Feature Selection Using Genetic Algorithms

Sriram Ranganathan
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
s22ranga@uwaterloo.ca

***Abstract*—In this paper, a specific application of the genetic algorithms in feature selections to prepare training data for the machine learning model is discussed. The computational complexity of feature selection is directly proportional to the number of features present in the dataset. Although the brute force method for solving this selection problem guarantees convergence, the time it takes to attain the convergence exponentially increases with the number of features present. Hence, this study highlights an essential aspect of natural selection using genetic algorithms utilized in feature selection. We examine the accuracy attained with and without feature selection using this strategy in conjunction with random forest and Support Vector Machine classification techniques. For this investigation, publicly accessible datasets (arrythmia, breast cancer, wine quality) were used. There was 9% increase in SVM classification accuracy when the model was trained using the feature subset of arrythmia dataset obtained by feature selection.**

Keywords—Genetic algorithms, Feature selection, Artificial life, Natural selection, Machine Learning, Random Forest, Support Vector Machine

I. INTRODUCTION

The idea of evolution through natural selection is fundamental to biology, and this has had a profound impact on how we see biological systems. Many artificial life models are based on evolutionary computation models called genetic algorithms. Both as instruments for tackling real-world issues and as scientific models of evolutionary processes, genetic algorithms have been employed. Genetic algorithms tend to mimic the evolution of the biological system to optimize a computationally hard problem. These algorithms involve crossover, mutation, variations, and filtering populations based on fitness (survival of the fittest) which can be closely related to a chromosome selection, variations, and mutations involved in a biological system. The use of genetic algorithms started as an idea of how biological systems evolve into a more optimized systems in comparison to the previous generations.

Natural selection is the process through which populations of living organisms adapt and shift. A population's members are naturally diverse, which means they vary in certain ways. This variation shows that some individuals have characteristics that are more suited to their surroundings than others. Those with different traits provide them with an advantage—are more likely to survive and procreate. The descendants of these individuals, therefore, inherit their capacity for adaptation. Over time, these advantageous traits become more prevalent in the populations. Natural selection results in beneficial traits being passed down through the generations. It may cause a species to produce fully unique individuals, which would then cause evolution.

The concept of using evolutionary processes to solve optimization issues in engineering dates back to the 1950s and 1960s, when computer scientists first began to explore this topic. Three significant distinct implementations of evolutionary computing resulted from this, two of which being genetic algorithms and evolution strategies[2].

John Holland pioneered the use of genetic algorithms for fixed length character strings in 1975 which was clearly mentioned in his book Adaptation in Natural and Artificial Systems[3]. In [3] John Holland described genetic algorithms as an approximation of the evolution through natural selection. Holland's GA technique involves selection, crossover, mutation processes to move from a generation of chromosomes to another generation. Holland referred each chromosome in a given generation to be candidate solution. Selection is the process of filtering chromosome by the fitness for crossover(associated to "Survival of the fittest" Darwinian evolution). The crossover operation is similar to the biological reproductivity wherein the parts of the selected chromosomes are exchanged to create new chromosome. Mutation is the process of randomly changing the values of chromosome. Mutation allows the algorithm to explore the different areas in the solution space. In addition to mutation John holland also proposed Inversion technique which randomly reverses a subset of the chromosome.[6] Due to the computational complexity of the inversion operation, In this study Genetic algorithm without Inversion is used.

In [7] a genetic programming technique which searches for the optimal computer program is described. John Koza created genetic algorithm to generate programmes in 1992 that could carry out certain tasks. The approach was called "genetic programming" (GP). Many difficult problems of machine learning and artificial intelligence can be reduced to a search problem, where a computer program generates desired outputs for inputs is searched.[7]

. The next section discusses about the background of genetic algorithm, how it was formulated based on the biological evolution and the concept of feature selection. It also discusses how genetic algorithm can be used to find an optimal feature subset that would yield better training model for a dataset.

II. BACKGROUND

A. *Genetic Algorithm Overview*

Genetic Algorithms are nature-inspired search algorithms shaped from Darwin's Theory of Evolution. It is a stochastic method for function optimization that is rooted in natural genetics and evolutionary theory. In nature, organisms' genetic traits tend to evolve over subsequent generations to better conform to their surroundings. The genetic algorithm is a heuristic optimization method that employs natural evolution processes. Genetic algorithms may create high-quality solutions for a variety of problems, including search and optimization, by imitating the processes of natural selection, reproduction, and mutation. Genetic algorithms work on a population of individuals to generate greater accurate approximations with every generation. In each generation, the method generates a fresh population by choosing individuals depending on their fitness level in the given problem. These individuals are then cross - linked using random single point or two-point crossover to form offspring. The offspring may undergo genetic change or a mutation where in a randomly chosen trait or the attribute of the offspring is changed randomly. The flowchart in figure 1 shows the process of a simple genetic algorithm.

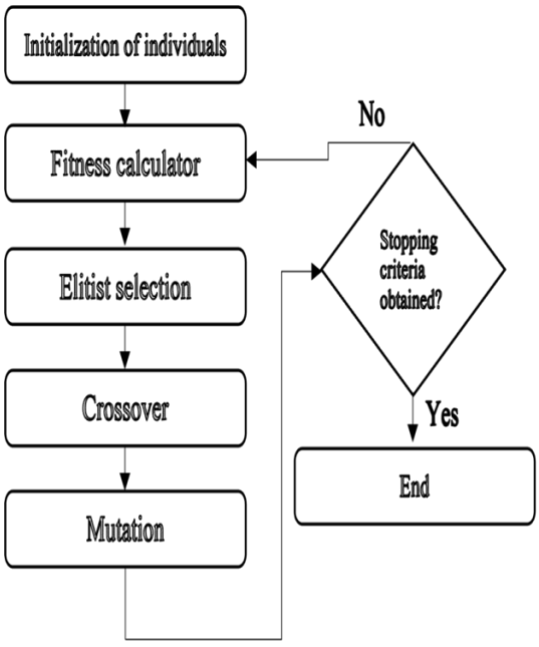

Figure 1 Genetic algorithm flow

The initiation of potential solutions takes place at random. The fitness evaluation of each potential solution follows this operation. Elite individuals are chosen based on the fitness value calculated for each candidate. In most cases, the elite individuals are immediately added to the following generation. The fitness value is used to determine the mating pool. The parents for the crossover procedure can be chosen using a variety of selection techniques. The individuals with the greatest fitness scores may be chosen using a straightforward procedure. Another approach uses a roulette wheel and is known as fitness proportionate selection; in this approach, candidates are chosen based on their overall likelihood [9]. The likelihood of the individuals with lower fitness values entering the mating pool will be low. The fitness proportionate selection has a high likelihood of locating the best solutions because, unlike the first technique, it gives candidates with lower fitness values a chance to contribute to the next generation rather than eliminating them. The roulette wheel-based selection is shown in Figure 2 [9].

