Project Report - CISC 452

Gabriele Cimolino - 10009098
Jack East - 10111410
Ly Sung - 10150760

April 5, 2019

Problem & Motivation

Surgical mortality is one of the most important outcomes that is used to measure the quality of medical care. When it comes to one's survival rate, many factors are taken into consideration. With the involvement of machine learning, many have proposed methods for predicting post surgical results. We are interested in using different neural network models to predict the survival outputs of patients. We will be using thoracic surgery data set related to postoperative life expectancy in lung cancer patients to train and test each model. The reason we have chosen this problem is because we wanted a problem which has good solutions using smaller neural networks so that we could experiment with new approaches that involve creating many instances of solutions. Additionally, we are curious to see how each model does against a logistic regression and each other.

Related Work

Risk factor identification and mortality prediction in cardiac surgery using artificial neural networks - Jack

The problem addressed by this paper was to develop a method to select risk variables and predict mortality after cardiac surgery using artificial neural networks[4]. The data used in this paper was collected from 18,362patients undergoing cardiac surgery and 128 European institutions in 1995. This method made use of a multilayer perceptron with 1 hidden layer and 1 hidden node. The results of this implementation gave an operative mortality rate of 4.9%, which was very successful. When it came to building my own network, I decided to take some inspiration from this implementation based on how successful it was and use more hidden layers to increase the order of accuracy, even though it would make the network more computationally expensive.

Neural Networks as a prognostic tool of surgical risk in lung resections - Jack

The goal of this paper was to determine the surgical risk in patients undergoing pulmonary resections[1]. In this paper, 141 patients underwent lung resections and their data was collected retrospectively. The data was divided into a training set (n=113) and a testing set (n=28). A genetic algorithm was used to produce the best result with an accuracy of 79%. When it came to building my own implementation, I decided that since another group member was already using a genetic algorithm I wouldn't use their implementation, however, because they had a smaller sample size similar to our data I used a similar training/testing split to allow my implementation to have enough training data.

Research Paper: A Deep Neural Network Model using Random Forest to Extract Feature Representation For Gene Expression Data Representation - Ly

Summary

Gene expression data is found to be difficult for classification model since the number of samples is known to be less than the number of features. Cell population exhibit a large heterogeneity, but data results from different labaories are inconsistent, hence the small sample size and sparse feature set. The article wishes to use supervised random forest to learn about the sparse data and feed its representation to a deep neural network[3]. The idea is to use random forest as a feature detector to reduce the dimensionality and DNN as the final result predictor. Random forest was chosen because it takes into consideration all the given feature set rather than a single predicted probability score. DNN was chosen because it has a good evaluation mechanism.

Methods

The random forest is an ensemble of different decision trees. The tree is constructed using X, the input samples and Y, which represents the class label. After the tree is fitted, each testing input sample X_i goes through $f_i : \cdot = \cdot f(x_i; \theta)$ to calculate the binary output for the tree. Once the forest is fully formed, the output matrix will be piped to two hidden layers. The deep neural network is trained using stochastic gradient descent algorithm to minimize the cross-entropy loss function.

$$\Psi = \ln \Sigma_i = \ln(y_i \log(p_i) + (1 - y_i) \log(1 - p_i))$$

$$\text{The activation used in DNN is ReLU function: } \Sigma \text{ ReLu}(x) = \max(x, 0)$$

ReLu was chosen instead of sigmoid because it can deal with the vanishing gradient problem that is often seen during the optimization section. The article uses mini-batch training method to train a small batch of samples in each iteration. The goal is to start with a small network and iteratively build on top of it so that it has more time to learn.

Compare

ReLu was chosen instead of sigmoid because it can deal with the vanishing gradient problem that is often seen during the optimization section. The article uses mini-batch training method to train a small batch of samples in each iteration. The goal is to start with a small network and iteratively build on top of it so that it has more time to learn.

Research paper: Using a Random Forest to Inspire a Neural Network and Improving on It - Ly

Summary

Different methodologies such as convolutional neural network can increase the accuracy of implementation by reducing the dimensionality of the input parameters. The article plans on converting a traditional random forest that is already robust into a neural network[8]. The goal of the article is to construct a neural network model that is similar to a random forest in order to give a better classification and to reduce overfitting.

Method

The model first constructs a randomized decision tree in the form of a neural network. Learning process of a neural network is applied with backpropagation. In addition, it uses decision making functions along with forward and backpropagation to reduce time complexity. The architecture of the model involves three layers: input, hidden and output layer. In a traditional neural network, there is a clear distinction between input layer and hidden layer. However, this model introduces a different approach in which nodes in the hidden layer are able to receive inputs directly from the selected features as well as outputs from other nodes. The neural network is structured exactly like a binary tree where each node contains exactly two children, and the output of each node gets fed into the next immediate node (the other node remains a 0 for inactive). A number of randomly selected features are used in order to perform the split at each node, thus keeping the network inherently randomized. One important property of the tree is that, if the parent node is 0 then all its children nodes will also become 0 to denote the inactivity of the path. The output layer takes in all the outputs from all the previous nodes to make a final class prediction. As a result, it makes it a lot easier for backpropagation since the network only has to traverse back through the active path. The algorithm uses Leaky ReLu in hope to outperform tanh as the activation function.

