengthen the running time of the algorithm (for a given dataset). The time complexity of one iteration is $\mathcal{O}(P \log(P) \cdot m \cdot \mathcal{O}(\text{Model Training}))$ [Ank91], where the model will be trained on s samples. Since our alphabet is binary and the data we input has d features, there are 2^d possible combinations of candidate solutions. So long as $P \cdot t2^d$, GAs are more efficient than an exhaustive search. In practice, P is limited to between 10 and 200 [GH05].

The remaining hyperparameters determine the convergence rate of the GA. The mutation rate, λ, is what allows genetic algorithms to escape local optima and explore a large swath of the parameter space. However, a large λ may cause the population to vary too quickly, skipping actual optimal solutions or corrupting important information stored in each chromosome. $0.01 \leq \lambda \leq 0.05$ is a typical range [GH05] but we may extend the upper bound to 0.1 for the sake of a robust search. g is the number of children created each generation and can range from 1 to P. The number of crossover points ranges from 1 to d; in situations where premature convergence is an issue, choosing a larger c can resolve the problem [IC19].

A summary of this algorithm is presented in Algorithm 1 and a list of variables and their definitions may be found in Appendix A for quick reference.

3 Results

Table 1 presents the test accuracy of running three classifiers on MNIST with all 784 pixels as features. It also shows the test accuracy of the same classifiers trained on the data with features selected through our GA. We

Algorithm 1 Genetic Algorithm for Feature Selection

Input: $X \in \mathbb{R}^{n \times d}$, t. **Hyperparameters**: P, λ, g, c, m, s, SelectParentsAlgorithm, model

$\Gamma \leftarrow$ initialize population with random individuals

$F \leftarrow \text{Fitness}(\Gamma,\ \text{model},\ m,\ s)$ ▷ Evaluate fitness of the population

for i in $\{1,\ ...,\ t\}$ **do**

  $\Gamma \leftarrow \text{sort}(\Gamma)$ ▷ Sort individuals in the population based on the corresponding values in F

  $\Phi_p \leftarrow \text{SelectParentsAlgorithm}(\Gamma)$

  for parents in Φ_p **do**

    $\Phi_c \leftarrow \text{CreateChildren}(\text{parents},\ c)$

  end for

  $\Phi_c \leftarrow \text{Mutate}(\Phi_c,\ \lambda)$

  $\Gamma[P - g :] \leftarrow \Phi_c$ ▷ Replace least fit individuals with new children

  $F \leftarrow \text{Fitness}(\Gamma,\ \text{model},\ m,\ s)$

  Record minimum value in F

end for

Return estimate for optimal value/solution

run a random search on the hyperparameter space for all models, except for the nearest neighbours classifier where we use a grid search. We can use the results from the hyperparameter search on the model trained on the full dataset when inputting our training model to the GA. This gives us a rough first pass on the hyperparameters, which will affect which features our GA selects as important. The second hyperparameter search is necessary to refine our model further. For example, it is possible that the optimal max depth of a decision tree trained on the reduced dataset is lower than on the full data. The hyperparameters we use are $P = 10$, $\lambda = 0.05$, $c = 784$, $g = 4$, $m = 80$, and $s = 1000$ with a rank-based selection algorithm. We run the GA for $t = 30$ iterations, resulting in a total of $31 \cdot P$ calls of the fitness function.

(MNIST)	Accuracy with all Features	Features Selected (/784)	Accuracy with Selected Features	Running Time (s)
Decision Tree	0.8836	365	0.8807	6,568
Naive Bayes	0.837	413	0.8245	6,863
3-NN	0.9705	421	0.9626	28,032

Table 1: Testing accuracy of various classification models when trained on the original MNIST dataset and when trained on the most important features, as determined by a GA.

(FASHION MNIST)	Accuracy with all Features	Features Selected (/784)	Accuracy with Selected Features	Running Time (s)
Decision Tree	0.8113	393	0.8014	9,253
Naive Bayes	0.6654	384	0.6636	5,641
4-NN	0.8577	406	0.8535	35,892

Table 2: Testing accuracy of various classification models when trained on the original Fashion MNIST dataset and when trained on the most important features, as determined by a GA.