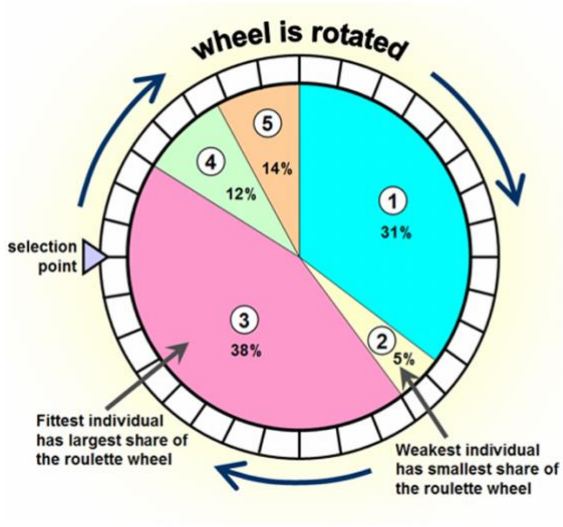

Figure 2 Roulette wheel selection [9]

Equation (1) is used to calculate the cumulative probability $P_c(i)$ of i^{th} individual. Here $f(i)$ represents the fitness value of i^{th} candidate, n represents the number of candidates present in the generation.

$$P_c(i) = {f(i)}\Big/{\sum_{j=0}^{n} f(j)} \qquad (1)$$

The crossover procedure is subsequently performed, and the crossover probability determines whether or not the crossover between parents occurs. The crossover operation takes two parents from the mating pool as input and produces offspring. The degree of inheritance by children generated by crossover from either parent is random mimicking biological reproductivity. There are various crossover procedures like single point, multi point and gaussian crossover, however the two point crossover is used in this study. The two-point crossover approach is described in Figure 3.

The crossover process is followed by mutation, which includes changing the values in the offspring at random locations. The mutation rate determines whether mutation occurs or not. This procedure is performed

over and again until the best and most optimum solution is discovered.

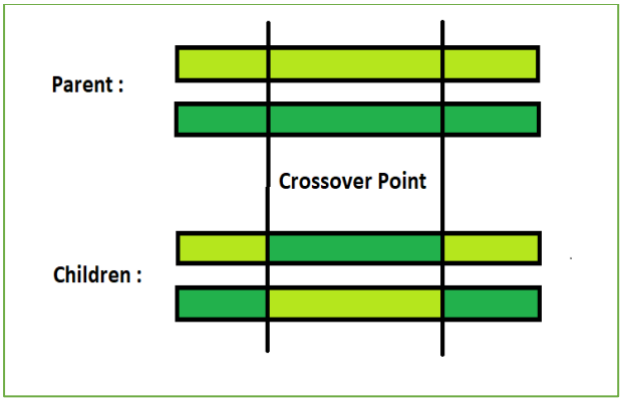

Figure 3 Two Point Crossover

The genetic algorithms have been used extensively in solving many NP-complete problems like the knapsack problem, Travelling salesman problem.

B. *Feature Selection*

Feature selection and dimensionality reduction is a vital task in data mining and data exploration. The problem of feature selection is NP-hard because finding the globally optimal solution takes an exponentially increasing amount of time. All classical feature selection algorithms use heuristic rules to find locally optimal solutions, and their solutions perform differently on different datasets [10]. The number and quality of features used to represent each instance of datapoint influence the success of subsequent classification [11]. As stated in [12], extensive research has been conducted in Nature Inspired Algorithms tailored to solve the feature selection problem. This review paper [12] provides a concise overview of relevant papers that address the feature selection problem using nature-inspired algorithms.

One of the initial and primary challenges of classification or regression machine learning tasks is an optimal feature selection because the recent improvement in resource accessibility has resulted in the accumulation of data in databases and data warehouses in which most of it are irrelevant or redundant. Using redundant and irrelevant features in a machine learning task has many drawbacks, such as the possibility of overfitting the learning algorithm, an algorithm with a poor test accuracy [13]. Feature selection is the process of obtaining the optimal subset of the initial feature vector which will best represent the features but in a lower dimension. The redundant feature in the feature space creates a symmetry, so feature selection technique intuitively reduces the no. of features also reducing the symmetry in the feature vector [5]. Each feature in the subset of the feature vector is highly unrelated with the other features in the subset, training a machine learning model using this subset of features as training data will result in a more accurate model.

C. *Genetic Algorithms in feature selection*

Feature selection is an optimization problem and genetic algorithms are very good optimizers inspired from the nature's evolution. Hence, Genetic algorithms are used in this study to address the problem of feature selection. In [8], a GA was used to find the best binary vector, where each bit corresponds to a feature; if the nth bit is 1, it is chosen for the classifier's training, otherwise, it is dropped. The fitness of the feature subset obtained from the genetic algorithm is evaluated by determining the accuracy of the KNN model created with the feature subset. The strategy was expanded in [14] where real valued coefficients were used in place of binary bits. All features in the dataset were normalized and then linearly scaled by the corresponding co-efficient, also if a feature's coefficient is 0 the feature is dropped. Figure 4 shows an example of how the feature selection vector is represented as a chromosome with binary values. In this each bit corresponds to a feature.

Figure 4 Feature selected vector

Although Genetic Algorithm does not guarantee convergence, but when compared with a traditional search algorithm genetic algorithm has better chance of convergence to a global minima or maxima. Also, Genetic Algorithm provides a different optimal solution as it operates on the population of solutions which evolve into optimal solutions in parallel. As a result, unlike traditional approaches that search from a single solution, it can avoid being locked in a local optimum solution. The paper [15] discusses the topic of feature selection, where GA has been proven to be a successful approach for an optimal solution. They employed a master slave Parallel Genetic algorithm with Hadoop MapReduce to boost performance, and they proposed a wrapper strategy employing a KNN classifier for guided feature selection. The only problem when using a genetic algorithm is the representation of the possible solutions and determining a fitness evaluator. If the solutions can be represented as chromosomes, then genetic algorithms can be useful searching for the best solution. In the feature selection problem, the best optimal solution is a subset of the main feature vector. The effectiveness of the feature subset can be evaluated by training a model with the feature subset and the score of validation can be the fitness. Since the feature selection problem is a search problem where the solutions can be represented as chromosomes upon which the fitness evaluation can be performed, genetic algorithms can be found useful for this search and optimization problem.

III. TRAINING AND TESTING DATASETS

In this research, the use of genetic algorithm for the purpose of feature selection in 3 datasets is discussed. Table I shown below is the description of the datasets

and their features and the type of Machine Learning (ML) task.

