Developing a Linear Classifier for Determining the Presence of Malignancy in Breast Cancer Tumors

Team Members:

- Clemens Vallejo
- Zayd Hammoudeh

Project Summary:

Use a genetic algorithm to generate a linear classifier that can determine whether a breast cancer tumor is malignant or benign.

Classification Analysis Overview:

In predictive analytics, there are two primary classes of problems. The first type is regression; these problems focus on using an existing, known (i.e. training) data set to generate numerical (usually continuous) models. The models quantify how the variables in the data set interact to form the numerical, output result. One of the most common uses of a regression is least squares analysis to create a best fit line for a set of two dimensional data.

Our project will focus on the other primary domain of predictive analytics, namely **classification**. A classification algorithm involves using a training data set to generate models that can classify future data members ($\vec{x}$) into one of a finite set of categories (y). If the total number of classification categories is two (as in our problem), then it is referred to as binary or Boolean classification. Figure 1 is an example block diagram of the structure of a classification algorithm.

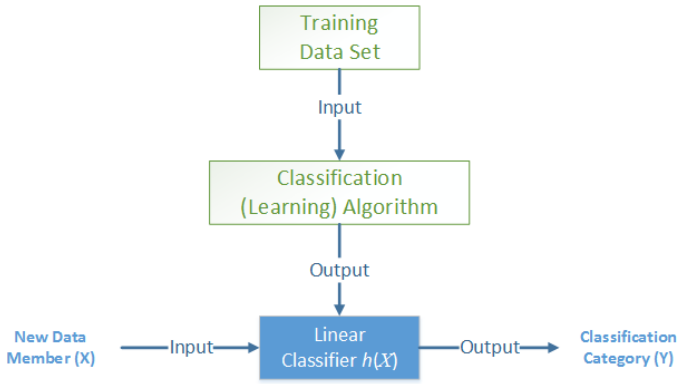

Figure 1 – Classification Algorithm Block Diagram

A decision boundary is used by a classification algorithm to determine whether a data member is classified into one category or another. These boundaries are most often represented as equations and can be either linear or nonlinear depending on the type of problem and the source data set. Linear decision boundaries are most commonly used because of their relative simplicity to generate as well as their low computational cost to calculate. If a decision boundary is linear, then it is referred to as a **linear classifier** (notation for linear classifiers in this paper is $h(\vec{x})$).

Linear classifiers operate on individual data members ($\vec{x}$); these vectors are *N*-dimensional, where *N* is the number of features (i.e. variables) that are present for each data member. The linear classifier has two additional terms. The first is $\vec{w}$, which is an *N*-dimensional **weight vector**. Each element, w_i, in the weight vector $\vec{w}$ is the scalar weight associated with the i^{th} element, x_i, in the input data vector, $\vec{x}$. The linear classifier's second term is a scalar **offset term**, w_0. A linear classifier $h(\vec{x})$ is defined by the equation:

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

Note $\vec{w}^T$ is the transpose of the weight vector, $\vec{w}$, and the centered dot symbol " $\cdot$ " is the dot/scalar product of the two vectors. 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 the value of $h(\vec{x})$ and the classification threshold(s), the data member is classified into a category, y.

Genetic Algorithm Overview:

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 individual 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 (**parents**) are selected at random based off their relative fitness; these pairs then reproduce to form a new chromosome (**child**) that is part of the successor generation. The process where the two paternal chromosomes are merged to form the descendent chromosome is known as **crossover**. After crossover, successor chromosomes may also undergo a **mutation** process where part of their solution is randomly changed. Figure 2 is pseudocode showing the basic flow of execution for a genetic algorithm.

```
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) ≤ 1000 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
```

Figure 2 – Genetic Algorithm Pseudocode

In our project, we will use a genetic algorithm to determine the weight vector, $\vec{w}$, and the offset term, w_0 of the linear classifier.

Source Dataset:

The source dataset we will use for analysis is entitled "Wisconsin Breast Cancer Database"[1]. It was created by a team led by Dr. William H. Wolberg from the University of Wisconsin Hospitals. The dataset consists of 699 patient

[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).

tumors that are classified as either benign or malignant[2]. Each tumor has 9 distinct features that were rated on a 1 to 10 scale; the features are:

1. Clump Thickness
2. Uniformity of Cell Size
3. Uniformity of Cell Shape
4. Marginal Adhesion
5. Single Epithelial Cell Size
6. Bare Nuclei
7. Bland Chromatin
8. Normal Nuclei
9. Mitoses

We will divide the data set into two disjoint sets. The first set of *M* (e.g. 200) elements will serve as the training set that we will use in the genetic algorithm to create the linear classifier. The remaining data elements will be placed into a second verification set, which will be used to quantify the quality of the resulting linear classifier.

Software Program Implementation Overview:

We plan to implement our genetic algorithm and linear classifier in Java. The reasons we plan to use Java is:

1. **Portability** – The Java Virtual Machine (Java's runtime environment) is available for most platforms (e.g. Windows, Macintosh, Linux, Android, iOS, etc), and a program compiled on one platform can be run on all other platforms that have a Java Virtual Machine. In contrast, C/C++ code must be recompiled for each platform, and a vector based tool like Matlab requires a costly software package.

2. **Large User Base** – Java is rated by TIOBE as the second most commonly used programming language today[3]. This allows any code we developed to be readily understood and modified by a large developer community.

3. **Extensive Built-In Libraries** – Java comes bundled with many data structure libraries that reduces the amount of low level coding that is required by the programmer.

