**A Linear Classifier for Determining the**

**Presence of Malignancy in Breast Tumors**

**Final Report**

**Clemens Vallejo**

**Zayd Hammoudeh**

**Table of Contents**

[Table of Figures 3](#_Toc404909106)

[Introduction 4](#_Toc404909107)

[Linear Classification 4](#_Toc404909108)

[Genetic Algorithm Overview 4](#_Toc404909109)

[Chromosome (Solution) Structure 5](#_Toc404909110)

[Classification Strategy and Threshold 5](#_Toc404909111)

[Chromosome Population Size 6](#_Toc404909112)

[Reproduction Selection Algorithm 6](#_Toc404909113)

[Fitness Function 6](#_Toc404909114)

[Crossover 7](#_Toc404909115)

[Mutation Frequency 7](#_Toc404909116)

[Generation Count 7](#_Toc404909117)

[Random Restarts 7](#_Toc404909118)

[Breast Tumor Classifier Program Overview 7](#_Toc404909119)

[Genetic Algorithm Results 8](#_Toc404909120)

[Conclusions and Future Work 9](#_Toc404909121)

[Data Set Overview 10](#_Toc404909122)

[Running the Breast Cancer Classifier Genetic Algorithm 11](#_Toc404909123)

[List of References 13](#_Toc404909124)

[Appendix A – Genetic Algorithm Sample Output 14](#_Toc404909125)

[Appendix B – UML Class Diagram for the Breast Cancer Classifier Genetic Algorithm 15](#_Toc404909126)

# **Table of Figures**

[Figure 1 – Flow Diagram of a Classification Algorithm 4](#_Toc404909127)

[Figure 2 – Pseudocode for the Breast Cancer Genetic Algorithm 5](#_Toc404909128)

[Figure 3 – Block Diagram of the Components in the Breast Cancer Classifier Program 8](#_Toc404909129)

[Figure 4 – Relationship between Chromosome Population Size and Classifier Accuracy 9](#_Toc404909130)

[Figure 5 – Checking whether Java is Correctly Installed and Configured 11](#_Toc404909131)

[Figure 6 – Example Breast Cancer Genetic Algorithm Output 14](#_Toc404909132)

# 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. The basic flow of a classification algorithm is shown in Figure 1.