TABLE I. DATASETS

Dataset	No. of instances	No. of attributes	ML task type	No. of Output classes	Type of Classifier
Arrhythmia [17]	452	279	Classification	13	SVM
Breast Cancer Wisconsin (Diagnostic) [18]	569	30	Classification	2	Random Forest Classifier
White Wine quality [19]	4898	11	Classification	7	Random Forest Classifier

TABLE II. VALIDATION ACCURACY

Dataset	Validation Accuracy	Hyperparameters	Type of Classifier
Arrhythmia [17]	72.52%	Kernel: Polynomial Regularization: 1	SVM
Breast Cancer Wisconsin [18]	93.8%	Maximum depth: 50 Estimators: 200	Random Forest Classifier
White Wine quality [19]	68.57%	Maximum depth: 50 Estimators: 200	Random Forest Classifier

The datasets mentioned in Table I are split into training (80%) and validation (20%). Before splitting the datasets into training and validation, these datasets were subjected to preprocessing. These datasets contain many null values which were replaced with the average value of the corresponding columns. To compare the results of the feature selection, the validation accuracy of the model trained without feature selection is calculated. Table II shows the validation accuracy of the models obtained by training datasets.

IV. METHODOLOGY

Figure 6 depicts the suggested process. The genetic algorithm developed for the experiment is discussed in length in this section.

A. Representation of candidate solutions

Each possible solution is often represented as an array of bits (also called bit set or bit string). The fundamental advantage of these bit string representations is that their portions are readily matched to the feature set size, allowing for easy genetic operations. In this implementation of Genetic algorithm, the first and initial step involves encoding features into a binary vector stream (same as [8]). Each candidate solution for a dataset consists of a binary vector that has length same as the number of attributes in the dataset. Each feature corresponds to location in binary vector. Figure 5 demonstrates an example candidate (chromosome) solution generated for the wine quality dataset. If the value of the binary vector is 1 then the feature is selected and if the value is 0 the feature is dropped for training.

Attributes	Chromosome
fixed acidity	0
volatile acidity	1
citric acid	0
residual sugar	0
chlorides	1
free sulfur dioxide	1
total sulfur dioxide	1
density	1
pH	0
sulphates	0
alcohol	0

Figure 5 Example chromosome

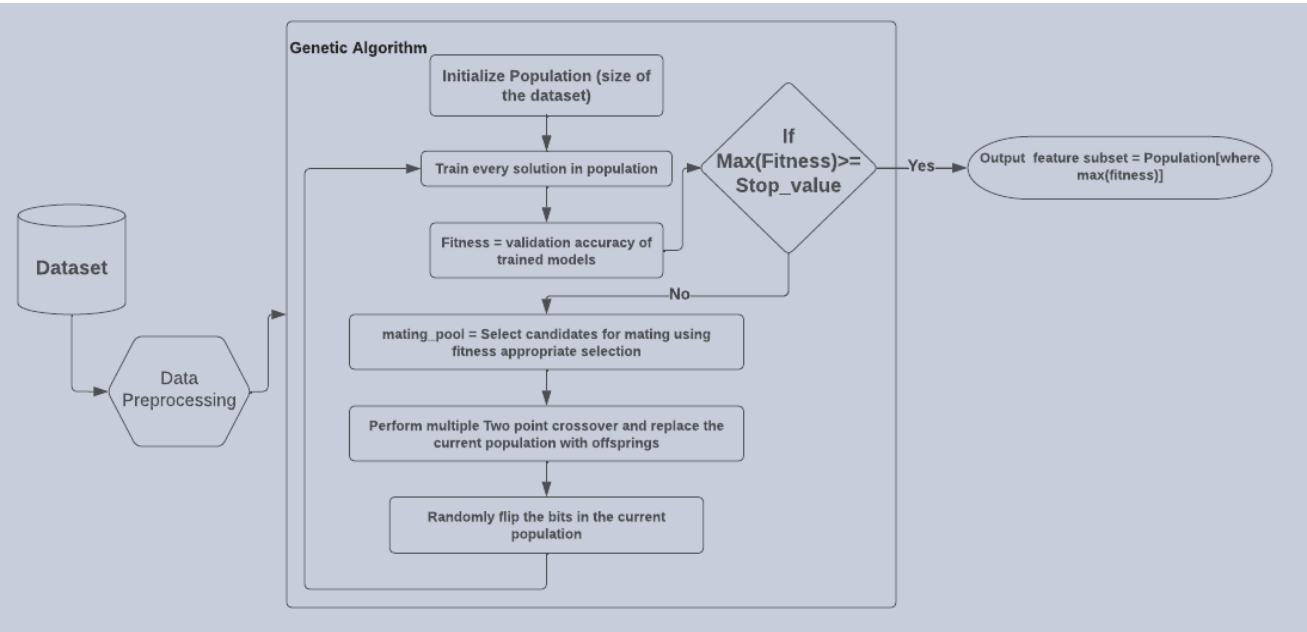

Figure 6 Proposed Methodology

B. Initializiation of population

The initial population is created by initializing the random binary vector of size n (where n is the number of features) for m times (where m is the population size). The initial population can be decided by the number of features present. The population size decides solution space considered in each generation. The population does not contain copies of the same individual (solution) to widen the search space at every generation.

C. Fitness function

The genetic algorithm proposed in the methodology considers validation accuracy of the models trained with the candidate solution as fitness. Therefore, to calculate the fitness value of an individual candidate solution, the feature vector is masked by the candidate solution (Binary vector) and then the model with the new feature subset obtained from masking. Once the model is trained with the training data the model is checked with the validation data. Validation accuracy is the measure of how correctly the model classifies the validation data. The model trained for Arrhythmia dataset is SVM, for Breast cancer dataset it is Random Forest and for wine quality it is Random Forest.

D. Mating Pool Selection

Selection is the next step following initialization of the population. Mating pool consists of the parents with the highest fitness values. Since validation accuracy is the fitness score, higher the value of validation accuracy higher is the fitness of the individuals. In this report for the comparative study, two selection methods were implemented, which are mentioned below.

- Fitness proportionate selection: This method (as mentioned in II. Background section) involves selecting parents based on their cumulative fitness values. Equation (1) represents the calculation of cumulative probability.
- Max fitness selection: This method involves random selection of candidates with top values of fitness.

Selection also involves setting mating pool size; the mating pool size must be smaller than that of the initial population to remove the worst candidate solutions. In this experiment the size of mating pool is set to one third the size of the population. For example, in the Max fitness selection, If the population size is 100 then top 33 candidates are added to the mating pool. But in roulette wheel selection the candidate with lower fitness will be selected with low probability and the candidates with higher fitness will be selected with higher probability, this facilitates a large solution space. Fitness proportionate method, the algorithm takes a bad step and widens the search space, this bad step can help attaining a global minimum.

E. Offspring production

Offspring production involves producing new candidate solutions by performing crossover between the parent solutions added in the mating pool. Crossover imitates the biological reproductivity. This function takes two different candidate solutions from the mating pool and two random indices as input. The genetic bits of the parent solutions present between the indices is exchanged to produce two new solutions. This crossover operation is governed by the crossover probability Cp. Higher the crossover probability higher the chance of the two-point crossover happening between the parents. For any genetic algorithm the cross over parameter must be set high to prevent multiple copies of the parents getting added to the next generation. Also, in Nature the crossover of genes between the parents happens with high probability. Before getting added to the new population the produced offspring go through mutation.