$$\text{Output: } g(W_{11} f11 + b_{11}, \text{ where}$$

$$g(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.2x & \text{if } x < 0 \end{cases}$$

$$g'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0.2 & \text{if } x < 0 \end{cases}$$

$$\text{Signal vector / Weight vector: } s_{11}(1) = I(a_{11}(1), a_{11}(2)) s_{11}(2) = I(a_{11}(2), a_{11}(1)$$

$$\text{Output at next node: } P_{2j} = \begin{cases} g(W_{2j} * [f_{2j}, p_{11}(j)] + b_{11}) & \text{if } s_{11}(j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$s_{2j} = \begin{cases} [I(P_{2j}(1), P_{2j}(2)), I(P_2 j(2), P_{2j}(1))] & \text{if } s_{11}(j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Output layer uses softmax function to predict the final output.

Compare

The implementation of my neural decision tree stems from the inspiration of this article. Although the architecture is taken from research paper, I have changed a few things such as using sigmoid function and how the split is calculated to see if the accuracy for classification will remain the same using different methods. Using sigmoid as an activation function is used because it was simple and differential everywhere. For each node within my implementation, it uses a gini index as the cost function to determine the split point for the decision tree node.

Memetic Algorithms - Gabriele

Memetic algorithms are a variant on genetic algorithms that includes an extra phase in each generation that is meant to simulate Lamarckian evolution[2]. Where the genetic algorithm portion is a global search method, following the usual phases of crossover, mutation, evaluation, and selection, memetic algorithms make use of a local search method, that modifies individuals in a guided way, that is meant to mimic cultural evolution[2].

Algorithm 1: Genetic Algorithms

```
Create initial population;
Evaluate the fitness of each member;
while Termination criteria not met do
    while New population size is less than current
     population size do
        Select parents;
        Combine the parents' genes to produce two
         child solutions;
        Mutate the children randomly;
        Add both children to the new population;
    end
    Evaluate the fitness of each member;
end
```

Algorithm 2: Memetic Algorithms

```
Create initial population;
Evaluate the fitness of each member;
while Termination criteria not met do
    while New population size is less than current
     population size do
        Select parents;
        Combine the parents' genes to produce two
         child solutions;
        Mutate the children randomly;
        Mutate the children using a local search
         engine;
        Add both children to the new population;
    end
    Evaluate the fitness of each member;
end
```

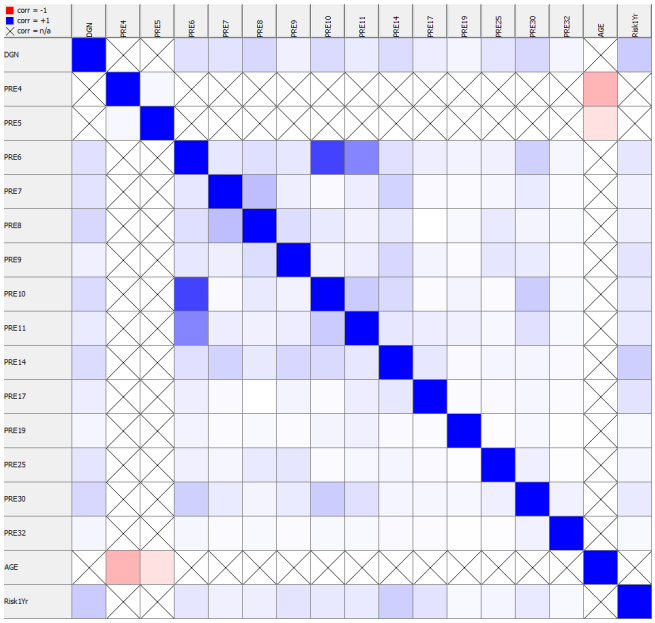
A variant on genetic algorithms, called differential evolution, with a memetic component has been employed for classification problems with medical data[5] and the related field of bioinformatics[6]. The technique was used to predict the benign or malignant status of breast cancer, the angiographic disease status of hearts, and the cellular localization cites of E.coli and yeast proteins. The authors' approach differs from the one presented here in that their local search engine involves a Random Walk with Direction Exploitation, which does not make use of neighbouring solutions to guide the search. The experimental results indicate that the local search method helps to produce more accurate solutions than differential evolution alone.

My approach differs from the one described above in two respects, aside from the use of differential evolution, in that the local search method I have devised makes use of fit population members to guide the search and selection for crossover and cultural exchange is constrained to a grid neighbourhood local to each solution. To find fit neighbours during exchange, local search is performed repeatedly as it is interleaved with evaluation of the fitness of each member. At discrete intervals each population member is given a fit solution, where each member's fitness is determined as a running tally of correct predictions made during the evaluation step, to learn from before evaluation resumes. The neighbourhood of a solution is determined with a radius value. Solutions that occupy grid positions that differ from a given member's position by at most the radius value in any direction are part of that member's neighbourhood.