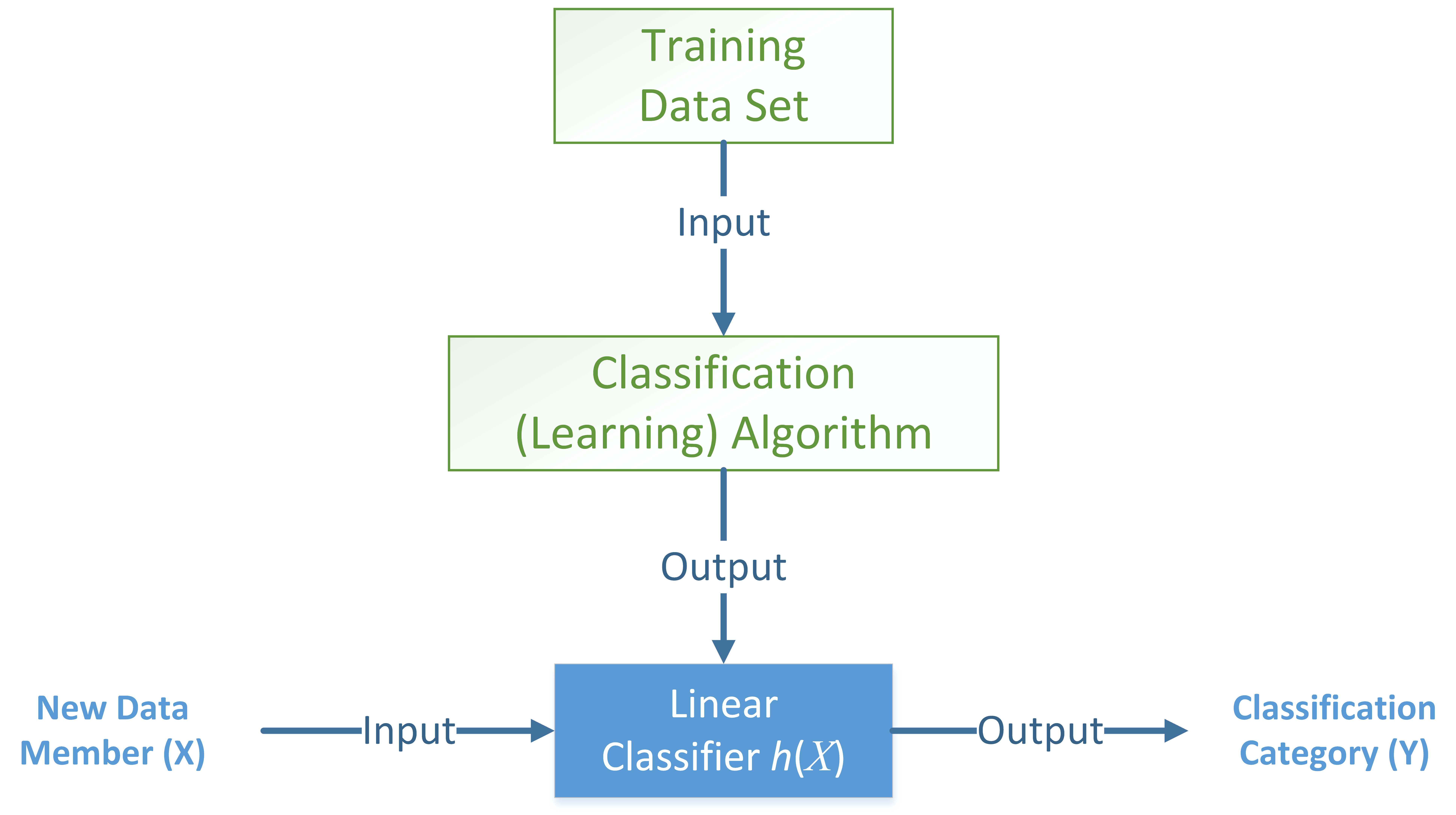


Figure – Flow Diagram of a Classification Algorithm

This paper describes an algorithm that classifies breast cancer tumors as either malignant or benign. For a description of the source data set, see section “Data Set Overview”.

## Linear Classification

The input to a classification algorithm is generally a set data vectors; each data vector, , is *N*-dimensional. In a linear classification algorithm, there is a single *N-*dimensional weight vector , where each element, , in represents a scalar weight for the *i*th dimension. A linear classifier, , is defined by:

|  |  |
| --- | --- |
|  | ( 1 ) |

is the transpose of the weight vector, , and the centered dot symbol “” is the dot/scalar product of the two vectors. is an optional scalar offset term. Since a dot product is used and given that the offset term, , is a scalar, the linear classifier, , is also a scalar. Depending on this scalar value of and a set of classification threshold(s), each data member is classified into a category, , as shown in Figure 1. In this paper, the linear classifier will categorize breast cancer tumors as either malignant or benign.

# Genetic Algorithm Overview

There are multiple techniques that can be used to determine appropriate values for the linear classifier terms. and , including support vector machine, neural networks, etc. In this paper, we usea genetic algorithm to generate the linear classifier constraints. It was selected due to its overall robustness and low execution overhead, both in terms of memory and computational resources.

A genetic algorithm (GA) 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 the 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. Over a series of iterations (called generations), pairs of chromosomes from the previous generation (parents) are merged to form the new chromosomes that comprise the successor generation. The process where the two paternal chromosomes are merged to form the descendent chromosome is known as crossover. After crossover, the successor chromosome undergoes mutation where part of the solution is randomly changed. After a specified number of generations, a new chromosome seed population can be optionally created. At the end of all generations and random starts, the best solution (i.e. the linear classifier’s weight vector, , and offset scalar, ) is returned.

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 – Pseudocode for the Breast Cancer Genetic Algorithm

The following subsections review the 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 (). 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 classified as benign while any data member whose classifier function value is positive is classified 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 algorithm is 1000 since it provided a reasonable tradeoff between solution quality and overall algorithm execution time.