F. Mutation in offsprings

Once the offspring are produced, they are subjected to mutation. In mutation the bits from the offspring are chosen at random and flipped intentionally. In nature mutation is significant because it leads to a whole new DNA sequence from a particular gene. This facilitates evolution, but in nature mutation happen at very low rate. In humans' mutation rate is 10^{-6} per gene per generation [16]. The mutation is governed by the probability of mutation. The mutation probability is set very low to mimic the nature's mutation rate. The new generated offspring after going through the process of mutation gets added to the new population. Before getting added to the new population, the new population is checked for prior existence of a copy of the offspring. If there exists a redundant copy of the offspring in the new population, the offspring is not added. If there are redundant candidates in a population, the consecutive new generations produced will be the same which will result in a loop of same candidates produced. The production of offspring and the mutation occurs until the size of the new generation is same as that of the previous generation.

G. Evolution of generations

The fitness calculation is followed by the crossover and mutation operation this is repeated until best optimal solution is produced. The termination of the above loop can be done in two ways,

1. The algorithm evolves for a certain number of generations (if explicitly mentioned) and then terminates. In this case the top solutions from all the generations is stored and the best solution from the top solutions is retrieved.
2. In this method the fitness value threshold is explicitly mentioned, and the algorithm stops evolving once the solution generated has the fitness value greater than or equal to the threshold.

V. EXPERIMENT

The algorithm mentioned in IV Methodology section is programmed in Python 3, and the study was conducted in Google Compute Engine backend with Intel(R) Xeon(R) CPU @ 2.20GHz processor and with a total 16 GB RAM. The experiments were conducted with the 3 datasets mentioned in the Table I. Table III shows the various hyper parameter settings with which the experiments were carried out for three datasets.

TABLE III. HYPERPARAMTERS

Hyper-parameter	Arrhythmia	Breast Cancer	White Wine quality
Crossover Probability	0.9	0.9	0.9
Mutation rate	0.01	0.01	0.01
Population size	50	20	10
Mating pool	25	10	5
Type of fitness evaluator	SVM	Random forest	Random forest
Total generation	100	100	20

The population size is chosen to be 10 for white wine quality dataset because the size of the chromosome (same as the feature vector) is 7. And by application of the brute force, we can find the solution in 128 iterations as there are only 128 possible feature subsets for this dataset. There are 9.7133445×10^{83} possibilities of feature subset for the Arrhythmia dataset as there 279 features and 1.073×10^{9} possibilities for Breast cancer dataset as there are 30 features. In the worst-case scenario, the total time that would have taken for the brute force approach to find a best solution will be represented in terms of years. The accuracy obtained without feature selection is shown in the Table II in the section III. The goal of this experiment is to find a feature subset that produces the best Machine learning model.

VI. RESULTS AND CRITICAL EVALUATION

There were significant improvements to validation accuracy on application of genetic algorithms to remove the irrelevant features.

TABLE IV. ACCURACY IMPROVEMENTS FOR DIFFERENT SELECTION METHODS

Dataset	Random Selection of parents		Fitness proportionate selection of parents	
	Top Validation accuracy	Best solution generation	Top Validation accuracy	Best solution generation
Arrythmia	80.21%	18	83.51%	7
Breast Cancer	99.9%	8	99.9%	24
White Wine quality	71.6%	6	72.04%	12

The experiments were conducted with two selection models as mentioned in Table IV with the same hyper parameters and the results were compared. Figure 7 shows the plot of the variation in average, minimum, maximum fitness throughout the generations of evolution for the three different datasets with max fitness-based selection approach. Figure 8 shows the similar plot as Figure 7 but with selection based on the roulette wheel approach (fitness proportionate selection). In Figure 7 and 8, there are many valleys of the fitness values in the graph, this can be attributed to the fact that the algorithm has taken a bad step that is produced a feature subset that is not optimal. Since for mating parent candidates with higher fitness value are selected, so the resultant offspring theoretically should have the better fitness than that of their parents. But the results show that crossover between two top candidates can also result in a bad generation of candidates.

The validation accuracy increases by 7% (random selection) and 9% (fitness proportionate) for Arrhythmia dataset, 7% (same for random and fitness proportionate) for breast cancer dataset and 4% (random selection), 5% (fitness proportionate selection) for white wine quality dataset when the model is trained with feature subset obtained from the genetic algorithm. The increase in accuracy can be accounted to the features dropped being redundant and irrelevant for classification. There were 113 features dropped in the Arrhythmia dataset, 13 features for breast cancer and 3 features for the white wine quality dataset. The features with have no correlation to the classification was also removed by the genetic algorithm which decreased the time complexity of the classification model. Also, lower no. of features trains a model that is less computationally expensive

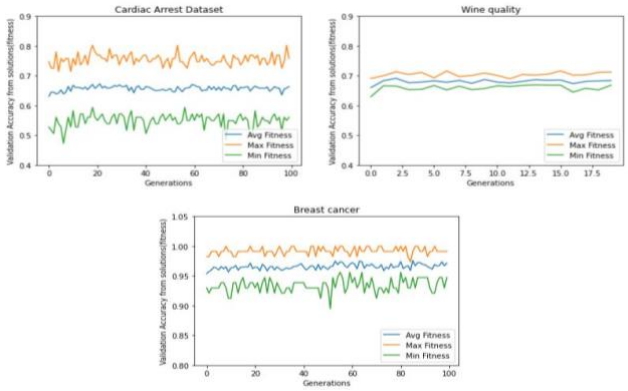

Figure 7 Random Selection (fitness in generations)

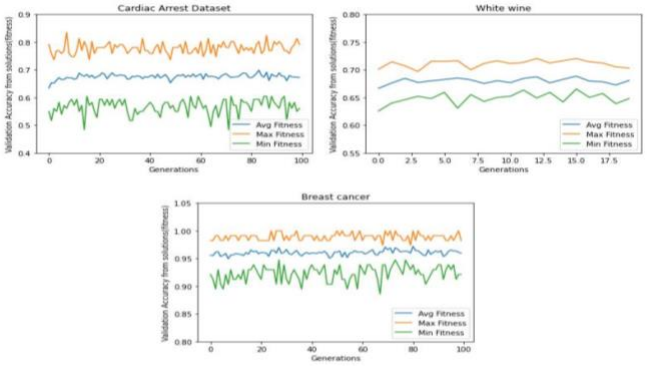

Figure 8 Fitness proportionate selection (fitness through generations)

From the Table IV, it can be deduced that the type of selection doesn't concern in how fast the genetic algorithm converge to a local minimum. For the Breast cancer and White wine quality dataset there is no impact of the selection method because of their small solution space. But for Arrhythmia dataset the solution space is so wide with many local minima. The fitness proportionate selection leads the algorithm to take bad steps as the unfit candidate solutions are also chosen for crossover with lower probability. Hence there were better results for the Arrhythmia dataset when the fitness proportionate selection was used.

