COS 314 Assignment 3
Keelan Matthews – u21549967

Technical Specification

Objective

The goal of this task is to create a neural network and genetic program that can evolve decision trees, and then test their accuracy using the Wisconsin breast cancer dataset. To ensure that the results are valid, we will use the J48 decision tree algorithm from Weka and include its output in the statistical analysis.

Seed

Seed = 0

Hardware Used

The assignment was completed on an Asus Vivobook Pro 17 laptop with the following specifications:

- CPU: Intel Core i7-8550U, with a clock speed of up to 4.0 GHz
- RAM: 16GB
- Operating System: Windows 11
- Development Environment: Visual Studio Code

Data Pre-processing

The dataset was encoded using one-hot encoding to make it usable by the algorithms. Each attribute was converted into a binary vector that the algorithms could understand. To represent each attribute as a double value, a normalization technique called "One Hot Normalization" was used. This process resulted in a double value that represents a range between the number of digits in the binary vector of that attribute. The size of the resulting array of double values for each dataset entry corresponded to the number of attributes in that entry.

In cases where a value was unknown, it was set to 0.

To prevent the algorithms from becoming too focused on one outcome, the entries in the dataset were shuffled.

Next, 70% of the dataset was designated as the training set, and the remaining 30% was designated as the testing set. This allowed us to evaluate the performance of the algorithms on data that they had not seen during training.

Neural Network Configuration

The neural network used in the experiment had a few key configurations. Firstly, the input size was set to the number of attributes in a dataset entry minus one to account for the class attribute.

The network had a single hidden layer with five neurons that used the ReLU activation function. The output layer consisted of a single node due to the binary classification problem, and the sigmoid activation function was used. The sigmoid function converted any input into a range between 0 and 1, allowing the values above 0.5 to represent true and anything below as false. This also enabled the output to match the same range as the input values.

The weights were initialized using the "He Initialization" method, which multiplies the Gaussian by the square root of the variance between the input and hidden size. This technique helps prevent the gradient from vanishing or exploding during training.

The learning rate was set to 0.001 to prevent the algorithm from over-fitting or converging too early. This allowed a larger portion of the training set to be analyzed before the stopping condition was reached. The maximum iterations were set to 4000 for safety, but the algorithm often terminated at around 1000 iterations due to the tolerance in the stopping condition. The stopping condition was based on the error per epoch, and it checked if the error did not change over 10 instances, with a tolerance of 0.1. This helped prevent over-fitting by stopping the training process if the error had already converged.

Lastly, the backpropagation algorithm was utilized to update the weights during training.

Experimental setup (testing)

During the optimization process, different sizes of hidden layers and learning rates were experimented with. It was found that hidden layers that were too small did not have enough weights to accurately predict the outcome, often resulting in an accuracy of less than 40%. On the other hand, hidden layers that were too large returned low accuracy due to having too many weights for a small input amount.

Regarding the learning rate, rates that were too high caused the training process to stop early, and the network would become overfit to the training set. Learning rates that were too low did not adjust the weights enough to produce viable results either.

Initially, the testing and training sets were split 50/50. However, since the dataset was imbalanced, the network would overfit to recurrence-events. To resolve this issue, the the proportion of the training set was increased and randomized the entries in each dataset.

It was found that a lower tolerance (0.001) was not sufficient to stop the epochs from generating due to it being too small to make the values equal each other. As a result, it was opted for a larger tolerance.

Lastly, the pre-processing was initially done by inputting the one-hot encoding values straight into the network without normalizing them first. This approach required 52 input neurons, which was inefficient since it scaled the entire network up.

GP Decision Tree Configuration

The genetic program employed several key configurations for the experiment. The population size (POPULATION_SIZE) was set to 100 individuals, and the maximum number of generations (MAX_GENERATIONS) was limited to 50, in accordance with the specified requirements.

The crossover rate (CROSSOVER_RATE) was set at 0.7, indicating that there was a 70% chance for crossover to occur during reproduction. This allowed for a degree of elitism, where trees with higher fitness values had a chance to be replicated into the next generation.

In contrast, the mutation rate (MUTATION_RATE) was set to 0.0, meaning that no explicit mutation was introduced. Instead, the setLeaf(bool value) function in the Node class was responsible for randomly assigning values for the "yes" or "no" categories each time a node became a leaf. This inherent randomness provided sufficient mutation during the pruning process. Additional mutations were avoided to prevent excessively diverse populations, which could lead to poorer average accuracies.

To determine the fittest individual for reproduction, a tournament selection process was employed. The TOURNAMENT_SIZE was set to 10, where ten individuals were randomly selected, and the one with the highest fitness value was chosen. This approach struck a balance between generating variation among the offspring and favoring trees with higher fitness values for the next generation.

