A Linear Classifier for Determining the Presence of Malignancy in Breast Tumors

Final Report

Clemens Vallejo

Zayd Hammoudeh

Table of Contents

List of Figures

List of Tables

Introduction

In predictive analytics, a classification algorithm is a supervised learning technique that uses a training data set to generate models that can classify future data elements into one of a finite set of categories. Each element in the training data set is a pairing of an input data vector and a classification value. This fundamental flow of a classification algorithm is shown in Figure 1.

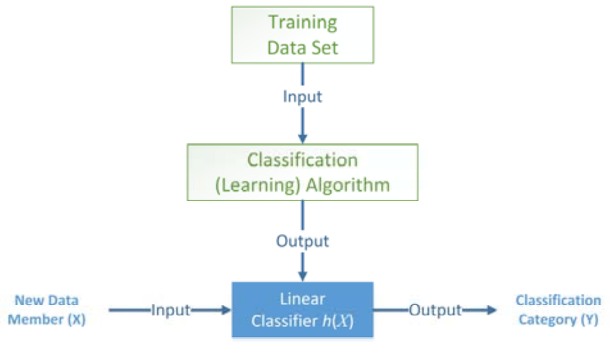


Figure 1 – Flow Diagram of a Classification Algorithm

This report describes an algorithm that generates a model to classify breast cancer tumors as either malignant or benign. For a description of the source data set, see the section entitled "Data Set Overview".

Linear Classification

One of the inputs to a classification algorithm is a set data vectors; each data vector, $\vec{x}$, is *N*-dimensional. In a linear classification algorithm, there is a single *N*-dimensional weight vector, $\vec{w}$; an element, w_i, in $\vec{w}$ represents a scalar weight for the *i*th dimension. A linear classifier, $h(\vec{x})$, is defined as:

$$h(\vec{x}) = \vec{w}^T \cdot \vec{x} + w_0 \tag{1}$$

$\vec{w}^T$ is the transpose of the weight vector, $\vec{w}$, and the centered dot symbol " $\cdot$ " is the dot/scalar product of the two vectors. w_0 is an optional scalar offset term. Since a dot product is used and given that the offset term, w_0, is a scalar, the linear classifier, $h(\vec{x})$, is also a scalar. Depending on this scalar value of $h(\vec{x})$, each data member is classified into a category, Y, using a set classification threshold(s) as shown in Figure 1.

Genetic Algorithm Overview

There are multiple techniques that can be used to determine appropriate values for the linear classifier's terms, $\vec{w}$ and w_0, including support vector machine, neural networks, etc. In this paper, we use a genetic algorithm (GA) to generate the linear classifier constraints; we selected this approach due to the methodology's overall robustness and low execution overhead, both in terms of memory and execution time.

A genetic algorithm is a local search, learning algorithm that is modeled after the biological process of natural selection. Genetic algorithms begin with a set of randomly generated solutions to a problem. Each solution is referred to as an "individual" or "chromosome" while the set of all solutions is referred to as the "population". Each solution is given a quality rating by a fitness function, f. Over a series of iterations (called "generations"), pairs of chromosomes from the previous generation (i.e. parents) are merged to form the new chromosomes (i.e. children) that comprise the successor generation. The process where the two paternal chromosomes are merged to form a descendent chromosome is known as "crossover". After crossover, the successor chromosome undergoes "mutation" where part(s) of the solution may be randomly changed. After a specified number of generations, a new chromosome seed population can be optionally created; this is known as a "random restart". At the end of all generations and random restarts, the best solution (i.e. the linear classifier's weight vector, w, and offset scalar, w_0) is returned by the algorithm.

Figure 2 is pseudocode detailing the implementation of our genetic algorithm.

```
for restart_number = 1 to NUMBER_RANDOM_STARTS do

    population ← generate N random chromosomes
    for generation_number = 1 to MAX_NUMBER_GENERATIONS do
        evaluate all chromosomes using the fitness functions
        copy M best chromosomes to new_population
        while size(new_population)  ≤ MAX_POPULATION_SIZE do
            parent1 ← select chromosome
            parent2 ← select chromosome
            child ← crossover(parent1 , parent2 )
            potentially randomly mutate child
            add child to new_population
        end while
        population ← new_population
    end for
    if(fitness(best_solution) < fitness(population.best_solution))
        best_solution = population.best_solution

end for
return best_solution
```

Figure 2 – Pseudocode for the Breast Cancer Genetic Algorithm

The following subsection describes the specific implementation details of our genetic algorithm.

Chromosome (Solution) Structure

The data set has 9 features, and there is an additional term in the linear classifier function for the offset (w_0). In our algorithm, each weight is a 32-bit two's complement integer. Hence, given the 9 features and the offset term, a chromosome is 320 (i.e. $32 * 10$) bits long.

