SOCIETY OF ACTUARIES

Article from:

Forecasting & Futurism

December 2013 – Issue 8

Diagnosing Breast Tumor Malignancy with a Genetic Algorithm and RBF Network

By Jeff Heaton

In this article I demonstrate how to use a genetic algorithm to devise a radial basis function (RBF) network to predict the malignancy of breast tumors. The genetic algorithm is implemented in Microsoft Visual C#. This determination is made using the following nine attributes collected from a tumor.

- Clump thickness
- Uniformity of cell size
- Uniformity of cell shape
- Marginal adhesion
- Single epithelial cell size
- Bare nuclei
- Bland chromatin
- Normal nucleoli
- Mitoses.

I used training obtained by Dr. William H. Wolberg, University of Wisconsin Hospitals.[1] Using these attributes I constructed an RBF network to perform malignancy classification into two classes. These two classes were either malignant or benign.

This program is meant to be very versatile and extendable. This same program can be used to perform either classification or regression on a wide array of data sets. If you wish to use an RBF network, only changes to the script.xml file are necessary. If you would like to create your own model, you can easily add a model class to the C# source code. I tested this program on several data sets from the University of California at Irvine Machine Learning Dataset Repository.[2]

USING THE PROGRAM

This program can be downloaded from the SOA Forecasting & Futurism site at the following link: http://www.soa.org/news-and-publications/newsletters/forecasting-futurism/default.aspx.

I included both the source and compiled forms of this program. Additionally, I included the data file for the breast cancer research. This program operates on Excel .XLSX files. To take the program through its paces, use the following steps. The following files are included with the submission.

Script File: script.xml
Training data: breast-cancer-.xlsx
Evaluation data: breast-cancer--eval.xlsx
My Sample Output: output.xlsx
My Sample Population: population.ser

The included script file will automatically use the above filenames. The training data and evaluation data both contain non-overlapping samples from the original data. This allows you to evaluate with non-trained data. The training data has considerably more rows than the evaluation. It included two sample output files. However, you will overwrite them once you complete the steps below.

Use the following steps to run the .

1. Launch the GeneticAlgorithmUtil.exe .
2. Fill in the location of the script.xml file that you would like to use. If you are running from the folder I sent, it will most likely find the included script.xml file.
3. Click **Load** to load the script.
4. Click **Generate** to generate and score an initial random population.
5. Click **Train** to train the population. The program will show the number of child genomes created and the current best score. Stop training when the score goes to near 0.01. This can take a few minutes, depending on your computer's speed. It usually takes me around 70k genomes to get to 0.01.
6. At this point you have a trained population; you can **save** it if you wish.

CONTINUED ON **PAGE 44**

7. Now let's evaluate it on data not part of the training set. Click **Evaluate**.

8. A file named Output.xlsx will be generated. You can see the predictions from the program. It should be very accurate.

When you run the "Evaluate" option you will see the program attempt to classify data that it was not trained on. It is always important to hold back some of your training data for evaluation. This ensures that the model has actually learned and not simply memorized (over fit) the training data.

RADIAL BASIS FUNCTION NETWORK DESCRIPTION

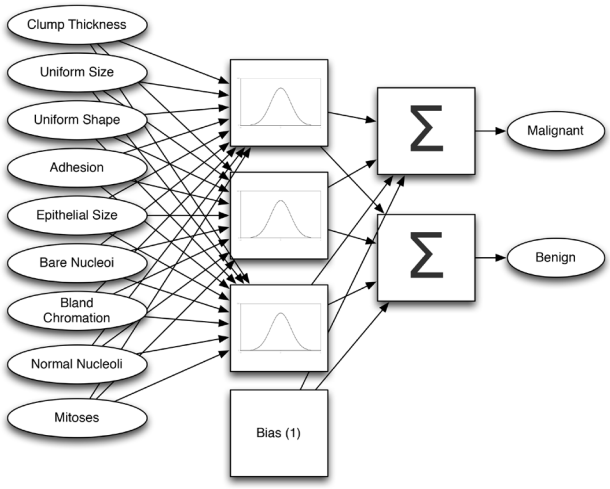
An RBF network is a statistical model that can be used for both classification and regression. It provides a weighted summation of one or more RBFs, each of which receives the weighted input attributes used to predict. The following equation summarizes an RBF network.

$$f(X) = \sum_{i=1}^{N} a_i p(\| b_i X - c_i \|)$$

Where **X** is the input vector of attributes, **c** is the vector center of the RBF, **p** is the chosen RBF (i.e., Gaussian), **a** is the vector coefficient (or weight) for each RBF, and **b** species the vector coefficient to weight the input attributes.