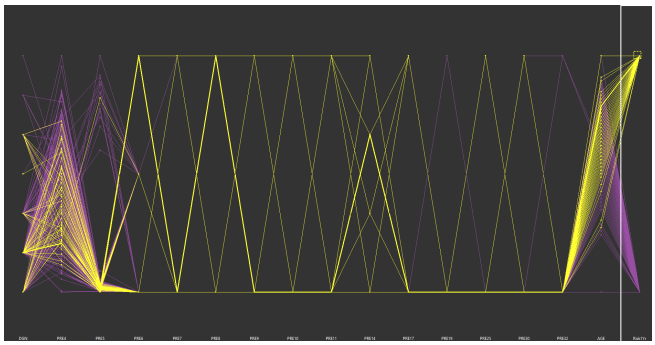
Data

The dataset used is the Thoracic Surgery Dataset which contains 16 attributes related to the medical status of lung cancer patients undergoing major lung resections. Each record includes a boolean value representing the patient's survival for one year after the surgery. This is the feature we have tried to predict, and in the data we have used roughly 85% of records describe patients who survived while the other 15% died. Because of this, the difficulty with using this data is that an 85% accurate model is easily possible if it always predicts survival. Our efforts have been focused on avoiding such simple and useless solutions, through means such as using samples of the entire dataset that contain the same number of records from each class.

In processing the data each group member has taken their own approach, however our early analysis of the data has informed the decisions we made about what attributes to use. We used techniques such as linear correlation matrices and parallel coordinates displays to determine which attributes are least discriminating.



A correlation matrix from KNIME. The blueness of an entry indicates positive correlation among the attributes, red indicates negative correlation.



A parallel coordinates display rendered using GGobi. Lines colored yellow represent records of patients that died. Purple is for patients that survived.

Using these methods we were able to determine that several attributes were poorly correlated with survival and had differing values from the majority in only a few records. These were PRE19, which encodes the occurrence of myocardial infarction within the 6 months prior to surgery, PRE25, the presence of peripheral arterial diseases, and PRE32, corresponding to asthma. It was left to the discretion of each group member whether or not to remove these attributes or others that they found unnecessary. Many of the remaining attributes corresponding to attributes that have been found to be valuable in developing surgical risk models by the authors of [7].

Multilayer Feedforward Backpropagation Network - Jack

For my implementation, I decided to use a multilayer feedforward backpropagation network. I decided to use a classification network instead of a prediction network because I believed that it would give me more accurate results as the output could only be one of two things (survival or not).

I used a learning rate of 0.05, which I obtained through experimentation. Although other learning rates produced a similar result, I found that this learning rate came to the final accuracy the quickest. I decided to leave momentum out for our current implementation and just leave it at the default value, which is 0. The reason for this is that I wanted to keep as much as possible constant to attempt to get better results and having many different parameters that could affect a result made this a little trickier.

I used a three-layer network, with an input layer, a hidden layer and an output layer. The hidden layer uses a reLu activation, and the output layer uses a SoftMax function. There are two hidden layers and 1 output layer. A reLu function was used because it results in much faster training and it also works very well with the TensorFlow framework so I wouldn't need to implement the hidden layers myself.

The tool I found most useful for the completion of this project was a combination of Keras, with a Tensorflow backend, and Python. Keras has a flexible architecture, and very good documentation and resources. It definitely matches our needs in terms of functionality, and Python is a language which functions extremely well in tangent with TensorFlow. Keras also makes it very easy to implement a network as it only requires the specification of the algorithms as well as the network. It does all the other calculations itself.

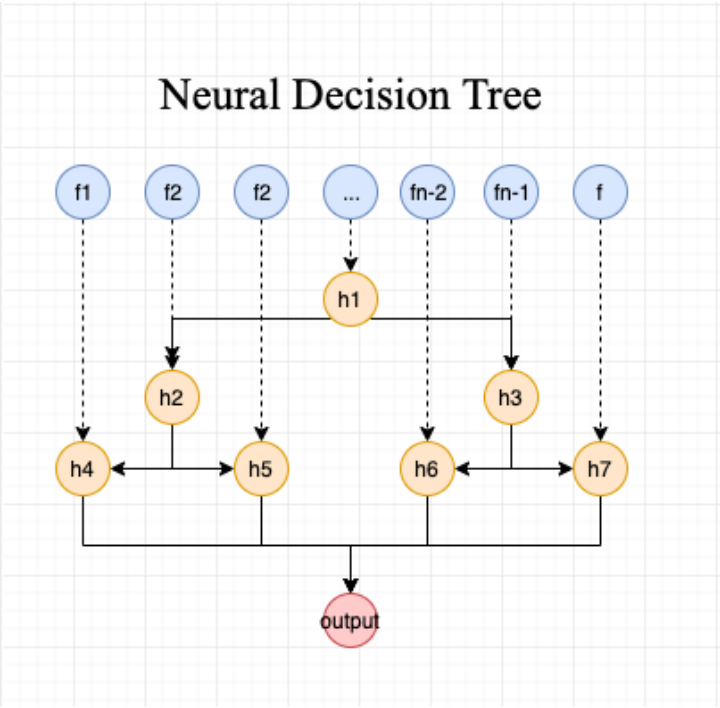
Results

Overall, the multilayer feedforward backpropagation network performed very well. A test accuracy of 74% was achieved using this implementation. I believe this is quite successful compared to the teams results and the team's validation process.