Classification Strategy and Threshold

A linear classifier returns a scalar value. In our implementation, any patient tumor with a negative classifier value is categorized as benign while any data member whose classifier function value is positive is categorized as malignant. This approach was selected because it is the simplest to implement and is the easiest for a user to understand. What is more, this approach leads to no reduction in flexibility. These combined factors make this approach ideal.

Chromosome Population Size

A large population increases solution diversity. However, as the population size increases, the incremental population diversity decreases. The default population size for our algorithm is 1000 since it provides a reasonable trade-off between solution quality and algorithm execution time. However, this parameter is configurable by the user as detailed in the section entitled "Running the Breast Cancer Classifier Genetic Algorithm."

Reproduction Selection Algorithm

In a genetic algorithm, tournament selection involves randomly choosing *n* chromosomes from the population; from these *n* possible solutions, the chromosome with the highest fitness is selected to be a parent of a successor chromosome (i.e. solution). Tournament selection has low computational overhead and prevents the algorithm converging too quickly (assuming *n* is not too large relative to the population size). For these reasons, we used this approach as the natural selection paradigm in our algorithm.

Fitness Function

A fitness function measures the quality of any solution (i.e. chromosome). Eq. (2) is the genetic algorithm's primary fitness function, f_1; it quantifies the number of data set members that were correctly classified.

$$f_1 = \sum_{i=1}^{M} g_i \tag{2}$$

g_i is defined in eq. (3) and returns an integer reward for the correct classification of tumor i.

$$g_i = \begin{cases} 1 & \text{if } \theta_i \cdot h(\overrightarrow{x_i}) > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{3}$$

In eq. (3), *M* is the size of the population; *i* represents the *i*th tumor in the data set, and $h(\overrightarrow{x_i})$ is the value of the linear classifier function for tumor vector, $\overrightarrow{x_i}$. θ_i is a normalizing scalar defined by the relation in eq. (4); this normalizing scalar is used to reward correct classification of benign tumors, which will have negative scores.

$$\theta_i = \begin{cases} 1 & \text{if tumor } i \text{ is malignant} \\ -1 & \text{if tumor } i \text{ is benign} \end{cases} \tag{4}$$

Although not shown in eq. (3), our genetic algorithm supports the use of a malignancy bias factor, which is an additional weighting term to prioritize the correct classification of malignant tumors.

The main priority of the fitness function is to correctly classify the maximum number of patient tumors. However, in many cases, multiple chromosomes will have equivalent values for eq. (2); this is most likely to occur when the chromosome population size and/or generation count are large. As such, an additional fitness function is used to enable the algorithm to further determine solution quality. This second fitness function, f_2, is shown in eq. (5), and quantifies the total error margin of each solution. Note that this secondary function is only used when there is a tie for fitness function f_1.

$$f_2 = \sum_{i=1}^{M} \theta_i \cdot h(\vec{x}) \qquad (5)$$

Crossover

Crossover is the process of combining two parent chromosomes to form a child chromosome. With the exception of one-point crossover, we did not observe a strong correlation between the number of crossover points and the classification accuracy. As such, we selected three-point crossover since it provides an adequate balance between classification accuracy and execution complexity.

Mutation Frequency

A higher mutation frequency is correlated with increased solution diversity. However, if the mutation frequency is too high, it can have a deleterious effect as it can corrupt otherwise good solutions. The bit mutation frequency is set to 1% in our solution, which, while high, showed no negative effects on the algorithm's accuracy.

Generation Count

Each round in a genetic algorithm is referred to as a "generation". By default, our genetic algorithm specifies that there will be 1000 generations for each initial chromosome population. This number was selected since we saw solution convergence below 1000 generations; by limiting the generation count, the algorithm's performance is improved by eliminating the need to continue examining solutions which have a very limited likelihood of further improvement.

Random Restarts

In a random restart, all members of the current chromosome population are discarded; the only exception is the population's best solution, which is stored for future evaluation. After discarding the old population, a new, complete, random solution set is generated, which serves as the parents for the subsequent generation. By allowing random restarts, a genetic algorithm's solution diversity is increased which in turn increases the likelihood of bypassing local minima. In our algorithm, we set the number of random starts to five (i.e. four random restarts). This was used as we saw solution convergence generally between the second and fourth random start.

Breast Tumor Classifier Program Overview

Figure 3 is a block diagram of the components of the program that develops the linear classifier, $h(\vec{x})$. The breast cancer data set is imported into our program and split into two disjoint sets (i.e. training and verification). On each run of the program, the members of the two disjoint sets are respecified; the program's default setting is to randomly divide the two data sets. However, as explained in the section entitled "Running the Breast Cancer Classifier Genetic Algorithm", the user does have some flexibility in specifying how the two data sets are created.