The third column displays how many features the GA selected as important, as determined by its most accurate candidate solutions across all generations. Each of the three classifiers performed as well when trained on the entire MNIST dataset or when trained on the selected subset. The test accuracies are within 1% of each other. This is the mark of a successful feature selection algorithm; we discarded a significant number of features, more than half in the case of a decision tree, and maintained our classification accuracy. The running time in the last column is the amount of computing time it took the GA to reach 30 iterations, from its first initialization to its last pass of f. This time does not include the hyperparameter search or training the model on the subset of data after the GA returns its solution.

The number of features the GA selected during each run of the experiment is different, indicating that our feature selection method is model-dependent, but not conclusively. Since genetic algorithms are necessarily stochastic the number of features selected, and which features, will likely be different at the end of every run of the GA unless it runs for an impractically high number of iterations.

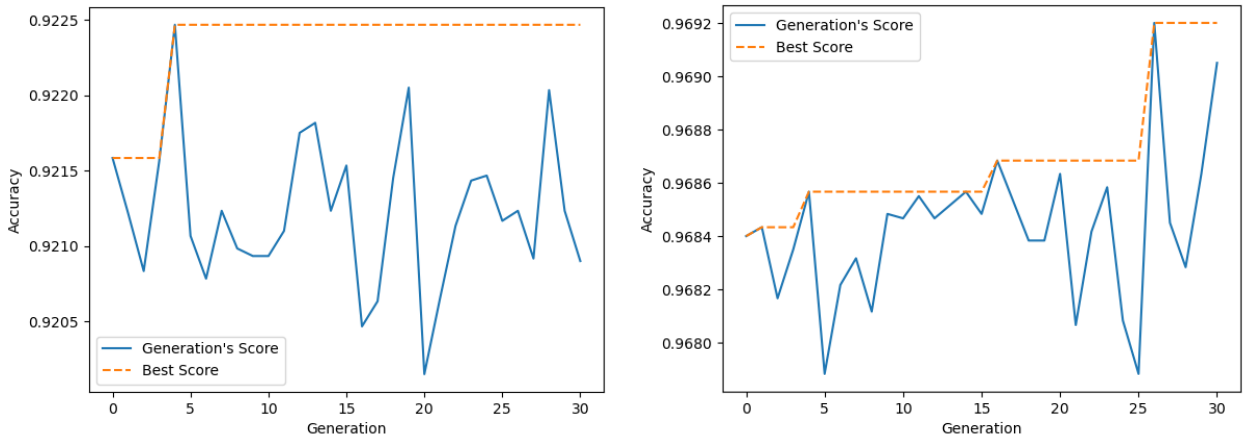
Table 2 shows the results of the same process as Table 1 when using the Fashion MNIST dataset. All our conclusions regarding Table 1 also apply here. This provides evidence that using a bagging method for our fitness function is valid and does not just happen to find a good solution for one dataset.

While our results are certainly exciting and show that our algorithm is capable of achieving the desired results, it is not the only algorithm to do so. In particular, using the same GA process with cross-validation as a fitness function works similarly well [GC17]. Table 3 shows the results of using a 5-fold cross-validation based genetic algorithm to select features from the MNIST dataset. The same hyperparameters as in Table 1 are used. This GA selects a nontrivial subset of the original data and maintains test accuracies before and after feature selection, although the difference in accuracy for the decision tree is now $\sim 1.5\%$ instead of 1%.

Using a bagging based fitness function speeds up the genetic algorithm when using decision trees for classification, but not a naive Bayes or nearest neighbours classifier. Bagging appears to run faster than cross-validation for models with longer training times, as the decision tree model is the only one whose training time is worse than linear in n. Testing complexity, on the other hand, has the opposite effect

(MNIST)	Accuracy with all Features	Features Selected (/784)	Accuracy with Selected Features	Running Time (s)
Decision Tree	0.8894	407	0.8741	7,313
Naive Bayes	0.837	378	0.8329	1,786
3-NN	0.9705	396	0.9653	15,260

Table 3: Testing accuracy of various classification models when trained on the original MNIST dataset and when trained on the most important features, as determined by a GA with cross-validation.