In the future, we can do some things to improve our model. The first thing we could do is try and pick better attributes from our dataset. These attributes can have all the difference when trying to predict the success of a surgery. By choosing a different set of attributes we could increase the correlation of the data and increase the accuracy of the model. Another option would be to implement a different model. Although I believe this model was a good choice to investigate it is clear based on the team's different implementations that other models could be more successful. Another possible option for improving the

accuracy would be to find a different dataset. Although our dataset had very clean data, if there had been a larger dataset, we could have used more data to train the model.

Neural Random Decision Tree - Ly



A diagram of the Neural Decision Tree architecture.

A classical decision tree is good at selecting features and determining which are the most useful features. Through multiple decision rules, the tree is able to predict the class label. However, decision trees are pruned to overfitting due to the training data used to construct the tree itself. To prevent overfitting, I decided to use random forest which is an ensemble of randomized decision trees. This method is a way to generate variation among trees and to aggregate less predictive model by taking bad solutions and making a good prediction out of it. In addition, random forest is great for classification problem while providing great accuracy of the overall prediction. Since the dataset that we are using involve only two classes: alive or not, random forest makes a very good candidate. However, one thing that random forest lacks is the ability to learn internal representations of input data. To accommodate for this problem, each decision tree is turned into a neural network with backpropagation so that it can learn from mistakes. The goal of integrating neural network into each decision tree is to make efficient self learning trees in hope to produce better results than a regular logistic regression model.

Random forest is successful with classification problems because it uses methods such as feature bagging and aggregation of all the trees. Feature bagging is a method that selects random input samples from the original data to use during the construction of a decision tree. This methodology will reduce correlation among trees to produce trees of different types. The next method is aggregating all the neural decision tree results and averaging them out in order to predict the final class result. The neural decision tree is structured exactly like a binary tree where each node has exactly two children. The connection from each node to another is represented by a weight value. Each node within the decision tree uses gini-index as a cost function to determine purity of the split. In addition, the decision node determines where the routing is to take place. The leaf node uses sigmoid function to calculate the final prediction. One thing to note is that only one active path can be activated in each input sample. The overall architecture of the network involves three layers: input layer, tree layer and output layer. The input layer uses feature bagging as described previously to select randomized set of features from the original. Each node within the input layer is mapped directly to another node in the tree layer, each with an associated weight value. The hidden layer decision nodes has exactly the same number of nodes in the input layer. Decision node is responsible for routing input either from the input layer or another decision node to its next children (left/right). Decision nodes are able to determine the routing of an incoming input by calculating the gini index to determine how perfectly separated the feature set is. At the end of the hidden layer are the leaf nodes which contain the final results. Each leaf node uses sigmoid function to determine the class label. If the predicted class of a neural decision tree is wrong, the tree will apply backpropagation to correct the weights along the active path. Lastly, the output layer takes in all the outputs from each decision tree to compute the final predicted class label for a given input sample.

Algorithm

Algorithm 3: Neural Random Decision Tree

foreach *tree* **do**
  Select **n samples from the original training data** at random
  Construct a decision tree with
    Input layer: contains randomly selected features
    Tree layer: connects each node from the input layer to another node in the tree
      Each **decision node uses sigmoid** function to determine the routing
    Output layer: contains probability distribution overa set of classes
  Calculate error and propagate it back through the active path to update weights
end
Train n neural decision trees
foreach *data point in the test data* **do**
  Get result from each decision tree
  Average the result from all trees using **majority rule**
  Return final class result
end

Data pre-processing

- Convert T/F to 1/0
- Convert categorical data to numerical value

Initial data

[DGN2,2.88,2.16,PRZ1,F,F,F,T,T,OC14,F,F,F,T,F,60,F]

After processing

[2,2.88,2.16,1,0,0,0,1,1,14,0,0,0,1,0,0]

Results

Model	Accuracy
Regression model	75%
Random forest	83%
Neural random forest	78% - best

Overall, the neural random forest performed quite well if compared to a regular regression model. However, the original random forest model seem to outperform the neural random forest. One possible reason for the lack of accuracy from the neural random forest is due to training the model on balanced data set. Since there were more survivors than non-survivors, the training data were kept to have the same number of survivors and non-survivors. As a result, the training data gets reduced very significantly. Because of this, there were less data for each neural decision tree to learn and correct its weight values. Additionally, the neural decision trees weren't able to learn from too many different input samples.

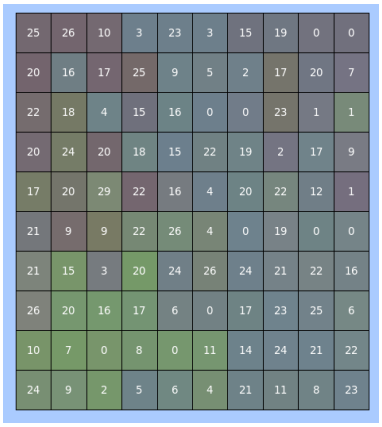
It does make sense that the neural random forest was able to outperform the regression model because it was able to maintain the methodologies of a random forest and random forest in general is known to produce high accuracy.

Identify how you can improve the results

- More data set to better train the neural network with more input samples
- Reduce / increase number of selected features to see if it can affect the overall accuracy of the implementation.
- Use different a activation function such as ReLu function as seen in the first research paper.