On increasing the population size there is an increase in the accuracy as the solution space considered is widened for each iteration. There is no significant impact on the breast cancer and white wine quality dataset in changing the population size because of their small solution space. But for the Arrhythmia dataset, we can deduce that many individuals in the population increase the chances of finding the best optimal feature subset in less time. Figure 9 shows the plot of max fitness in each generation when the population size is 100, 200, and 300 for the Arrhythmia dataset. The best solution found with population size 100, 200, and 300 in 100 generations is 82.41%, 84.61%, and 85.71% respectively. Fitness proportionate selection was used in this experiment. The best solution was found in 22, 33, and 41 generations for the population size of 100, 200, and 300 respectively.

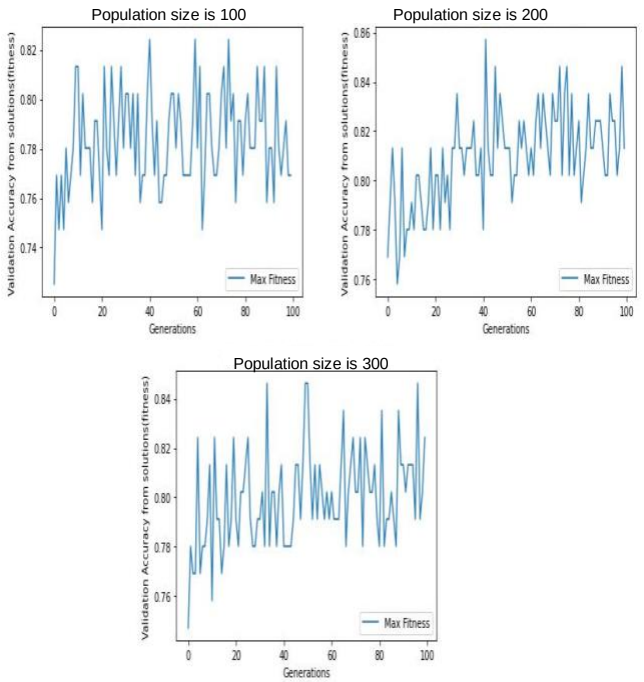

Figure 9 Varying population sizes

From Figure 9 it is evident that the higher the size of the population size higher the chance of obtaining an optimal solution in lower time. For the population size of 300, it took only 41 generations to attain a solution with a validation accuracy of 85.71% which was never attained in 100 generations with a population size of 100 and 200. From the above results we can infer that the population size for the genetic algorithm must be set proportionate to the solution space for better convergence.

Figure 10 shows the scatter plot of candidate solutions in various generation and how they got evolved into better generations for the Arrhythmia dataset.

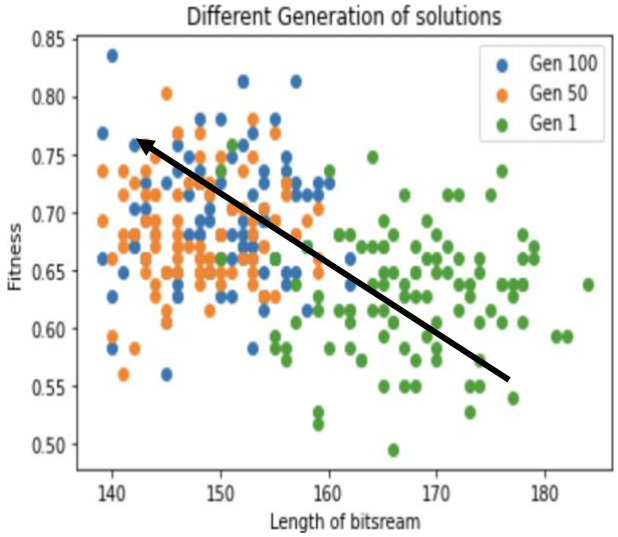

Figure 10 Scatter of solutions in different generations

The x-axis of the plot in Figure 10 is the length of the feature subset evolved. The total number of features was 279 but when the evolution started the candidate solutions in the generation evolved to move towards the top left of the graph. The top left part of the plot in Figure 10 indicates the maximum validation accuracy and minimum the feature size. As the generation progresses, we can see the movement of the solutions towards maximizing the validation accuracy and minimizing the no of features selected. The movement of solutions towards the global minima indicates that the evolution results in candidates with better traits. Each solution in the population is competing for its survival and crossover which is same as the biological evolution. In biological evolution the best traits are passed on to the next generations which result in generations that are more fit.

VII. Conclusion and Future Scope

In this report, we discussed how the nature-inspired genetic algorithm can be used to efficiently find a feature subset that can be used to train a model resulting in high validation accuracy. The algorithm tries to reduce the dimension of the feature, simultaneously increasing the classification performance. The validation accuracy obtained from the model developed using the feature subset obtained from the genetic algorithm is the measure of the performance of the proposed algorithm. Also, from the results obtained we can conclude that

genetic algorithms can perform better for bigger solution space. The performance of the genetic algorithm solely depends on the hyperparameters setting. In the above experiment, we found that population size parameter should be given a value proportionate to the size of the solution space for the algorithm to be effective. The candidate solution evolving through generations can be analogized with the biological creatures which evolve into fitter individuals.

The genetic algorithm can be applied to a variety of dimensionality reduction problems, these algorithms can be used in image compression as image compression is a problem of reducing dimensions and reducing the compression loss. The quick convergence and the higher chance of finding global minima, we can conclude that the genetic algorithms can be extended to any optimization problem that can be represented in chromosomes and has an objective function to calculate the fitness.

REFERENCES

[1] "Genetic algorithms for feature selection," Genetic algorithms for feature selection | Neural Designer. [Online]. Available: https://neuraldesigner.com/blog/genetic_algorithms_for_feature_selection. [Accessed: 26-Oct-2022].

[2] K. De Jong, D. B. Fogel, and H.-P. Schwefel, ''A history of evolutionary Computation,'' in Handbook of Evolutionary Computation, T. Baeck, D. B. Fogel, and Z. Michalewicz, Eds. Boca Raton, FL, USA: CRC Press, 1997.

[3] J. H. Holland, Adaptation in Natural and Artificial Systems. Ann Arbor, MI, USA: Michigan Univ. Press, 1975.

[4] Z. Michalewicz and M. Schoenauer, ''Evolutionary algorithms for constrained parameter optimization problems,'' Evol. Comput., vol. 4, no. 1, pp. 1–32, 1996.

[5] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on Genetic Algorithm: Past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, 2020.

[6] M. Mitchell and S. Forrest, "Genetic algorithms and artificial life," *Artificial Life*, vol. 1, no. 3, pp. 267–289, 1994.

[7] Koza, J.R. Genetic programming as a means for programming computers by natural selection. *Stat Comput* 4, 87–112 (1994). https://doi.org/10.1007/BF00175355

[8] W. SIEDLECKI and J. SKLANSKY, "A note on genetic algorithms for large-scale feature selection," *Handbook of Pattern Recognition and Computer Vision*, pp. 88–107, 1993.