(a) 3NN. Bagging, $m = 80$, $s = 1000$

(b) 3NN. 5-fold cross-validation

Figure 2: The GA's estimate for the optimal solution at each generation using a 3-nearest neighbours model. The solid line shows the highest accuracy out of all members in Γ. The dashed line gives the best accuracy out of any previous generation; this is the result the GA will return when it finishes running.

and slows down the bagging GA. For instance, bagging does not speed up the fitness function for KNN classification; we classify $60,000$ training points by running 80 models trained on $1,000$ points, with at worst 784 features. This is a $\mathcal{O}(3.8 \cdot 10^{12})$ computation. Whereas for 5-fold cross-validation, we classify $12,000$ points using $48,000$ training points and at worst 784 features, repeating five times. This computation is $\mathcal{O}(2.2 \cdot 10^{12})$, where this order has the same multiplicative constant as the bagging method. This is to be expected since the bagging method breaks up the training phase into small, fast chunks and measures the training accuracy on the complete training set, while cross-validation uses much larger training subsets and smaller test batches.

It is also worth investigating the behaviour of the GA as it progresses. Fig. 2 shows the best estimate the GA calculates at each generation for both bagging and cross-validation using a 3-nearest neighbours model. At first glance, it would appear that the bagging-based method is not even optimizing when compared to the

cross-validation approach; it finds some solution within a few generations before remaining roughly constant. However, the cross-validation method is not much better upon closer inspection. The range of accuracies is very small, around 0.1% for cross-validation and 0.2% for bagging, which is very likely due to the random nature of bagging and cross-validation rather than the GA exploring the parameter space to find an optimal solution. Furthermore, while the cross-validation GA reaches its optimal estimates after more generations and in smaller steps, the blue line in Fig. 2b still appears to be randomly distributed around some constant with an increasing deviation. This leaves us with a puzzle: the GA does not appear to be optimizing, but the output is still suitable and leads to good results as seen in Tables 1 to 3. One explanation is that the GA is very quickly reaching a local minimum since MNIST has many features with little to no information: the GA initializes, quickly pools together some pixels from the center of the image where most numbers are written, and then adds/removes the extraneous features without improving or worsening the accuracy. This would indicate that our choice of dataset is not wholly compatible with this feature selection method. However, this conclusion is not supported by the results using Fashion MNIST, which exhibits the same behaviour.

4 Conclusion

Genetic algorithms using a bootstrap aggregating approach to implement the fitness function are successful for feature selection. They can select subsets of a dataset, sometimes in a relatively short number of iterations, which only slightly adversely affect the model's prediction capability. While using a bagging fitness function in our genetic algorithm is not a perfect solution, as demonstrated by the comparison to cross-validation and the peculiar behaviour in Fig. 2, it is another viable technique to consider when choosing to approach feature selection using a GA. In addition, there are countless variations to explore and test. It is also worth noting that while this report focuses on classification, the same algorithm can be used to perform feature selection in a regression setting. This involves only changing the fitness function appropriately to reflect the new task.

One possible improvement is to record unique candidate solutions and their corresponding fitness scores in a dictionary to avoid recomputing previously visited points. However, it is not obvious that this is a valid approach given the random nature of our fitting function; the measured accuracy may, by chance, be too high or too low. Given the scale of the y-axis on Fig. 2, this is likely an important factor. There is no logical reason why we should take the first bagging run and not any later ones. Furthermore, we would only be trading time complexity for spatial complexity. Currently, our genetic algorithm takes $\mathcal{O}((P+1)\cdot d)$ space to store each candidate solution in Γ as well as the optimal solution. This memoization approach would, in the worst-case scenario where each candidate solution is unique across all generations, multiply this order by t. For datasets with a very large d, or optimization runs with a high t, this may not be preferable.

The term 'genetic algorithm' encompasses a varied array of tools and methods with broad applications. This report has explored one of these, so far overlooked, implementations. Although the results are not conclusive and raise more questions, that is the nature of experimentation and should be viewed as an invitation to explore further rather than a discouraging end.

References

[Alt+23] Mohammed Ghaith Altarabichi et al. "Fast Genetic Algorithm for feature selection — A qualitative approximation approach". In: *Expert Systems with Applications* 211 (2023), p. 118528. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.118528. URL: https://www.sciencedirect.com/science/article/pii/S0957417422016049.

[Ank91] Carol Ann Ankenbrandt. "The Time Complexity of Genetic Algorithms and the Theory of Recombination Operators". PhD thesis. Tulane University, Computer Science Dept. School of Engineering New Orleans, LA, United States, 1991.