Memetic Algorithms - Gabriele

I set out to take a more literal approach to the cultural evolution aspect of the local search phase of memetic algorithms by having population members "talk" to each other. The idea is to assign members of each generation to cells in a grid, or multiple grids, and constrain their crossover partner selection to members of some local neighbourhood. Each population member propagates their genes to the next generation, filling their cell, but fitter solutions will be selected for crossover more frequently by those in their neighbourhoods. The local search phase will work in much the same way, but members will choose a fit neighbour to learn from. The hope is that the spatial constraints on crossover and cultural exchange will lead to clusters of similar solutions that can be thought of as distinct cultures. This is desirable in that solutions that are similar to their neighbours can be dissimilar to solutions further afield, keeping solution diversity higher than it would be otherwise. The hope is that local solution directions will allow effective memes to spread throughout the population while less successful memes will stay local to a portion of the grid, allowing the optimization method overall to explore several different directions, over the entire population, concurrently.

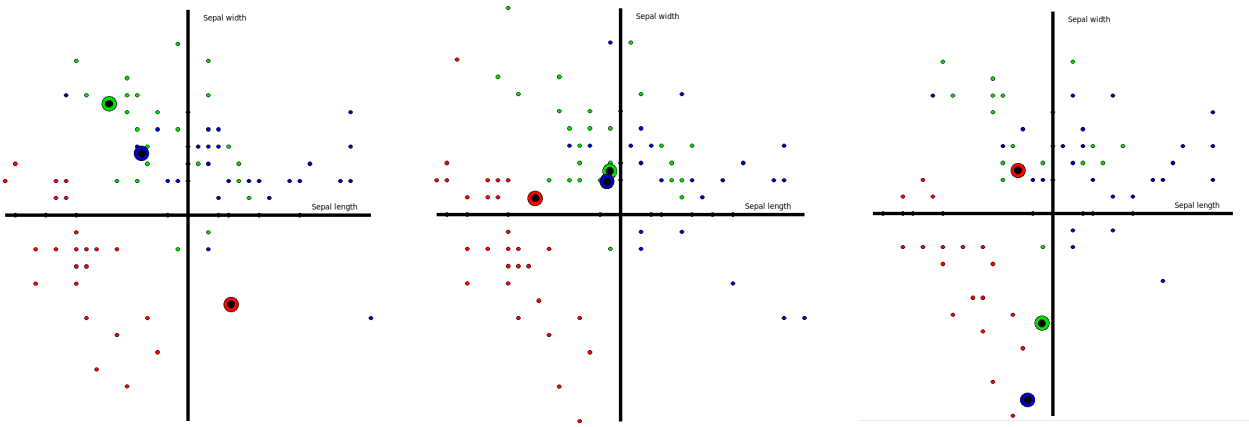


A visualization of the algorithm showing a population colored by similarity and labeled with each member's fitness.

Invertible Neural Networks

I have explored two methods of cultural exchange, both of which make use of a new neural architecture devised for this purpose. This new model is called an Invertible Neural Network(INN), it's a feed-forward network that uses the inverse of each layer's activation function and the inverse of its weight matrix to flip the network's structure, yielding an ideal input given the encoding of a target class's desired network output. When a square network structure is used, one in which all layers have the same number of neurons, when given some output an INN will return, almost exactly, the input that lead to that output. For this reason I have used square networks while testing both approaches, although it is possible to get approximations of ideal inputs when the matrix pseudoinverse is used in non-square networks.

I have written a visualization tool, called the INNViewer, to show what the inverted output of a trained INN looks like. Given a dataset, a sample size, and the desired number of layers, the INNViewer constructs a square INN of the specified structure and repeatedly trains the network on a sample of the data using backpropagation. As data points are presented, they are drawn as small dots colored to indicate their class and the INN's ideal input for each presented class is shown as a large dot with a black center. As the network becomes more accurate its ideal inputs will appear more like cluster centers, in each dimension of the data, although the relationship between the location of data points in a class and that class' ideal input becomes less obvious with deeper networks.



1 layer 2 layers 3 layers

Output from the INNViewer showing the ideal input for each class in the Iris dataset when networks of different depths are used.