## Reproduction Selection Algorithm

Tournament selection in a genetic algorithm involves randomly picking *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 (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, ; it quantifies number of dataset members that were correctly classified.

|  |  |
| --- | --- |
|  | ( 2 ) |

is defined in eq. and returns an integer reward for all correct classification of a tumor as either benign or malignant.

|  |  |
| --- | --- |
|  | ( 3 ) |

where *M* is the size of the population, *i* represents the *i*th data member in the dataset, and is the value linear classifier function for patient vector, . is a normalizing scalar defined by the relation in eq. ; this normalizing scalar is used to reward correct classification of benign tumors, which will have negative scores.

|  |  |
| --- | --- |
|  | ( 4 ) |

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

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

|  |  |
| --- | --- |
|  | ( 5 ) |
|  |  |

## Crossover

Crossover is the process of combining two parent chromosomes to form the 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 time.

## 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 because it can corrupt otherwise good solutions. The bit mutation frequency is set to 1% in our solution, which, while high, showed no deleterious 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 configuration below 1000 generations; by limiting the generation count, performance is improved by eliminating the need to continue examining solutions with limited likelihood of further improvement.

## Random Restarts

In a random restart, all members of the current chromosome population is discarded, with the exception of the best solution which is stored for future comparison. A 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

is a block diagram of the components of the program that develops the linear classifier . The breast cancer data set, which is in Comma Separated Variable (CSV) format, 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 created; the default setting in the program is that the two data sets are generated randomly. However, as explained in section “Running the Breast Cancer Classifier Genetic Algorithm”, the user does have 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 it classifies patient tumors in the verification data set.

Our program is implemented in the Java programming language. Specific benefits of Java include its portability across platforms, large user base, and extensive built-in libraries.

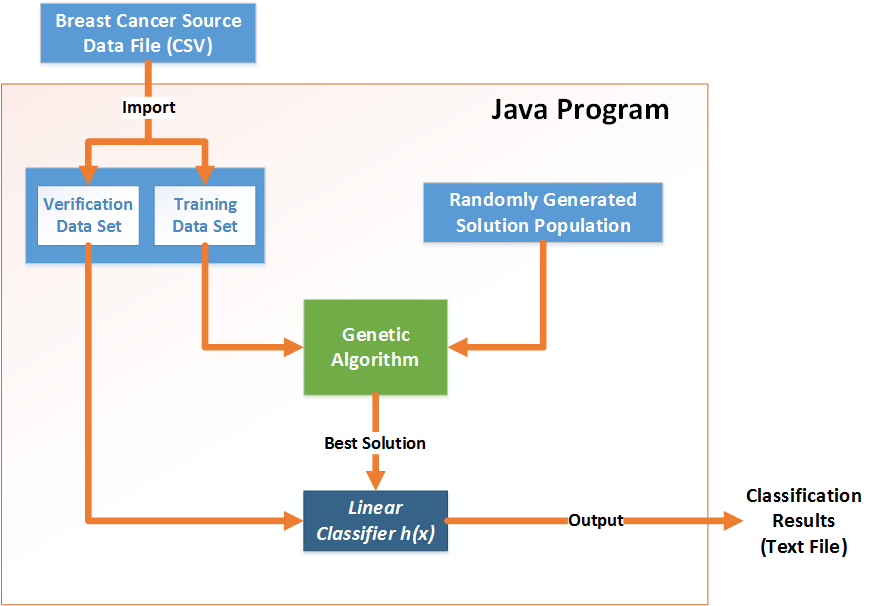


Figure – Block Diagram of the Components in the Breast Cancer Classifier Program

Java is primarily an object-oriented programming language; what is more, classification problems (especially genetic algorithms) are generally conducive to being programmed using an object based methodology. As such, our implementations heavily uses on the object-oriented paradigm in our implementation. is a UML class diagram of our programs. The five classes and their respective descriptions are listed 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 dataset. Each Patient object is categorized as either malignant or benign and has values for the nine features described previously.
3. **BreastCancerDataSet** – Aggregator of objects of the Patient class. The two 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 where 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 (i.e. linear classifier). This linear classifier is used to classify 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 verification 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 algorithm with respect to this type of error.