[CT96] Jianbo Cai and Georg Thierauf. "Evolution strategies for solving discrete optimization problems". In: *Advances in Engineering Software* 25.2 (1996). Computing in Civil and Structural Engineering, pp. 177–183. ISSN: 0965-9978. DOI: https://doi.org/10.1016/0965-9978(95)00104-2.

[GC17] Thineswaran Gunasegaran and Yu-N Cheah. "Evolutionary cross validation". In: *2017 8th International Conference on Information Technology (ICIT)*. 2017, pp. 89–95. DOI: 10.1109/ICITECH.2017.8079960.

[GH05] Geof H. Givens and Jennifer A. Hoeting. *Computational Statistics*. Wiley, 2005.

[IC19] Savio D Immanuel and Udit Kr. Chakraborty. "Genetic Algorithm: An Approach on Optimization". In: *2019 International Conference on Communication and Electronics Systems (ICCES)*. 2019, pp. 701–708. DOI: 10.1109/ICCES45898.2019.9002372.

[JHC13] Weihua Jin, Zhiying Hu, and Christine W. Chan. "A Genetic-Algorithms-Based Approach for Programming Linear and Quadratic Optimization Problems with Uncertainty". In: *Mathematical Problems in Engineering* 2013 (Dec. 2013), p. 272491. ISSN: 1024-123X. DOI: 10.1155/2013/272491.

[LH06] Michael Lecocke and Kenneth Hess. "An Empirical Study of Univariate and Genetic Algorithm-Based Feature Selection in Binary Classification with Microarray Data". In: *Cancer Informatics* 2 (2006), p. 117693510600200016. DOI: 10.1177/117693510600200016. eprint: https://doi.org/10.1177/117693510600200016. URL: https://doi.org/10.1177/117693510600200016.

[NK97] Hung T. Nguyen and Vladik Kreinovich. "Genetic Algorithms:"Non-Smooth" Discrete Optimization". In: *Applications of Continuous Mathematics to Computer Science*. Dordrecht: Springer Netherlands, 1997, pp. 153–173. ISBN: 978-94-017-0743-5. DOI: 10.1007/978-94-017-0743-5_8. URL: https://doi.org/10.1007/978-94-017-0743-5_8.

[SD08] S.N. Sivanandam and S.N. Deepa. *Introduction to Genetic Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-73190-0. DOI: 10.1007/978-3-540-73190-0_7.

[TDA22] Bui Quoc Trung, Le Minh Duc, and Bui Thi Mai Anh. "A Hybrid Approach Based on Genetic Algorithm with Ranking Aggregation for Feature Selection". In: *Advances and Trends in Artificial Intelligence. Theory and Practices in Artificial Intelligence*. Ed. by Hamido Fujita et al. Cham: Springer International Publishing, 2022, pp. 226–239. ISBN: 978-3-031-08530-7.

[TEC13] Chih-Fong Tsai, William Eberle, and Chi-Yuan Chu. "Genetic algorithms in feature and instance selection". In: *Knowledge-Based Systems* 39 (2013), pp. 240–247. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2012.11.005. URL: https://www.sciencedirect.com/science/article/pii/S0950705112003140.

[WM97] D.S. Weile and E. Michielssen. "Genetic algorithm optimization applied to electromagnetics: a review". In: *IEEE Transactions on Antennas and Propagation* 45.3 (1997), pp. 343–353. DOI: 10.1109/8.558650.

[Wol98] Laurence A. Wolsey. *Integer Programing*. Wiley, 1998.

A Notation

- $X \in \mathbb{R}^{n \times d}$: The dataset.
- $\Gamma \in \mathbb{R}^{P \times d}$: Population, the set of candidate solutions at a given generation.
- P: Population size, the number of candidate solutions to consider at each generation of a GA.
- λ: Mutation rate, the probability that a given entry of a chromosome will flip during mutation.
- t: Maximum number of iterations/generations to run the GA for.
- $f : \mathbb{R}^d \to \mathbb{R}$: Fitness function, the function used to determined the viability of a candidate solution. Either bootstrap aggregating or cross-validation in this paper.
- m: Number of models to train when bagging.
- s: The number of samples each bagging model will train on.
- c: The number of crossover points to use when performing recombination.
- g: The number of children to generate at each generation.