Once the two data sets have been formed, the program runs the genetic algorithm to create the linear classifier's parameters. The efficacy of this classifier is measured by quantifying how accurately the resulting model classifies patient tumors in the verification data set.

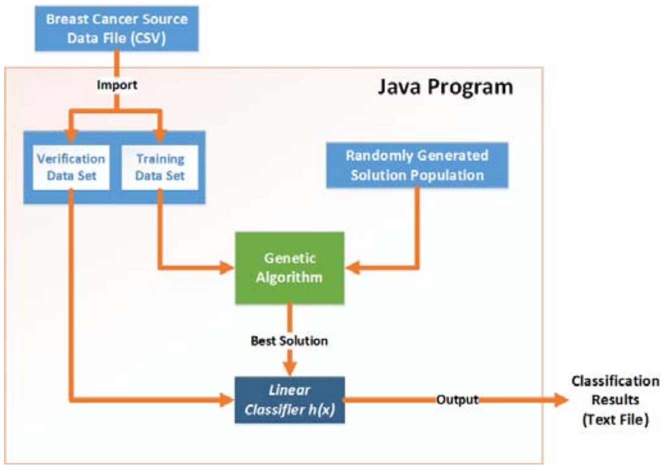


Figure 3 – Block Diagram of the Components in the Breast Cancer Classifier Program

Our program is implemented in the Java programming language. Specific benefits of Java include its portability across platforms through the Java Virtual Machine, a large user base, and extensive built-in libraries. What is more, Java is an object-oriented programming language. Classification problems (especially genetic algorithms) are generally conducive to being programmed using an object based methodology. As such, our implementation uses the object-oriented paradigm extensively.

Appendix A is a UML class diagram of our program. The names of the five classes and brief descriptions of each class' role are below:

1. **BreastCancerGeneticAlgorithm** – Main program class. It is through this class that an application programmer would interact with and run the program. This class aggregates objects of the other four classes.
2. **Patient** – This eponymous class represents one patient tumor in the source data set. Each `Patient` object is categorized as either malignant or benign and has values for the nine features described in the section entitled "Data Set Overview".
3. **BreastCancerDataSet** – Aggregator of objects of the `Patient` class. The two program objects of this type are: `trainingDataSet` and `verificationDataSet`.
4. **GAChromosome** – The Genetic Algorithm (GA) chromosome encapsulates the linear classifier's gain vector and offset scalar. Each `GAChromosome` object represents a possible solution; these classifiers are referred to as "chromosomes" since they undergo mutation and "sexual" reproduction when two `GAChromosome` objects are merged to form a successor `GAChromosome` object.
5. **GAChromosomePopulation** – Aggregator of objects of type `GAChromosome`.

Genetic Algorithm Results

The output of the genetic algorithm is a single chromosome; this linear classifier is used to categorize all patient tumors in the verification data set as either malignant or benign. There are two primary metrics we used to quantify the quality of this classification. They are:

1. **Total Accuracy** – This is the most straightforward quality measure and is simply the ratio of the number of correct classifications to the total size of the data set.
2. **Malignancy Classification Accuracy (MCA)** – The most serious type of classification error is a false negative, where a tumor is classified as benign when it is really malignant. This metric quantities the accuracy of the classifier with respect to this type of error.

Determining the Optimal Chromosome Population Size

To determine the appropriate size of the chromosome population for our genetic algorithm, we created a fixed training set of 200 chromosomes. The number of malignant tumors in the training data set was set to be proportional to the number of malignant tumors in the complete data set. Figure 4 shows the relationship between the chromosome population size and the classification accuracy (i.e. total accuracy and MCA). For each population size, the graph shows the average classification accuracies as well as the standard deviation of the accuracies when each population size was looped 100 times.