Local Search Methods

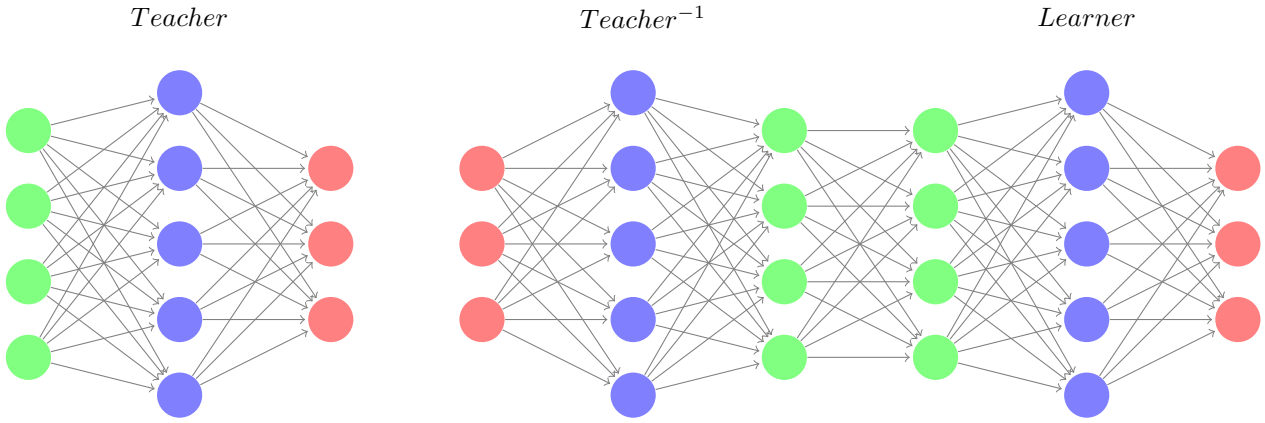
Bring Closer

The first method takes two members with identical network structures, a learner and a teacher, bringing the weights of the learner closer to the corresponding weights of the teacher with a step size determined using the amount of error found in the teacher's weights. Before the exchange, solutions are evaluated and error is assigned to the weights of each solution using the credit assignment approach of backpropagation. When two members "speak", the weights of the learner are modified such that if the teacher's corresponding weight has less error than the learner's then it is brought closer to that value. The weight adjustment for each of the learner's weights is

$$\Delta Learner's\ weight = Learning\ rate * \frac{1}{1 + Teacher's\ error} * (Teacher's\ weight - Learner's\ weight)$$

Inverted Teacher

The second method involves using INNs to train neighbouring solutions on the teacher network's ideal input for each target class using backpropagation. This trains the learner to make similar predictions as the teacher when presented with records similar to what the teacher believes records of each class ideally look like. Given a teacher and a learner, for each target class that class' desired output is fed back through the teacher and the teacher's ideal input is fed forward through the learner. The learner is then trained with backpropagation to give that class' desired output when presented with the teacher's ideal input for that class.



Cultural Exchange

Exchange is performed during the evaluation of a new population. This is necessary to use the accumulated fitness value as the final fitness value after exchange is complete. As well, this allows exchange to be permored multiple times during evaluation. In this way, the probability that a given member is selected as the teacher can be determined by its accumulated fitness and each population member's final fitness score will be influenced by the changes made during exchange.

This is done at regular intervals during evaluation. Each member has a fit teacher within their local neighbourhood chosen for it and, using one of the local search methods, it is brought closer to the teacher. This process repeats until evaluation is complete and each member has a final fitness score.

For the purpose of testing, solutions were evaluated using balanced samples, with 25 records from each class, from the entire dataset. The fitness of each member was calculated to be the total number of correct classifications of the records in the sample, for a maximum fitness value of 50.

Algorithm 3: Memetic Algorithms (Cultural Exchange)

```
Create initial population;
Evaluate the fitness of each member;
while Termination criteria not met do
    foreach Member in the population do
        Select a fit neighbour;
        Combine the parents' genes to produce a child solution;
        Mutate the child randomly;
        Add the child to the new population in the first parent's grid cell;
    end
    while Still evaluating fitness do
        Evaluate the fitness of each member on a sample of the data;
        foreach Member in the population do
            Select a fit neighbour;
            Perform cultural exchange;
        end
    end
end
```

Testing

Testing was performed using 6 configurations of the algorithm. A control test with no cultural exchange, the basic genetic algorithm approach, was done to be used for comparison with the results of the memetic tests. Testing was then performed using each of the two local search methods. These 3 configurations were each tested with and without using the grid for crossover partner selection. All 6 of these configurations were tested several times with population sizes in $\{n^2 | n \in [2, 10]\}$.

Data Pre-processing

Despite some attributes being lowly correlated with the target attribute, I decided to include all of the dataset's attributes. This is because when I removed the attributes that seemed to be less predictive of survival from the data we used to train a logistic regression model, which we used to validate our solutions' performance, it was less accurate than models trained using all of the data. Some of these attributes, such as the patient's diagnosis or their score on certain tests, were represented with discrete values that could not be sorted meaningfully. For instance, diagnosis 3 is not necessarily more or less severe than diagnosis 1. Instead of creating a new boolean attribute for each diagnosis or test score, which would require my solutions' network structure to have almost twice as many nodes per layer, or removing them from the data, which might negatively affect performance, I decided to process these attributes in the same way that I did the others.

For each attribute in the dataset I assigned each unique value an integer, for the categorical ones, and left numeric values unchanged. I then determined the range of values for each attribute, and using that range, scaled and translated each value to lie within [-1,1]. Below is an example of one record that has had its categorical values enumerated and then all of its values min-maxed.

['DGN3', 4.52, 3.36, 'PRZ1', 'F', 'F', 'F', 'F', 'T', 'OC12', 'F', 'F', 'F', 'T', 'F', 63.0 — Class: F]

[0, 4.52, 3.36, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 63.0 — Class: 0]

[-1.0, 0.2675, -0.9438, 0.0, -1.0, -1.0, -1.0, -1.0, 1.0, -0.3333, -1.0, -1.0, -1.0, -1.0, -1.0, 0.2727 — Class: 0]

Solution Structure

Due to the imprecision of the matrix pseudo-inverse used to flip the INN's when the network structure is non-square, a structure of 16 - 16 - 16 was used for all of the tests. This means that each solution had 14 more outputs than necessary, because survival is a two class prediction problem, and significantly more weights to use in crossover and exchange. I made this decision to be able to make a better comparison of the performance of each of the testing configurations used. It could be that more accurate solutions would have been found more quickly had a smaller network, with only 2 outputs, been used for the genetic and Bring Closer tests.