To determine the appropriate size of the chromosome population, 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 solution set population size and the total accuracy as well as the MCA. For each population size, the algorithm was looped 100 times; the graph shows the average accuracy as well as the standard deviation of the accuracy for each population size.

Figure – Relationship between Chromosome Population Size and Classifier Accuracy

When the solution size was larger than 8000, there was no statistically significant improvement in total accuracy or MCA. While there was a slight decrease in the variation for the total accuracy, it was very minor. As such, for each subsequent experiment, we used a population size of 8,000.

# Conclusions and Future Work

The overall accuracy of our algorithm was high at over 90% in most cases. Work has been done by other authors that returned classification accuracies greater than 94% ([ 4 ] & [ 5 ]).

# Data Set Overview

The source dataset is entitled the “Wisconsin Breast Cancer Database” and was created by a team from the University of Wisconsin Hospitals. The dataset [ 2 ][[1]](#footnote-1) consists of 699 patient tumors[[2]](#footnote-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). They are more typically 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, the nucleolus is very small and barely visible (if at all). 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.

Those tumors which are malignant are marked with a “4” while benign tumors are marked with a “2”. The dataset is published as a text file in comma separated variable (CSV) format.

# 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, the tool should still be runnable.

The Breast Cancer Classifier Genetic Algorithm was developed in the Java programming language. Hence, to run the tool, it is required that the java runtime environment be installed, which is available for free from Oracle.[[3]](#footnote-3) Before attempting any of the following steps, ensure that Java has been properly installed and accessible from the command line. To do this, open the command prompt and type “java –version”. Figure 5 shows that Java version 8 update 25 is installed. If a previous version is installed or an error message appears, Java is not correctly installed or configured.

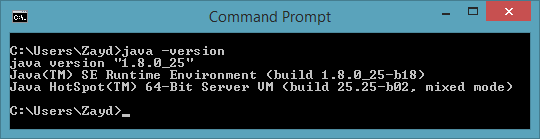


Figure – Checking whether Java is Correctly Installed and Configured

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 set and is the file named “breast-cancer-wisconsin.data.txt”; this 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 A. 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 1. For example, to run the genetic algorithm 10 times with a 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

Note the command line option flags are commutative.

|  |  |  |
| --- | --- | --- |
| **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 dataset members before dividing them into the training and verification datasets 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 when the algorithm is run a file with the name “GA Results.csv” already exists, it appends all new information to the end of the file. | File Output Disabled |
| -TDS *n* | **Training Dataset Size Flag** – Modifies the number of elements in the Genetic Algorithm’s 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 – Genetic Algorithm Command Line Options

# List of References

|  |  |
| --- | --- |
| [ ] | 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. |
| [ ] | 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. |
| [ ] | Salama et. al. “Breast Cancer Diagnosis on Three Different Datasets Using Multi-Classifiers”. International Journal of Computer and Information Technology. September 2012. |
| [ ] | Setiono, R. and Liu, H.. “NeuroLinear: From Neural Networks to Oblique Decision Rules”. Neurocomputing, 17. 1997. |
| [ ] | Abbass et. al. “An Evolutionary Neural Networks Approach for Breast Cancer Diagnosis.” Artificial Intelligence in Medicine, 17. 2002. |
|  |  |
|  |  |
|  |  |

# Appendix A – Genetic Algorithm Sample Output

Figure 6 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 screen 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