Figure 3 is a block diagram showing the expected components in our Java program. As described previously, the breast cancer data set, which is in CSV format, will be imported into our program and split into two disjoint sets (i.e. training and verification). The training set along with the randomly generated solution set is fed into the genetic algorithm. Over a fixed number of generations, the genetic algorithm generates a linear classifier; the best linear classifier from the algorithm is then graded against the verification dataset to determine the system's overall quality.

[2] 16 of the 699 tumors have incomplete data so only 683 instances will be used in our analysis.
[3] TOIBE ratings available at http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Accessed September 21, 2014.

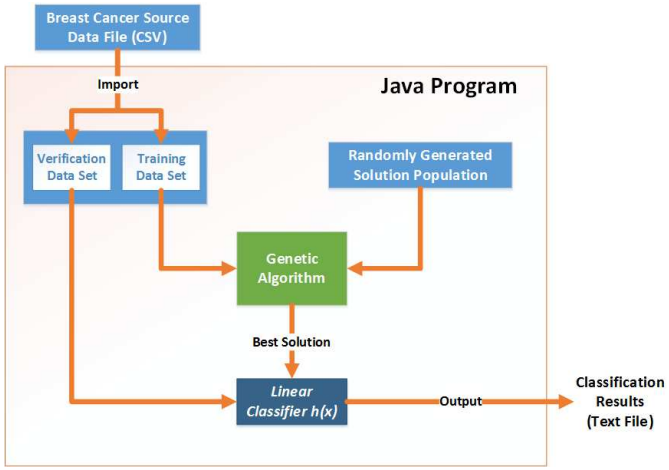

Figure 3 – Block Diagram of the Expected Components in our Breast Cancer Classifier Program

Genetic Algorithm Implementation Plan:

This section outlines the forecasted settings for the genetic algorithm that we will use. These are only preliminary values and may change in the final implementation depending on the results we observe during development and testing.

- **Classification Strategy and Threshold:** Any data member whose classifier function value is negative will be classified as benign while any data member whose classifier function value is positive is malignant. This is the simplest algorithm to implement and understand for a user. There is significant latitude when selecting a classification strategy since the algorithm simply adjusts during population evolution to whatever threshold is selected.

- **Population Size:** 1000. A large population increases solution diversity. However, as the population size increases, the incremental population diversity decreases. As such, population diversity must be balanced against computation complexity. A population size of 1000 is common for problems of this type.

- **Number of Generations:** 100 will be our starting point, but we may use solution convergence instead.

- **Chromosome (Solution) Structure:** The data set has 9 features, and there is an additional term in the linear classifier function for the offset. Each weight will be *b* bits in length; our current plan is to have *b* be 32 (i.e. the size of an integer in Java). The weights will be two's complement signed integers. Given the 9 features and the offset term, the chromosome will be $10 * b$ bits in length (e.g. 320 bits if a single weight is 32 bits in length).

- **Crossover Operator:** Three point crossover to increase the amount of splicing between parent chromosomes. If no significant difference is shown with one point crossover, we will use that due to its reduced computational complexity.

- **Mutation Frequency:** The mutation frequency is dependent on the chromosome length described previously; if the two were not coupled, then the mutation frequency could be too high or too low. We expect a mutation frequency in the range of $10 * b$ to $500 * b$ will be optimal.

- **Fitness Function** – The fitness function f measures the quality of any solution. Eq. (2) is our current anticipated fitness function.

$$f = \sum_{i=1}^{M} \theta_i \cdot h(\vec{x}_i) \qquad (2)$$

M is the number of data members in the training set; i represents the i^{th} data member in the set, and $h(\vec{x}_i)$ is the value linear classifier function defined by eq. (1). θ_i is a normalizing scalar defined by the relation in eq. (3); this normalizing scalar is used to reward correct classification of benign tumors, which will have negative classifying scores (as described in the classification strategy section).

$$\theta_i = \begin{cases} 1, & \text{if tumor } i \text{ is malignant} \\ -1, & \text{if tumor } i \text{ is benign} \end{cases} \qquad (3)$$

- **Reproduction Selection Algorithms:** Many variants are available for this including tournament, roulette-wheel, and truncation selection. More consideration is needed here. However, roulette wheel (also known as fitness proportionate selection) is the most likely choice due to its ability to balance rewarding good/fit solutions with solution diversity.

Linear Classifier Performance Measures:

After the linear classifier is generated, it is used to classify the remaining data elements in the verification data set. We will use three primary metrics to quantify the quality of our linear classifier. They are:

1. **Total Accuracy** – This is the most straightforward of the quality measures and is simply the ratio of the number of correct classifications to the total size of the verification data set.

2. **False Negative Rate** – A false negative is the case where a tumor is classified as benign while it is really malignant. This is the most serious type of classification error.

3. **False Positive Rate** – A false positive is the case where a tumor is classified as malignant while it is really benign. This is a less serious error than a false negative. Depending on our algorithm's overall classification accuracy, we may tune our fitness function to deliberately raise more false positives to reduce the likelihood of a false negative.

Depending on the results from our three primary metrics, we may also investigate the marginality of the incorrect classification results to further gauge the accuracy of our classification methodology.

A few other papers have been written analyzing the results in this dataset. Our results will also be compared versus their findings to further gauge our approach's success.

Biological Analysis:

The genetic algorithm will output a weight vector, $\vec{x}$. Those features in the vector that have the most correlation with a malignant tumor will have the largest positive weight while those features that are most correlated with benign tumors will have the most negative weights. The feature effects that are identified by our genetic algorithm will be examined from a biological perspective to see if those features can be associated with known biological effects and relations.