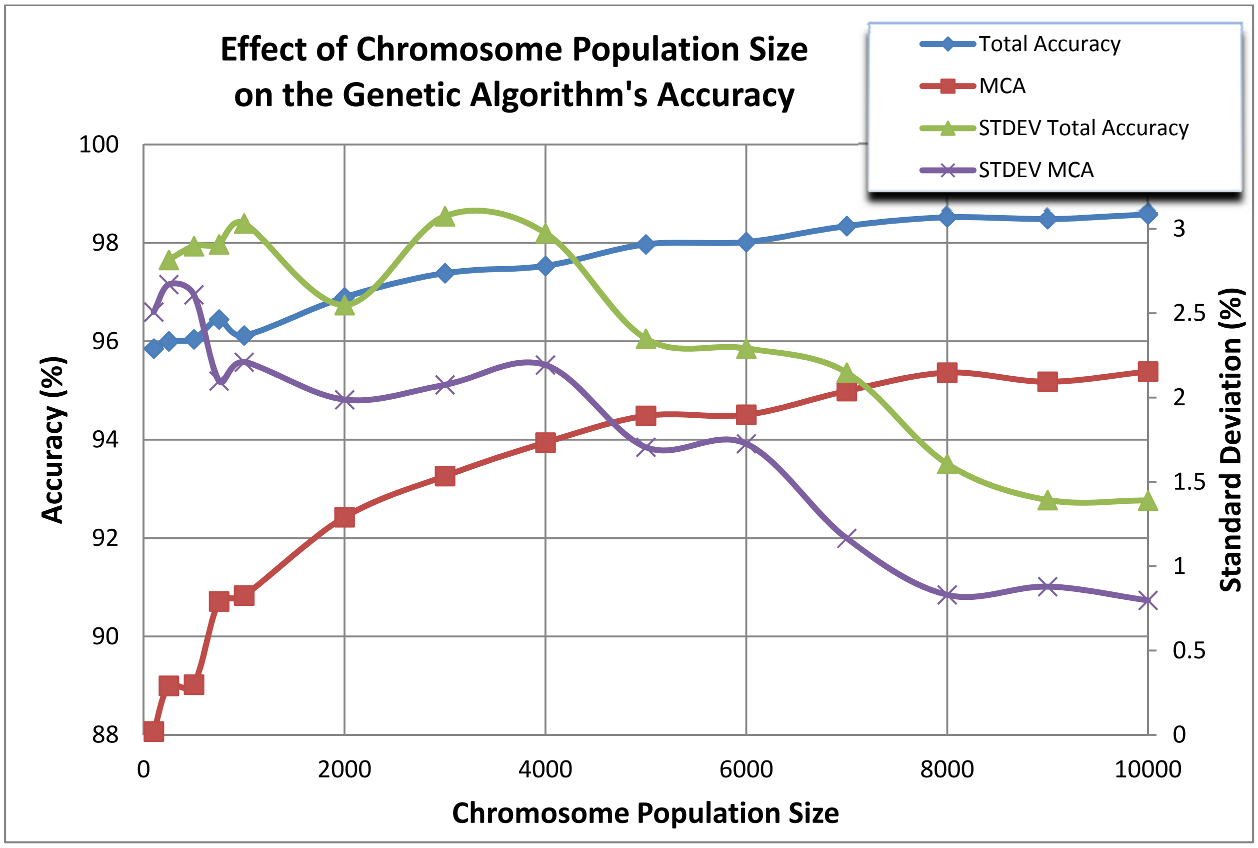


Figure 4 – Relationship between Chromosome Population Size and Classifier Accuracy

When the solution size was larger than 8,000, there was no statistically significant improvement in total accuracy or MCA. While there was some decrease in the variation for the total accuracy, the change was very minor. Based off this data, we used a chromosome population size of 8,000 for all subsequent experiments.

Composition of the Training and Verification Data Sets

The quality of the classifier is highly dependent on whether the training data set is representative of the entire data set. For example, if the number of malignant tumors in the training data set is too low, the resulting classifier may overly favor benign tumors; similarly, if the number of malignant tumors in the training data set is too high, the resulting classifier may overly favor malignant tumors.

To quantify the extent to which the number of malignant tumors in the source data set affected the variation in the classification accuracy, we ran the genetic algorithm where the training and verification data sets were entirely randomly selected. We also ran the algorithm where we required that the number of malignant tumors in both data sets be proportional to the data sets' respective sizes; this second approach would ensure the training data set was more representative.

	Total Accuracy	$\sigma_{Total\ Accuracy}$	MCA	σ_{MCA}
Proportional Data Sets	92.1%	2.14%	92.0%	3.88%
Fully Random Data Sets	91.6%	2.31%	91.0%	5.19%

Table 1 – Classifier Accuracy with and without Proportionality of Malignant Tumors in Training Data Set

Table 1 shows the average classifier accuracy and the variation of the accuracy (i.e. standard deviation) depending on whether proportionality of malignant tumors in the data sets was enforced. In both experiments, the genetic algorithm was repeated 100 times with shuffled data sets, and the chromosome population size was set to 8,000. As expected, the algorithm was more accurate when proportionality was enforced, but the effect was small (less than 1%). Moreover, data set proportionality reduced the variation in the algorithm's accuracy in particular for MCA.

The random data set results show the expected accuracy of our algorithm under the most stringent conditions for given training data set and chromosome population sizes. The algorithm has nearly a 92% average total accuracy as well as a 91% malignancy classification accuracy (MCA).

Malignancy Bias Factor

An incorrect classification of a malignant tumor is usually more deleterious than the misdiagnosis of a benign tumor as such an error may cause a patient with the malignant tumor to defer treatment, perhaps until it is too late. We examined whether a bias factor could be used to improve the malignancy classification accuracy (MCA) at the expense of a reduced total accuracy. Table 2 shows the effect of three different malignancy bias factors for randomly generated training data sets of size 200 and a chromosome population size of 8,000.