The maximum depth of the tree (MAX_DEPTH) was set to 5. Any nodes beyond this depth resulting from crossover were pruned. Trees shallower than a depth of 5 lacked sufficient categories to accurately represent the output, while trees deeper than this level became excessively large and time-inefficient due to the exponential growth of branches.

By employing these configurations, the genetic program aimed to strike a balance between maintaining high fitness values through elitism, generating variation through crossover and tournament selection, and controlling the size and efficiency of the trees.

Experimental setup (testing)

Different parameters for CROSSOVER_RATE and MUTATION_RATE were experimented with to strike a balance between the two in the genetic program. The goal was to find an optimal configuration that achieved the desired outcomes. As mentioned in the configuration, explicitly setting the MUTATION_RATE led to populations that were overly diverse. Conversely, a high CROSSOVER_RATE resulted in the loss of trees with high fitness values during crossover.

To address these issues, elitism was implemented to preserve trees with high fitness values, mitigating the impact of crossover. Additionally, the chance of crossover was reduced to maintain the presence of fitter trees in subsequent generations.

The size of tree depths was also subject to experimentation. Various depths were tested to identify the optimal balance. It was found that trees shallower than a certain depth lacked the necessary information to accurately represent the output. Conversely, trees deeper than a certain depth became excessively large and inefficient.

The researchers also explored different values for the TOURNAMENT_SIZE parameter. Low values, such as 2, had a low likelihood of selecting highly fit trees from a population of 100 individuals. On the other hand, high values, such as 40, caused the population to converge prematurely, as multiple copies of the same trees dominated the offspring generation.

Results

Algorithm	Accuracy	Positive			Negative		
		Precision	Recall	F-Measure	Precision	Recall	F-Measure
Neural Network	76.50%	0.757	0.949	0.842	0.307	0.727	0.432
GP Decision Tree	72.90%	0.839	0.788	0.813	0.474	0.391	0.429
J48 Decision Tree	76.60%	0.513	0.294	0.374	0.806	0.913	0.856

To analyze the three algorithms using the Friedman Test, the algorithms must first be ranked based on the average performance across all measures. This is done by averaging the ranks for accuracy and F-measure:

Algorithm	Accuracy Rank	Pos. F-measure Rank	Neg. F-measure Rank	Avg. Rank
Neural Network	2.00	1.67	2.33	1.67
GP Decision Tree	3.00	2.33	2.67	2.67
J48 Decision Tree	1.00	3.00	1.00	1.67

It can be decided that J48 Decision Tree has the highest average rank, followed by Neural Network and then GP Decision Tree. Next, the sum of the ranks for each algorithm is calculated:

Algorithm	**Sum of Ranks**
Neural Network	7.67
GP Decision Tree	10.67
J48 Decision Tree	6.67

Now, the Friedman test statistic will be calculated:

$$x^2 = \frac{12}{k * n * (n+1)} * \sum (Rj - T)^2$$

$$x^2 = \frac{12}{3 * 3 * 4} * ((7.67 - 4.5)^2 + (10.67 - 4.5)^2 + (6.67 - 4.5)^2)$$

$$x^2 = 17.609$$

Following that, the critical value from the chi-square distribution table is found using α = 0.05 and degrees of freedom (df) = k - 1 = 2. The critical value is 5.99.

Lastly, the calculated value of χ^2 is compared to the critical value. Since χ^2 (17.609) is more than the critical value (5.99), we reject the null hypothesis that the algorithms perform equally well on average. Therefore, we can conclude that there is a significant difference between the algorithms based on this data.

Analysis

Based on the significant differences between the three algorithms (Neural Network, J48 Decision Tree, and GP Decision Tree), we can identify when to use each algorithm:

Neural Network:

The Neural Network algorithm is suitable for scenarios where accurate predictions for positive class samples are important. It demonstrates relatively high precision, recall, and F-measure for the positive class. However, its performance for the negative class is comparatively lower, potentially due to dataset imbalance.

J48 Decision Tree:

The J48 Decision Tree algorithm stands out in terms of F-measure and accuracy for the negative class. It is appropriate for scenarios where accurate detection of negative instances is a priority. However, its performance for the positive class is relatively lower compared to the other algorithms due to the training approach using cross-validation.

GP Decision Tree:

The GP Decision Tree algorithm, although having lower accuracy and F-measure compared to the other algorithms, still shows competitive performance in terms of precision, recall, and F-measure for both the positive and negative classes. It is particularly useful in scenarios where transparency in the decision-making process is important due to its decision tree structure being navigable.

Conclusion

The choice of algorithm depends on the specific requirements and priorities of the problem at hand. The Neural Network algorithm is suitable for accurate predictions in the positive class, the J48 Decision Tree is effective for accurate detection of negative instances, and the GP Decision Tree provides transparency in decision-making. These factors determine which algorithm is best-suited.