RBF NETWORKS CAN BE ADAPTED TO BOTH CLASSIFICATION AND REGRESSION PROBLEMS.

Graphically this can be viewed as follows.



Arrows represent all coefficients from the equation. The arrows between the input attributes and RBFs are represented by **b**. Similarly, the arrows between the RBF's and the summation are represented by **a**. You will also note that there is a bias box. This is a synthetic function that always returns 1. Because the bias function's output is constant, no inputs are required to it. The weights from the bias to the summation function vary similarly to the vector intercept in linear regression.

Because there are multiple summations, you can see that this is a classification problem. The highest summation specifies the predicted class. If this were a regression problem, there would be single output. This single output would represent the predicted output for regression.

GENETIC ALGORITHM DESCRIPTION

A genetic algorithm typically evolves a population of potential solutions to a problem. Each potential solution is typically called a genome or chromosome. Each potential solution is represented as a "DNA strand. This DNA strand is an array of

numbers. To evolve an RBF, I treat each RBF as an array of numbers. These arrays contain the following values:

- Input coefficients
- Output/summation coefficients
- RBF width scalars (same width in all dimensions)
- RBF center vectors.

RBF networks are typically trained either using gradient descent or matrix transformations. However, such approaches only adjust the coefficients. This leaves the RBF parameters to manual selection.

Using a genetic algorithm allows me to evolve all parameters to the RBF network. The only aspect that I am not evolving is the number of RBF functions to use. This must be defined in the script.xml file. This is because most genetic algorithms use a fixed DNA size. This is biologically plausible because mating only occurs through genetically similar genomes (i.e., those of the same species).

Genetic algorithms require a score function to determine the superior genomes. The scoring function for this program is very simple. The training data is used to see how accurately a given genome (RBF network) can predict a given malignant or benign status.

This genetic algorithm employs many advanced techniques from current research.

- **Fully multithreaded**. This program makes use of the C# Parallel class to ensure that all available processors and cores are fully utilized. This results in very fast training on modern core CPUs.
- **Non-epoch based**. Unlike many genetic algorithms, the population is not rebuilt each epoch. Rather, parents are chosen from the population and resulting children replace weaker genomes. This is very efficient for threading. It is also biologically plausible. We do not have clear-cut lines (epochs) of when each generation begins and ends in the real world.
- **Tournament selection**. When a parent must be chosen, for mutation or crossover, we choose a random member of the population and then try 10 more random members to get a more suited potential parent. When unfit genomes must be removed, this process is run in reverse. This is very efficient for threading, and is also biologically plausible. You don't have to outrun the fastest tiger on earth, just the tigers you randomly encounter on a given day! This also removes the need for a common genetic algorithm technique known as elitism.
- **More than two parent crossover**. Why not have more than two parents? A child can be created from several optimal parents. This is a very interesting technique that I first learned about from a *Forecasting & Futurism Newsletter* article (December 2012) by Dave Snell.[3]

EXTENDING THE PROGRAM

A genetic algorithm can be used to optimize DNA for more than just RBF networks. To facilitate other models simply create a C# class that implements the following methods via the **IGAModel** interface.

void Init(ConfigScript script, string config);

The Init method simply gives you access to the config.xml script, as well as an optimal configuration string passed to your program. For the RBF network model this is the number of RBF functions.

Genome GenerateRandomGenome(Random rnd, ConfigScript script);

The GenerateRandomGenome method is called to generate a new random genome. This is typically performed as part of the initial population generation.

double[] Compute(double[] input, Genome genome);

CONTINUED ON **PAGE 46**

DIAGNOSING BREAST TUMOR … | FROM PAGE 45

The compute method computes the genome's output based on the given input.

double Score(Genome testSubject, double[][] training, double[][] ideal);

The score method determines how fit the genome is. Optional training data can be passed as well.

Genome Mutate(Random rnd, Genome genome);

Mutate the specified genome and return a child. The original genome is not changed.

Genome[] Crossover(Random rnd, Genome[] parents);

Perform a genetic cross over, based on the parents. The children are returned. ▼

Jeff Heaton

Jeff Heaton is EHR informatics scientist at RGA Reinsurance Company in Chesterfield, Mo. He can be reached at jheaton@rgare.com.

ENDNOTES

1 Mangasarian, O.L., and W.H. Wolberg. 1990. Cancer Diagnosis via Linear Programming. SIAM News 23(5), September, pp. 1 & 18.

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

3 Snell, D. 2012. Genetic Algorithms—Useful, Fun and Easy! *Forecasting & Futurism Newsletter.* December, pp. 7–15.