While there was a slight reduction in the standard deviation of the malignancy classification accuracy, there was not a meaningful improvement in the average MCA. With bias factors larger than 4, there may have been more dramatic improvements in the MCA.

Malignancy Bias Factor	Total Accuracy	$\sigma_{Total\ Accuracy}$	MCA	σ_{MCA}
1 (Standard)	91.6%	2.30%	91.0%	5.18%
2	92.0%	2.00%	91.1%	4.06%
4	91.8%	2.16%	91.0%	4.25%

Table 2 – Effect of a Malignancy Bias Factor on the Classifier Accuracy

Conclusions and Future Work

The overall accuracy of our algorithm was high at over 90% on average in the most stringent cases. Others ([4], [5]) have achieved classification accuracies greater than 94% for this data set. With further adjustment including improving the handling of the mitoses weight term, we believe that the average accuracy of our algorithm as well as the total variance in the results could be improved. What is more, it would be worthwhile to investigate how the classification accuracy would change if techniques other than a genetic algorithm (e.g. support vector machine, neural networks, etc.) were used to generate the classifier.

Data Set Overview

The source data set is entitled the "Wisconsin Breast Cancer Database" and was created by a team from the University of Wisconsin Hospitals. The data set [2][1] consists of 699 patient tumors[2]. Each tumor has 9 distinct features that were individually assigned a value on a 1 to 10 scale; the features are listed below with brief descriptions of the biological differences between benign and malignant tumors [3].

1. **Clump Thickness:** Benign cells tend to be clumped in monolayers while malignant cells are usually grouped in multilayers.
2. **Uniformity of Cell Size:** Benign cells are more uniform in size while the size of malignant cells can vary significantly.
3. **Uniformity of Cell Shape:** Benign cells have smooth and round edges with a surrounding fibrous capsule that is very well-circumscribed. In contrast, the edges of the malignant cells are usually very distinct and lack this uniformity.
4. **Marginal Adhesion:** A malignant mass is mobile and not attached to surrounding tissue. Benign masses tend to stick more tightly together.
5. **Single Epithelial Cell Size:** Epithelial cells line the cavities and surfaces of structures throughout the body. In benign tumors, surrounding epithelial cells form a single layer with normal cell size; malignant tumors usually have significantly enlarged epithelial cells.
6. **Bare Nuclei:** A bare nucleus is devoid of surrounding cytoplasm (i.e. the rest of the cell). This phenomenon is more commonly seen in benign tumors.
7. **Bland Chromatin:** In benign tumors, the nucleus generally has a uniform texture. In contrast, cancerous cells tend to have coarser nuclei.
8. **Normal Nucleoli:** Nucleoli are small structures in the cell nucleus. In benign cells, a nucleolus is very small and barely (if at all) visible. Malignant cells have more prominent nucleoli, and in some cases, they are more numerous as well.
9. **Mitoses:** It is the process in which the cell replicates and divides. Mitosis is rapid and uncontrolled in malignant cells.

In the source data set, malignant tumors are marked with a "4" while benign tumors are marked with a "2".

The data set is published as a text file in comma separated variable (CSV) format.

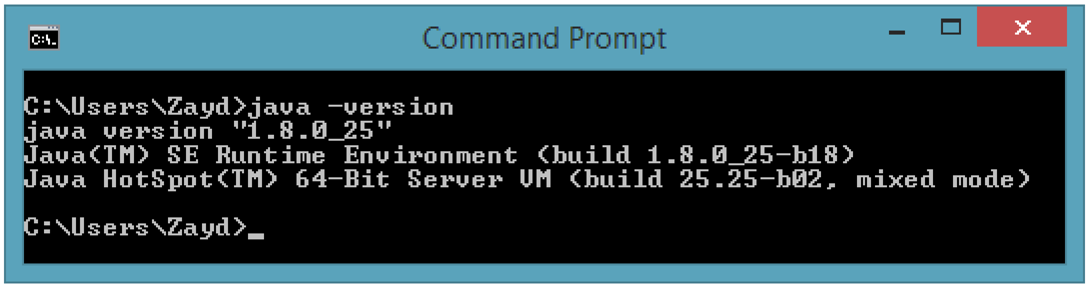
[1] The data set is available in the University of California, Irvine's Machine Learning Repository. A link to the data set is: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original).

[2] 16 of the 699 tumors have incomplete data so only 683 instances are used in our analysis.

Running the Breast Cancer Classifier Genetic Algorithm

Note: The followings directions were tested on a Windows 8 based PC. The procedure may vary slightly for other platforms (e.g. Apple OSX, Linux/Unix). However, as long as a Java Virtual Machine (i.e. the runtime environment for Java applications) is available on that platform and is properly configured, the tool should still be runnable.