[9] S. Pasala, B. N. Kumar, and S. C. Satapathy, "A study of Roulette Wheel and elite selection on ga to solve job shop scheduling," *Advances in Intelligent Systems and Computing*, pp. 477–485, 2013.

[10] R. Ge, M. Zhou, Y. Luo, Q. Meng, G. Mai, D. Ma, G. Wang, and F. Zhou, "McTwo: A two-step feature selection algorithm based on maximal information coefficient," *BMC Bioinformatics*, vol. 17, no. 1, 2016.

[11] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn, and A. K. Jain, "Dimensionality reduction using genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 2, pp. 164–171, 2000.

[12] R. Abu Khurma, I. Aljarah, A. Sharieh, M. Abd Elaziz, R. Damaševičius, and T. Krilavičius, "A review of the modification strategies of the nature inspired algorithms for feature selection problem," *Mathematics*, vol. 10, no. 3, p. 464, 2022.

[13] Brank, J.; Mladeni´c, D.; Grobelnik, M.; Liu, H.; Mladeni´c, D.; Flach, P.A.; Garriga, G.C.; Toivonen, H.; Toivonen, H. Feature Selection. In Encyclopedia of Machine Learning; Springer: Boston, MA, USA, 2011; pp. 402–406. doi:10.1007/978-0-387-30164-8_306

[14] Punch, W. F., Goodman, E. D., Pei, M., Chia-Shun, L., Hovland, P., and Enbody, R. (1993). "Further research on feature selection and classification using genetic algorithms", *Proc. of 5th International Conference on Genetic Algorithms*, 557–564.

[15] G. T. Hilda and R. R. Rajalaxmi, "Effective feature selection for supervised learning using genetic algorithm," *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, 2015.

[16] B. J. B. Keats and S. L. Sherman, "Population genetics," *Emery and Rimoin's Principles and Practice of Medical Genetics*, pp. 1–12, 2013.

[17] C.L. Blake and C.J. Merz. UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Sciences,1998.

[18] P.M. Murphy and D.W. Aha, UCI Repository of Machine Learning Databases: Machine Readable Data Repository. Univ. of California at Irvine, 1994.

[19] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547–553, 2009.