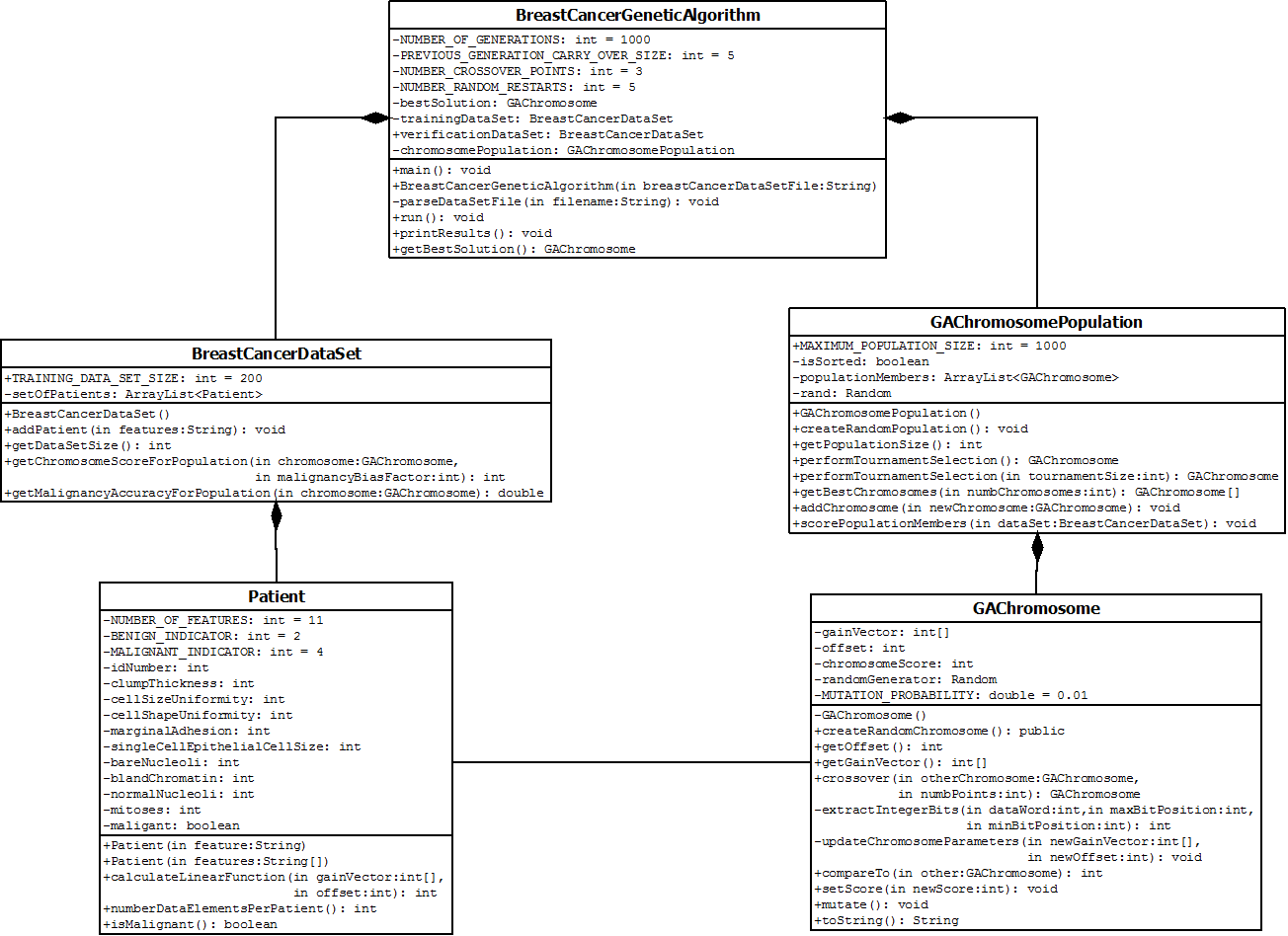
On the verification set, the score for the best solution is: 455

The percent correct is: 94.20%.

The percentage of malignant tumors correctly categorized is: 96.41%.

Figure – Example Breast Cancer Genetic Algorithm Output

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

****

1. The dataset is available in the University of California, Irvine’s Machine Learning Repository. A link to the dataset is: <https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)>. [↑](#footnote-ref-1)
2. 16 of the 699 tumors have incomplete data so only 683 instances are used in our analysis. [↑](#footnote-ref-2)
3. To download the Java for your computer, visit: <https://java.com/en/download/index.jsp>. As of the writing of this report, the latest Java was version 8 update 25. [↑](#footnote-ref-3)