The Breast Cancer Classifier Genetic Algorithm was developed in the Java programming language. Hence, to run the tool, the Java Runtime Environment is required.[3] Before attempting any of the following steps, ensure that Java has been properly installed and is accessible from the command line. To confirm this, open the command prompt, and type "java –version". Figure 5 shows that Java version 8 update 25 is installed (i.e. "1.8.0_25"). If a previous version is installed or an error message appears, then Java should be installed or updated before continuing.



```
C:\Users\Zayd>java -version
java version "1.8.0_25"
Java(TM) SE Runtime Environment (build 1.8.0_25-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)

C:\Users\Zayd>_
```

Figure 5 – Checking whether Java is Correctly Installed and Configured

Once Java has been correctly installed, there are two files needed to run the tool. The first is the Java Archive (JAR) file that contains all of the compiled Java byte code files; this file is named "Breast_Cancer_GA_Classifier.jar". The second is the breast cancer source data, which is named "breast-cancer-wisconsin.data.txt"; this text file's name should not be changed if the tool is run under default configurations.

From the command line, the tool is run by invoking the command:

```
java -jar Breast_Cancer_GA_Classifier.jar
```

An example tool output is shown in Appendix B. The tool supports a set of command line options that allow the user to modify the genetic algorithm's settings directly from the command line; these are summarized in Table 3. For example, to run the genetic algorithm 10 times with 8,000 chromosomes per generation while outputting to a file, the command would be:

```
java -jar Breast_Cancer_GA_Classifier.jar -OF -NR 10 -SS 8000
```

As a note, the command line option flags are commutative.

[3] To download the Java Runtime Environment for free, visit: https://java.com/en/download/index.jsp.

Command Line Flag	Flag Description	Default Value
-NR *n*	**Number of Repeats Flag** – Loops the entire genetic algorithm (including creating the training and verification data sets) *n* times. This is useful when performing repeatability analysis of the algorithm. **Note**: *n* must be an integer number.	*n* = 1
-BAL *shuffle*	**Malignant Population Balancer Flag** – Balances the malignant and benign patients proportionally between the training and verification data sets. Valid values for the *shuffle* flag are "1" to randomly shuffle the data set members before dividing them into the training and verification data sets while "0" assigns the elements based off their order in the source CSV file.	*shuffle* = 1
-SS *n*	**Maximum Solution Size Flag** – Modifies the genetic algorithm's chromosome population size to *n*. **Note**: *n* must be an integer number.	*n* = 1000
-OF	**Output to File Flag** – Instructs the genetic algorithm to store the algorithm's settings and results to a CSV file named "GA Results.csv". If a file with the name "GA Results.csv" already exists, the tool appends all new information to the end of that file.	File Output Disabled
-TDS *n*	**Training Data Set Size Flag** – Modifies the size of the training data set to *n*. **Note**: *n* must be an integer number.	*n* = 200
-MP *n*	**Malignancy Bias Factor Flag** – Modifies the dedicated penalty/reward for the incorrect/correct classification of malignant tumors to *n*. **Note:** *n* must be greater than or equal to 1.	*n* = 1

Table 3 – Genetic Algorithm Command Line Options

List of References

[1] Wolberg, W.H. & Mangasarian, O.L. (1990). Multisurface Method of Pattern Separation for Medical Diagnosis Applied to Breast Cytology. In Proceedings of the National Academy of Sciences, 87, 9193-9196.