Termination Criteria

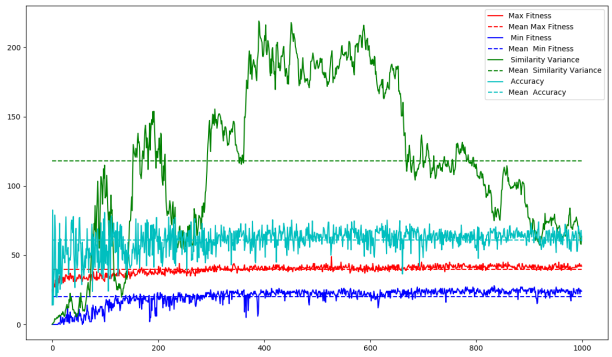
Each test was run for 1000 generations or until the variance in the mean cosine similarity of all population members fell below the threshold 0.001. The variance in the mean cosine similarity of the population is a measure of the diversity of the

current population. Terminating the test when this value drops below some threshold means that the test stops once the current population has converge to a single paradigmatic solution with only small variations between population members.

The cosine similarity of two population members, m_1 and m_2 from population P, is the dot product of their normalized weight vectors. So, $Similarity = m_1 \cdot m_2$ where $m_i = \frac{[w_0, w_1, ..., w_n]}{\|[w_0, w_1, ..., w_n]\|}$. The entire population can therefore be expressed as $P = \begin{bmatrix} m_1 \\ m_2 \\ ... \\ m_s \end{bmatrix}$ and the cosine similarity of all members with all other members, including itself, is PP^T. For each row of PP^T the expected value is calculated to give each population member's mean similarity to all other members. The mean and then variance of these values is calculated, yielding the population's variance in the mean cosine similarity of all members.

Performance Measures

For each generation 4 performance measures were tracked and recorded, they are the fitness of the fittest member, the fitness of the least fit member, the accuracy of the fittest member on all data points, and the similarity variance of the whole population. For each test these values were aggregated to make 3 new attributes from each, the maximum and minimum values as well as the mean. The performance measures of any test can be opened using the ResultViewer, included with my implementation, to generate a graph of these performance measures across all generations.



A plot of the performance measures records for an Inverted Teacher test run for 1000 generations with a population size of 100.

Results

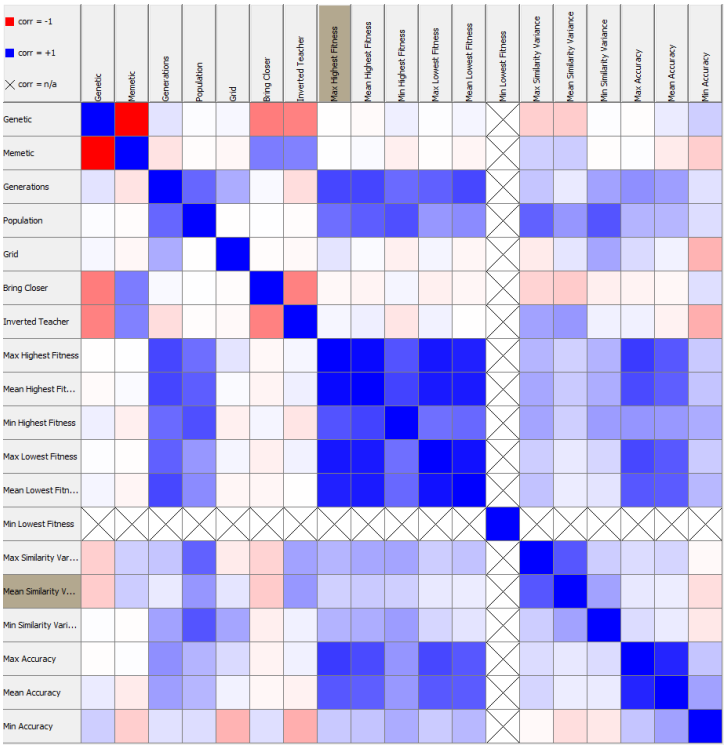


Figure 3: The correlation matrix for all of the aggregated performance measures.

The results generated from testing are somewhat difficult to interpret. I've summarized some of them, but even these have nuances that make them less indicative of performance of each approach than I would have liked. The mean accuracy column below shows the mean of the mean accuracy of the fittest member from each generation for each approach across all tests. It shows that the mean accuracy was highest when the basic genetic algorithm approach is used. Max accuracy is the highest accuracy achieved by the fittest solution from any generation for all of the tests. The approach that achieved the highest accuracy was a memetic configuration where grid selection was used and local search was performed using the Bring Closer method. Like max accuracy, the max fitness is the highest fitness achieved by any solution. Mean fitness is the mean fitness of the fittest member of each generation for all tests. Finally, mean similarity variance is a measure of population diversity, this is the average diversity of all generations for each test.