Appendix I

Genetic Algorithm code

```
"""Genetic_Algo_new_version.pynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1_uSUh0mgQnoZXi4XPKHcz
IkmFt9w06uE

Author: Sriram Ranganathan, Meng. ECE, University of Waterloo
(Main Code)
"""

import numpy as np
import random
from sklearn import svm
from sklearn.model_selection import train_test_split
import pandas as pd
from tqdm import tqdm
import sys
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import numpy.random as npr

def estimate(T_x, T_y, V_x, V_y, clf_type):
  if clf_type == "SVM":
    clf = svm.SVC(C = 1, kernel = 'poly', gamma = 'auto')
    clf.fit(T_x, T_y)
    return clf.score(V_x, V_y)

  if clf_type == "RFC":
    clf_ww = RandomForestClassifier(max_depth=50,
n_estimators= 200, random_state=0)
    clf_ww.fit(T_x, T_y)
    return clf_ww.score(V_x, V_y)

def roulette_select_one(c_population):
  max = sum([f.fitness for f in c_population])
  probs = [f.fitness/max for f in c_population]
  return c_population[npr.choice(len(c_population), p=probs)]

#Code author: S
class candidate:
```

```
  def __init__(self, bitstream,  fitness = 0.00):
    self.fitness = fitness
    self.bitstream = bitstream
  def __eq__(self, x):
    if self.bitstream == x.bitstream:
      return True
    return False

class top_solution:
  def __init__(self, new,  iter = 0):
    self.iter = iter
    self.new = new
class Genetic_algorithm:

  def __init__(self, input_data_x, input_data_y,
max_population, crossover_prob, mutation_r, stop_by_f,
stop_fitness, fitness_func):
    self.input_data_x = input_data_x
    self.input_data_y = input_data_y.to_numpy()
    #self.input_data_v_x = input_data_v_x
    #self.input_data_v_y = input_data_v_y.to_numpy()
    self.max_population = max_population
    self.columns = self.input_data_x.columns
    self.fitness_func = fitness_func
    self.populate()
    self.calculate_fitness()
    self.mating_pool_size = max_population//5
    self.crossover_prob = crossover_prob
    self.mutation_r = mutation_r
    self.Best_solutions = []
    self.Best_solutions_bit = []
    self.Best_iteration = 0
    self.stop_fitness = stop_fitness
    self.stop_by_f = stop_by_f
    self.fitness_dispersion = []
    self.len_bitstream_dispersion = []
  def evolve(self, no_iters):
    print("Genetic Algorithm Evolving")
    i = 0
    self.average = []
    self.Top_sols = []
    self.worst_sols = []
    self.tot_crossov = []
    self.tot_mut = []
    l = range(0,no_iters)
    self.crossover = 0
    self.mutation = 0
    for i in l:
```

```
      top_sol = self.current_population[0]
      self.Best_solutions.append(top_sol.fitness)
      self.Best_solutions_bit.append(top_sol.bitstream)
      print("Top solution fitness "+ str(top_sol.fitness))
      print("Iteration_No: ", i)

      fitness = [m.fitness for m in self.current_population]
      self.average.append(sum(fitness)/len(fitness))
      self.Top_sols.append(max(fitness))
      self.worst_sols.append(min(fitness))
      if ((top_sol.fitness > self.stop_fitness) &
(self.stop_by_f)):
        return top_sol
      self.current_population =
self.cross_over_mutate(self.current_population)
      self.calculate_fitness()
      i+=1
      self.current_population.sort(key=lambda x: x.fitness,
reverse=True)

    #self.Best_solutions.sort(key=lambda x: x.fitness,
reverse=True)
    #print("Best solution fitness: ",
self.Best_solutions[0].fitness)
    #print("Best Generation: ", self.Best_solutions)
      self.tot_crossov.append(self.crossover)
      self.tot_mut.append(self.mutation)
    return top_sol


  def populate(self, initial = False):
    print("Creating Initial population")
    self.current_population = []
    for i in range(0,self.max_population):
      bitstream = []
      for i in self.input_data_x.columns:
        if random.randrange(10)<=5:
          bitstream.append(1)
        else:
          bitstream.append(0)

      new_cand = candidate(bitstream)
      rep = False
      for i in self.current_population:
        if bitstream == i.bitstream:
          rep = True
          break
      if rep == True:
        continue
```

```
      self.current_population.append(new_cand)
    return

  def calculate_fitness(self):
    print("Calculating fitness")

    for i in self.current_population:
      new_data_frame = self.input_data_x
      bitstream = i.bitstream
      drop_columns = []
      for k in range(0,len(bitstream)):
        if bitstream[k] == 1:
          continue
        if bitstream[k] == 0:
          drop_columns.append(self.columns[k])

      new_data_frame = self.input_data_x.drop(drop_columns,
axis = 1)
      Train_x = new_data_frame.to_numpy()
      '''new_data_frame =
self.input_data_v_x.drop(drop_columns, axis = 1)
      Test_x = new_data_frame.to_numpy()'''
      X_train, X_test, y_train, y_test =
train_test_split(Train_x, self.input_data_y, test_size=0.2)
      i.fitness = estimate(X_train, y_train, X_test, y_test,
self.fitness_func)
    return


  def cross_over_mutate(self, current_population):
    self.fitness_dispersion.append([f.fitness for f in
self.current_population])
    y = [np.array(f.bitstream) for f in
self.current_population]
    y = [np.sum(l) for l in y]
    self.len_bitstream_dispersion.append(y)
    current_population.sort(key=lambda x: x.fitness,
reverse=True)
    new_population = current_population[0:2]
    print("Top 2 Fitness of new population",
new_population[0].fitness, new_population[1].fitness)
    mating_pool =
current_population[:self.mating_pool_size].copy()
    m = 0
    while(len(new_population)<len(current_population)):
      n = m
      if m>=len(mating_pool):
        n = m%len(mating_pool)
      p1 = roulette_select_one(mating_pool)
```

```
      new_mating_pool = mating_pool.copy()
      new_mating_pool.pop(n)
      new_cand = candidate([], 0)
      if random.uniform(0, 1)<=self.crossover_prob:
        self.crossover+=1
        p2 = roulette_select_one(mating_pool)
        trait_split =
random.randrange(self.input_data_x.shape[1])
        L = [k for k in
range(trait_split,self.input_data_x.shape[1])]
        trait_split1 = random.choice(L)
        new_bitstream =
self.mutate(p1.bitstream[0:trait_split] +
p1.bitstream[trait_split:trait_split1]+p2.bitstream[trait_spli
t1:])
        new_cand.bitstream = new_bitstream
        bs = [str(k) for k in new_cand.bitstream]
        rep = False
        '''
        for u in new_population:
          if new_cand.bitstream == u.bitstream:
            rep = True
        if rep:
          continue
        '''
        new_population.append(new_cand)
      m+=1
    current_population = new_population.copy()

    current_population.sort(key=lambda x: x.fitness,
reverse=True)
    return current_population


  def mutate(self, bitstream):
    for i in range(0,len(bitstream)):
      if random.uniform(0, 1)<=self.mutation_r:
        self.mutation
        if bitstream[i]==0:
          bitstream[i] = 1
        else:
          bitstream[i] = 0
    return bitstream

new_df =
pd.read_csv('/content/drive/MyDrive/ECE_750_data/C_A_D_cleaned
.csv')

new_df.head()
```

```
Train = new_df
#Vaidate = new_df.drop(Train.index)

Train_data_x = Train.drop(['Out'], axis = 1)
#Test_data_x = Vaidate.drop(['Out'], axis = 1)

Train_data_y = Train['Out']
#Test_data_y = Vaidate['Out']

GA_CAD = Genetic_algorithm(input_data_x =
Train_data_x,input_data_y = Train_data_y,
                           max_population = 100,
crossover_prob = 0.9,
                           mutation_r = 0.01, stop_by_f =
False, stop_fitness = 0.81, fitness_func="SVM")

GA_CAD.evolve(100)

ind = np.argmax(GA_CAD.Best_solutions)
ind

max(GA_CAD.Best_solutions)

best_bit = GA_CAD.Best_solutions_bit[ind]

x = Train_data_x.columns
drop = []
for i in range(0, len(best_bit)):
  if best_bit[i] == 1:

    continue
  else:
    drop.append(x[i])
print("No. of columns to be dropped: ",len(drop))
print(drop)

#plt.plot(range(0,len(GA_CAD.Top_sols)),GA_CAD.average, label
= "Avg Fitness")
plt.plot(range(0,len(GA_CAD.Top_sols)),GA_CAD.Top_sols, label
= "Max Fitness")
#plt.plot(range(0,len(GA_CAD.Top_sols)),GA_CAD.worst_sols,
label = "Min Fitness")
#plt.ylim(0.4, 0.9)
plt.xlabel('Generations')
plt.ylabel('Validation Accuracy from solutions(fitness)')
plt.legend(loc="lower right")

# displaying the title
```

```
plt.title("Cardiac Arrest Dataset Size = 300")

for i in range(0, len(GA_CAD.len_bitstream_dispersion)):

plt.scatter(GA_CAD.len_bitstream_dispersion[i],GA_CAD.fitness_
dispersion[i])
  plt.title("Generation "+ str(i))
  plt.xlabel("Length of bitsream")
  plt.ylabel("Fitness")
  plt.ylim(0.5, 0.85)
  plt.xlim(120, 180)

plt.savefig("/content/drive/MyDrive/ECE_750_data/GEN/Gen_"+str
(i)+".jpg")
  plt.close()

plt.scatter(GA_CAD.len_bitstream_dispersion[99],GA_CAD.fitness
_dispersion[99], label = 'Gen 100')
plt.scatter(GA_CAD.len_bitstream_dispersion[49],GA_CAD.fitness
_dispersion[49], label = 'Gen 50')
plt.scatter(GA_CAD.len_bitstream_dispersion[0],GA_CAD.fitness_
dispersion[0], label = 'Gen 1')
plt.title("Different Generation of solutions")
plt.xlabel("Length of bitsream")
plt.ylabel("Fitness")
plt.legend(loc="best")

GA_CAD.fitness_dispersion[0]

"""## White Wine Quality Data set"""

df_white_wine =
pd.read_csv('/content/drive/MyDrive/ECE_750_data/winequality-
white.csv')

df_white_wine.head()

Train_x_ww = df_white_wine.drop(['quality'], axis =1)
Train_y_ww = df_white_wine.quality

GA_WW = Genetic_algorithm(input_data_x =
Train_x_ww,input_data_y = Train_y_ww,
                          max_population = 20, crossover_prob
= 0.9,
                          mutation_r = 0.01, stop_by_f =
True, stop_fitness = 0.81, fitness_func="RFC")

GA_WW.evolve(20)
```

```
ind_ww = np.argmax(GA_WW.Best_solutions)
ind_ww

max(GA_WW.Best_solutions)

best_bit_ww = GA_WW.Best_solutions_bit[ind_ww]
best_bit_ww

x_ww = df_white_wine.drop(['quality'], axis =1).columns
drop_ww = []
for i in range(0, len(best_bit_ww)):
  if best_bit_ww[i] == 1:
    continue
  else:
    drop_ww.append(x_ww[i])
print("No. of columns to be dropped: ",len(drop_ww))
print(drop_ww)

plt.plot(range(0,len(GA_WW.Top_sols)),GA_WW.average, label =
"Avg Fitness")
plt.plot(range(0,len(GA_WW.Top_sols)),GA_WW.Top_sols, label =
"Max Fitness")
plt.plot(range(0,len(GA_WW.Top_sols)),GA_WW.worst_sols, label
= "Min Fitness")
plt.ylim(0.55, 0.8)
plt.xlabel('Generations')
plt.ylabel('Validation Accuracy from solutions(fitness)')
plt.legend(loc="lower right")

# displaying the title
plt.title("White wine")

"""### GA for Breast cancer"""

df_BC =
pd.read_csv('/content/drive/MyDrive/ECE_750_data/B_C_cleaned.c
sv')

df_BC.head()

Train_x_bc = df_BC.drop(['Out'], axis =1)
Train_y_bc = df_BC.Out

GA_BC = Genetic_algorithm(input_data_x =
Train_x_bc,input_data_y = Train_y_bc,
                          max_population = 20, crossover_prob
= 0.9,
                          mutation_r = 0.01, stop_by_f =
False, stop_fitness = 0.98, fitness_func="RFC")
```

```
GA_BC.evolve(100)

ind_BC = np.argmax(GA_BC.Best_solutions)
ind_BC

max(GA_BC.Best_solutions)

best_bit_BC = GA_BC.Best_solutions_bit[ind]

x_BC = Train_x_bc.columns
drop_BC = []
for i in range(0, len(best_bit_BC)):
  if best_bit_BC[i] == 1:
    continue
  else:
    drop_BC.append(x_BC[i])
print("No. of columns to be dropped: ",len(drop_BC))
print(drop_BC)

plt.plot(range(0,len(GA_BC.Top_sols)),GA_BC.average, label =
"Avg Fitness")
plt.plot(range(0,len(GA_BC.Top_sols)),GA_BC.Top_sols, label =
"Max Fitness")
plt.plot(range(0,len(GA_BC.Top_sols)),GA_BC.worst_sols, label
= "Min Fitness")
plt.ylim(0.8, 1.05)
plt.xlabel('Generations')
plt.ylabel('Validation Accuracy from solutions(fitness)')
plt.legend(loc="lower right")

# displaying the title
plt.title("Breast cancer")

plt.plot(range(0,len(GA_BC.Top_sols)),GA_BC.average, label =
"line 1")
plt.xlabel('Generations')
plt.ylabel('Avg Validation Accuracy from solutions(fitness)')

# displaying the title
plt.title("Avg fitness vs Generations")

plt.plot(range(0,len(GA_BC.Top_sols)),GA_BC.Top_sols, label =
"line 1")
plt.xlabel('Generations')
plt.ylabel('Max Validation Accuracy from solutions(fitness)')

# displaying the title
plt.title("Max fitness vs Generations")
```

```
plt.plot(range(0,len(GA_BC.Top_sols)),GA_BC.worst_sols, label
= "line 1")
plt.xlabel('Generations')
plt.ylabel('Min Validation Accuracy from solutions(fitness)')

# displaying the title
plt.title("Min fitness vs Generations")
```