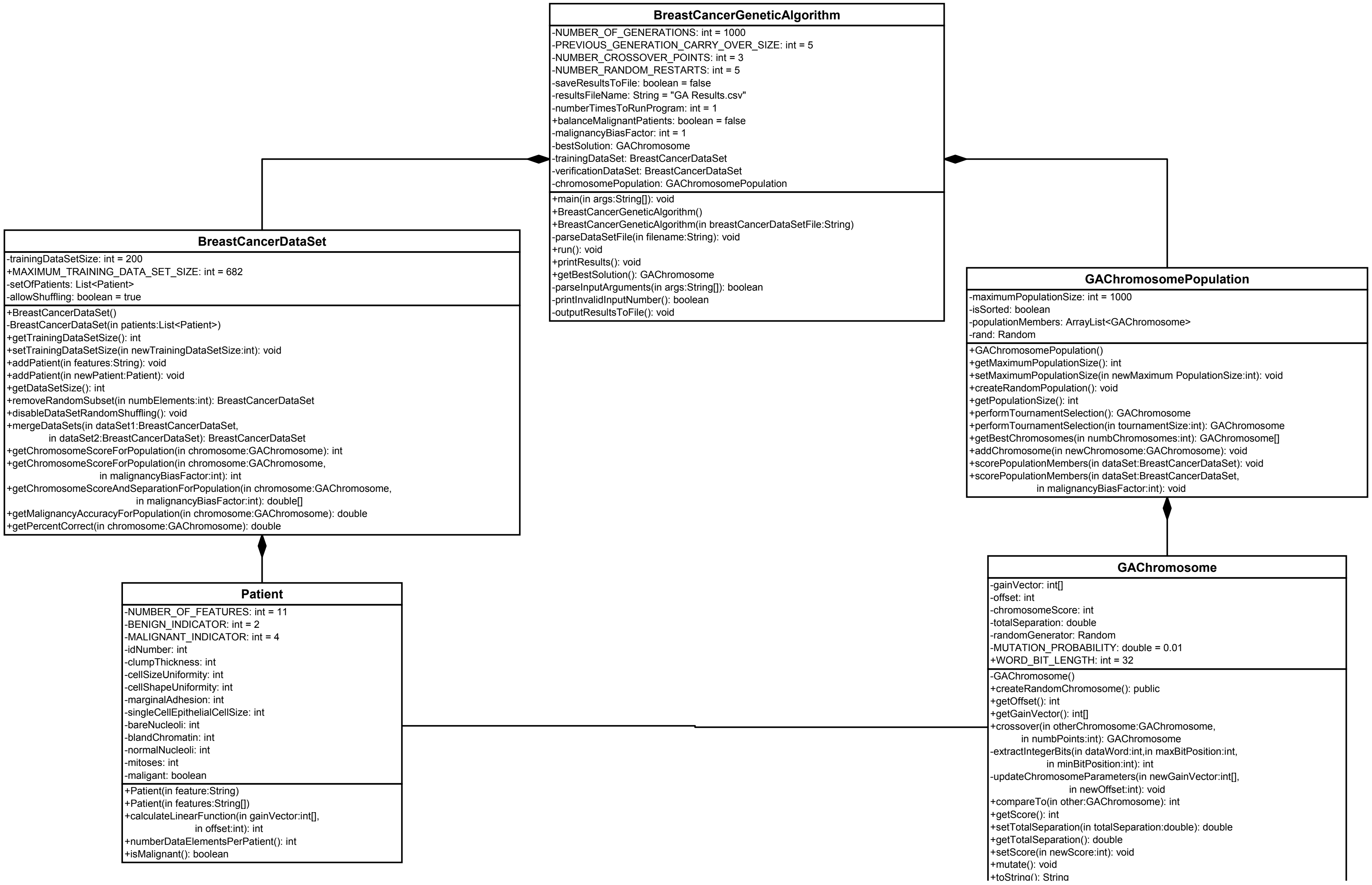
[2] Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[3] Salama et. al. “Breast Cancer Diagnosis on Three Different Datasets Using Multi-Classifiers”. International Journal of Computer and Information Technology. September 2012.

[4] Setiono, R. and Liu, H.. “NeuroLinear: From Neural Networks to Oblique Decision Rules”. Neurocomputing, 17. 1997.

[5] Abbass et. al. “An Evolutionary Neural Networks Approach for Breast Cancer Diagnosis.” Artificial Intelligence in Medicine, 17. 2002.