Many of these values seem odd to me. The mean accuracy was highest for the genetic approach, yet the mean fitness was highest with the Inverted Teacher search method with selection in the grid. An approach's accuracy, given all data points, and fitness, accuracy given a balanced sample, should be more closely related, yet the results indicate otherwise. The max accuracy values are less surprising given the other performance measures, and the fact that 85% accuracy is guaranteed with a strong bias towards predicting survival. For this reason, early solutions will quickly learn

to prefer survival, and therefore have higher accuracy than future generations that have learned more of the data's nuances. The highest accuracy achieved, by a Bring Closer test with roughly 86%, is lower than the highest accuracy I've seen achieved by a single solution trained on balanced samples with backpropagation. If a single solution can be trained to be almost 90% accurate then its shocking to me that none of these optimization methods were able to do as well.

Overall, it can be seen in the linear correlation matrix generated from the aggregated results from all 192 tests that accuracy was highest when no local search method was used. This means that the basic genetic algorithm approach outperformed the two memetic approaches combined. However, the results from the Inverted Teacher tests were promising since they were highly positively correlated with diversity. I believe that the greater diversity in the populations generated during the memetic Inverted Teacher tests could enable the approach to run for far more generations than any of the other approaches without converging to a single solution with only small variations between members. This is desirable because it means that new solutions are still possible even after 1000 generations, increasing the probability that a better global search direction is found.

The results obtained were a little disappointing, since I was unable to achieve a *good* solution with accuracy higher than the 85% guaranteed by always predicting survival. However, I believe that there are several ways that these results could be improved. For the genetic and Bring Closer tests, it would have been preferable to use a network structure with only two outputs. The extra 224 weights leading to the 14 extra output nodes likely interfered with the accuracy of the solutions created in these tests. As well, the weights leading to these 14 extra outputs could have been assigned a fixed value of 0 for the Inverted Teacher tests. Having weights fixed at 0 would allow the INN's to have square weight matrices and prevent the error from the unused output nodes from influencing their ideal inputs.

Testing configuration	Mean accuracy	Max accuracy	Max fitness (Out of 50)	Mean fitness	Mean similarity variance
Genetic (No grid)	**59.7676**	85.3191	48	33.6909	0.6944
Genetic (Grid)	56.6169	85.5319	47	30.946	0.8781
Bring Closer (No grid)	54.9766	**86.1702**	47	32.1769	0.527
Bring Closer (Grid)	56.6577	85.3191	46	31.8074	0.9777
Inverted Teacher (No grid)	52.4331	85.5319	48	31.0851	9.3863
Inverted Teacher (Grid)	58.4891	85.5319	**49**	**35.4302**	**19.5327**

Summary & Future Work

Group member	Accuracy
Jack	74%
Ly	78%
Gabriele	60% (Mean)

Each of us tried a different approach involving very different neural architectures, yet we were unable to find a solution which was more accurate than always predicting survival. The most promising solution was the random forest, used as a control test for comparison with the neural random decision tree, with an accuracy that is just slightly below the guaranteed accuracy of this simple solution. Going forward, if a neural architecture is applied to this problem then it may be best to leverage the strengths of other models, such as random forests, or optimization methods, like genetic algorithms. We've shown how this can be accomplished, but clearly more work is needed to yield better results.

The issue overall is that any model trained on balanced samples of the data is less accurate than always predicting survival. This is a problem for any model trained on balanced samples because future patients should survive at a similar rate as they do in the data. So, if there were a better balance between the two classes in the original data then a model may be able to do better than this if deployed in a real world setting where the distribution of patients from the two classes will resemble the distribution of records in each class in the data. Ideally, a much larger dataset with more records corresponding to patients that did not survive would be used to construct a model that beats this obvious solution.

References

[1] Hugo Esteva, Alberto Marchevsky, Tomás Núñez, Carlos Luna, and Mercedes Esteva. Neural networks as a prognostic tool of surgical risk in lung resections. *The Annals of thoracic surgery*, 73:1576–81, 06 2002. http://dx.doi.org/10.1016/S0003-4975(02)03418-5 doi:10.1016/S0003-4975(02)03418-5.

[2] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[3] Yunchuan Kong and Tianwei Yu. A deep neural network model using random forest to extract feature representation for gene expression data classification. *Scientific Reports*, 8, 12 2018. http://dx.doi.org/10.1038/s41598-018-34833-6 doi:10.1038/s41598-018-34833-6.

[4] Johan Nilsson, Mattias Ohlsson, Lars Thulin, Peter Höglund, SamerA. M. Nashef, and Johan Brandt. Risk factor identification and mortality prediction in cardiac surgery using artificial neural networks. *The Journal of thoracic and cardiovascular surgery*, 132 1:12–9, 2006.

[5] Yannis G. Petalas and Michael N. Vrahatis. Memetic algorithms for neural network training on medical data.

[6] Yiannis G. Petalas and Michael N. Vrahatis. Memetic algorithms for neural network training in bioinformatics. 2004.

[7] David M. Shahian, Eugene H. Blackstone, Fred H. Edwards, Frederick L. Grover, Gary L. Grunkemeier, David C. Naftel, Samer A.M. Nashef, William C. Nugent, and Eric D. Peterson. Cardiac surgery risk models: A position article. *The Annals of Thoracic Surgery*, 78(5):1868 – 1877, 2004.

[8] Suhang Wang, Charu Aggarwal, and Huan Liu. *Using a Random Forest to Inspire a Neural Network and Improving on It*, pages 1–9. 06 2017. http://dx.doi.org/10.1137/1.9781611974973.1 doi:10.1137/1.9781611974973.1.