Appendix II
Data cleaning code

```
# -*- coding: utf-8 -*-
"""ECE_750_project.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1hCsLUO-
jyptCVdzGjWn6hrX4xNLHg7Sc
Author: Sriram Ranganathan, Meng ECE Uwaterloo
### APPENDIX I (Data Cleaning code)
"""

import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import time

"""### Cardiac Arrest dataset cleaning and classification"""

df =
pd.read_csv('/content/drive/MyDrive/ECE_750_data/C_A_D.csv')

df.head()

len(df)

out_types = df["Out"].unique()

set(df.Out)

mean_values = {}
outliers = []
for j in df.columns:
  mean_values[j] = {}
  mean = 0
  length = 1
```

```
  new_list = df[j]
  for k in range(0, len(new_list)):
    try:
      mean += int(new_list[k])
      length+=1


    except:
      outliers.append(new_list[k])
      continue

  mean = mean/length
  mean_values[j] = mean
print(set(outliers))

for j in df.columns:
  for i in set(outliers):
    df[j] = df[j].replace([i], mean_values[j])

for i in df.columns:
  for j in df[i]:
    for k in set(outliers):
      if j == k:
        print(i, j, k)

dtypes = {}
dtypes.update({k: np.float64 for k in df.columns})

df = df.astype(dtypes)

df.head()

df.to_csv('/content/drive/MyDrive/ECE_750_data/C_A_D_cleaned.csv', index=False)

new_df =
pd.read_csv('/content/drive/MyDrive/ECE_750_data/C_A_D_cleaned.csv')

new_df.head()

Train_x = new_df.drop(columns=['Out'])

Train_x.head()

Train_y = df["Out"]

Train_y.head()
```

```
Train_x = Train_x.to_numpy()

Train_y = Train_y.to_numpy()

Train_x.shape

X_train, X_test, y_train, y_test = train_test_split(Train_x,
Train_y, test_size=0.2)

clf = svm.SVC(C = 1, kernel = 'poly', gamma = 'auto')
start = time.time()
clf.fit(X_train, y_train)
print("Total time: ", time.time() - start, "seconds")
clf.score(X_test, y_test)

(0.03465723991394043* (2**279))/3.154e+7

"""### Breast Cancer Diagnostic Dataset cleaning and
classifiction (SVM)"""

df_BC =
pd.read_csv('/content/drive/MyDrive/ECE_750_data/B_C_cleaned.c
sv')

df_BC.head()

Train_x_BC = df_BC.drop(['Out'], axis =1)
Train_y_BC = df_BC.Out

dtypes = {}
dtypes.update({k: np.float64 for k in df_BC.drop(['Out'], axis
=1)})

Train_x_BC = Train_x_BC.astype(dtypes)

Train_x_BC = Train_x_BC.to_numpy()

X_train_BC, X_test_BC, y_train_BC, y_test_BC =
train_test_split(Train_x_BC, Train_y_BC, test_size=0.2)

clf_BC = RandomForestClassifier(max_depth=2, n_estimators=
200, random_state=0)
clf_BC.fit(X_train_BC, y_train_BC)
clf_BC.score(X_test_BC, y_test_BC)

"""### White wine quality Dataset cleaning and training SVM"""
```

```
df_white_wine =
pd.read_csv('/content/drive/MyDrive/ECE_750_data/winequality-
white.csv')

df_white_wine.head()

len(set(df_white_wine.quality))

Train_x_ww = df_white_wine.drop(['quality'], axis
=1).to_numpy()
Train_y_ww = df_white_wine.quality.to_numpy()

X_train_ww, X_test_ww, y_train_ww, y_test_ww =
train_test_split(Train_x_ww, Train_y_ww, test_size=0.2)

clf_ww = RandomForestClassifier(max_depth=50, n_estimators=
200, random_state=0)
#clf_ww = svm.SVC(C = 1, kernel = 'poly', gamma = 'auto')
clf_ww.fit(X_train_ww, y_train_ww)
clf_ww.score(X_test_ww, y_test_ww)
```