Appendix A – UML Class Diagram for the Breast Cancer Classifier Genetic Algorithm



```
BreastCancerGeneticAlgorithm
-NUMBER_OF_GENERATIONS: int = 1000
-PREVIOUS_GENERATION_CARRY_OVER_SIZE: int = 5
-NUMBER_CROSSOVER_POINTS: int = 3
-NUMBER_RANDOM_RESTARTS: int = 5
-saveResultsToFile: boolean = false
-resultsFileName: String = "GA Results.csv"
-numberTimesToRunProgram: int = 1
+balanceMalignantPatients: boolean = false
-malignancyBiasFactor: int = 1
-bestSolution: GAChromosome
-trainingDataSet: BreastCancerDataSet
-verificationDataSet: BreastCancerDataSet
-chromosomePopulation: GAChromosomePopulation
+main(in args:String[]): void
+BreastCancerGeneticAlgorithm()
+BreastCancerGeneticAlgorithm(in breastCancerDataSetFile:String)
-parseDataSetFile(in filename:String): void
+run(): void
+printResults(): void
+getBestSolution(): GAChromosome
-parseInputArguments(in args:String[]): boolean
-printInvalidInputNumber(): boolean
-outputResultsToFile(): void
```

```
BreastCancerDataSet
-trainingDataSetSize: int = 200
+MAXIMUM_TRAINING_DATA_SET_SIZE: int = 682
-setOfPatients: List<Patient>
-allowShuffling: boolean = true
+BreastCancerDataSet()
-BreastCancerDataSet(in patients:List<Patient>)
+getTrainingDataSetSize(): int
+setTrainingDataSetSize(in newTrainingDataSetSize:int): void
+addPatient(in features:String): void
+addPatient(in newPatient:Patient): void
+getDataSetSize(): int
+removeRandomSubset(in numbElements:int): BreastCancerDataSet
+disableDataSetRandomShuffling(): void
+mergeDataSets(in dataSet1:BreastCancerDataSet,
        in dataSet2:BreastCancerDataSet): BreastCancerDataSet
+getChromosomeScoreForPopulation(in chromosome:GAChromosome): int
+getChromosomeScoreForPopulation(in chromosome:GAChromosome,
                in malignancyBiasFactor:int): int
+getChromosomeScoreAndSeparationForPopulation(in chromosome:GAChromosome,
                in malignancyBiasFactor:int): double[]
+getMalignancyAccuracyForPopulation(in chromosome:GAChromosome): double
+getPercentCorrect(in chromosome:GAChromosome): double
```

```
GAChromosomePopulation
-maximumPopulationSize: int = 1000
-isSorted: boolean
-populationMembers: ArrayList<GAChromosome>
-rand: Random
+GAChromosomePopulation()
+getMaximumPopulationSize(): int
+setMaximumPopulationSize(in newMaximum PopulationSize:int): void
+createRandomPopulation(): void
+getPopulationSize(): int
+performTournamentSelection(): GAChromosome
+performTournamentSelection(in tournamentSize:int): GAChromosome
+getBestChromosomes(in numbChromosomes:int): GAChromosome[]
+addChromosome(in newChromosome:GAChromosome): void
+scorePopulationMembers(in dataSet:BreastCancerDataSet): void
+scorePopulationMembers(in dataSet:BreastCancerDataSet,
                in malignancyBiasFactor:int): void
```

```
Patient
-NUMBER_OF_FEATURES: int = 11
-BENIGN_INDICATOR: int = 2
-MALIGNANT_INDICATOR: int = 4
-idNumber: int
-clumpThickness: int
-cellSizeUniformity: int
-cellShapeUniformity: int
-marginalAdhesion: int
-singleCellEpithelialCellSize: int
-bareNucleoli: int
-blandChromatin: int
-normalNucleoli: int
-mitoses: int
-maligant: boolean
+Patient(in feature:String)
+Patient(in features:String[])
+calculateLinearFunction(in gainVector:int[],
                in offset:int): int
+numberDataElementsPerPatient(): int
+isMalignant(): boolean
```

```
GAChromosome
-gainVector: int[]
-offset: int
-chromosomeScore: int
-totalSeparation: double
-randomGenerator: Random
-MUTATION_PROBABILITY: double = 0.01
+WORD_BIT_LENGTH: int = 32
-GAChromosome()
+createRandomChromosome(): public
+getOffset(): int
+getGainVector(): int[]
+crossover(in otherChromosome:GAChromosome,
        in numbPoints:int): GAChromosome
-extractIntegerBits(in dataWord:int,in maxBitPosition:int,
                in minBitPosition:int): int
-updateChromosomeParameters(in newGainVector:int[],
                in newOffset:int): void
+compareTo(in other:GAChromosome): int
+getScore(): int
+setTotalSeparation(in totalSeparation:double): double
+getTotalSeparation(): double
+setScore(in newScore:int): void
+mutate(): void
+toString(): String
```

Figure 6 – UML Class Diagram of the Breast Cancer Classifier Program

Appendix B – Genetic Algorithm Sample Output

Figure 7 is a sample output of the genetic algorithm. For each of the nine features in the data vector, a signed, 32-bit weight is assigned. The algorithm prints to the console the integer weight for each of the parameters. For example, in the figure below, the integer multiplier for normal nucleoli is 251085468 while the mitoses vector element is multiplied by the weight -10254865.

```
After run #1, the percent correct on the training set is: 94.50
After run #2, the percent correct on the training set is: 96.00
After run #3, the percent correct on the training set is: 96.00
After run #4, the percent correct on the training set is: 96.00
After run #5, the percent correct on the training set is: 96.00
On the training set, the score for the best solution is: 192
The percent correct is: 96.00%.

The linear function weights are:
Mitoses Weight: -455525836
Clump Thickness Weight: -251376661
Cell Size Uniformity Weight: 925633042
Cell Shape Uniformity Weight: 265173415
Marginal Adhesion Weight: -6570220
Single Epithelial Cell Size Weight: -265601765
Bare Nucleoli Weight: 466751329
Bland Chromatin Weight: -219178877
Normal Nucleoli Weight: 494713474
Offset Weight: -2147121672
On the verification set, the score for the best solution is: 455
The percent correct is: 94.20%.
The percentage of malignant tumors correctly categorized is: 96.41%.
```

Figure 7 – Example Breast Cancer Genetic Classifier Output