CIS6005 WRIT1 – st20138532

Part 1

To begin, the downloaded data was viewed in the WEKA .arff viewer and observed for any errors. The data looked sound, so it was converted into the comma separated format for further processing. Jupyter Notebook using Python was the chosen environment and language because the many data manipulation tools that python integrates into Notebook made it an easy choice. And Python has extensive documentation to assist with this project. Although any environment and language could have been used Python, with its numpy and pandas library are simple to use and easy to understand.

Not all of the data was going to be needed for processing, upon inspection, 5 columns contained data with over 100 N/A’s. Row 13,14 and 15 exceeded 500 defunct data points, nearly over half of their whole column. While there is no hard rule on when data should be omitted these 2 columns were dropped to increase the reliability of the data.

The dataset was then normalised using min max scaling to bring every datapoint into the range of 0,1, as well as having the columns renamed to match the data description. Inputs and outputs were defined as certain columns in the data. The final stage of data preparation was to divide the dataset randomly into two groups using the ‘test\_train\_split’ method imported from sklearn. This put aside 80% of the data to be used to train the program, and 20% used to test.

The network uses supervised learning that is the most appropriate for the kind of problem the network is trying to solve, which is classification. The neural network needs input, hidden and output neurons to function. 16 input neurons were needed to go along with the 16 remaining columns of input data. 20 hidden neurons were to be used at the beginning in a single hidden layer, and 1 output neuron was needed to parse the output. The activation function that would be used is sigmoid. Since our network is classifying along a scale of 0-1, a simple binary function like step must be used. This will track our results onto a curve for us to analyse.

The network was first trained with the training data using a learning rate of 0.01 and 40000 epochs. The first results show the error going down significantly, and it can be assumed that if more epochs were used that figure would decline even further. The last printed epoch measured the error at 0.118932

Chart, histogram

Description automatically generated

Figure 1: Training data

When the neural network was tested using the test data there was an significant decline in the amount of errors, the error rate being 0.294679. Demonstrating that the neural network has performed the task at hand.

The first accuracy measure used was a confusion matrix, which was used to determine how many correct predictions the model had made.

Chart

Description automatically generated

Figure 2: Confusion Matrix

Seaborn was used to illustrate this confusion matrix. It shows how many true and false positives and negatives that the model scored. In this case, the model scored 86 true positive, and 78 true negatives. Combining the false positives and negatives the model made 67 mistakes with classifying the test data. The model also scored an accuracy of 0.709 (rounding to 3 figures) and a F1-score of 0.719. This is a good result for a model that has only been trained once with 1 hidden layer, but it can be improved in many ways. The first way to seek out an improved accuracy is to experiment with the number of hidden neurons in the model. First, the model will be re-run using less hidden neurons, 12 instead of 20.

Chart, histogram

Description automatically generated

Figure 3: 12 neuron training

The first observation that can be made just from the training data is that the error rate starts higher ends higher. The error rate only gets as low as 0.173841 by epoch 39001(the last reported epoch/error result), whereas at that same epoch the initial test had an error rate larger than this, which is a noticeable difference and already demonstrates that this network classifier is performing worse with less neurons. Although the difference is not too substantial, the accuracy measurements show the accuracy fell to 0.683, a significant drop in quality.

Chart

Description automatically generated

Figure 4: 12 neuron CM

This model performed overall worse than the original network. Here it can be seen that the 12 neuron model made more mistakes, with a combined 73 false positive and negatives. 6 more than the original. It can be inferred that with less neurons the model is making more mistakes in its testing. Although notably, this model did score more true positives than its predecessors, but this is likely down to the small number of epochs used and the random nature of weights. Sometimes a model may perform better in one specific way, but it does not indicate an overall stronger model.

Chart, line chart, histogram

Description automatically generated

Figure 5: 30 neuron training

Straightaway we can see that the error rate drops to the lowest it has been, dropping to 0.051205. This suggests that the more hidden neurons are used in the model, the lower the error rate becomes over time. This is because the more neurons present, the more calculations that can be done, the finer the changing in numbers, the model can do more in less time.

Chart

Description automatically generated

Figure 6: 30 neuron CM

Interestingly, this model actually performed worse than the 12 neuron model in its accuracy measure. The accuracy dropped to 0.636, which while not being a significant drop does indicate that the number of neurons, while lowering the error rate in training, does not improve accuracy in testing. This is perhaps because the model does not have enough epochs or batches being used, and that if more training was allowed the higher neuron model would exceed the accuracy of its predecessors.

There are other measures that could be undertaken to improve the network. Pruning the lowest weights to remove the least influential ones could trim the network size to improve performance. There could also be a change to the learning rules used. This model used the delta learning rule, which was developed by Widrow and Hoff, and is the most common learning rule used today. This model focuses on weight changes based on errors, and usually focuses on one output, while directly measuring error rate.

**Part 2**

To construct a new neural network the libraries of scikitlearn were used. This library is built on top of numpy, and features lots of machine learning support such as decision trees, random forest and utilities for pre and post data processing. The problem to solve here is still a classic classification problem, meaning a logistic regression model is best suited. The discrete output produced by this mode is the best suited for the 0,1 outcome style of the data. The default settings for the logistic regression method were used. Meaning the lbfgs algorithm solver, which is a popular and reliable algorithm for this task. The delta

To get a fair comparison with the homemade network, all of the hyperparameters and rules will stay the same for the first test. This means the same number of features, and the same split in data testing and training sets.

Graphical user interface, text, application

Description automatically generated

The mode achieved a modest accuracy of 0.67, almost on par with the homemade network.

Chart

Description automatically generated

Figure 7: LogReg CM

Text

Description automatically generated

The homemade model achieved an F-score of 0.719, whilst this logistical regression model scored a little lower, similar to accuracy. It also did worse by scoring more false positive and negatives than the original model.

AUC measuring is a useful classifier for understanding the difference in performance between models. We will use it to judge the changes that are about to be made to the model. Even though the model uses fairly balanced data, accuracy alone is not enough to understand the performance of the model. The AUC measure tells us the model sensitivity and specificity, which accuracy does not. AUC uses the true and false positive rate of the model across various cut-off thresholds, accuracy only measures one outcome, this measure uses more depth.

Chart, line chart, scatter chart

Description automatically generated

Figure 8: LogReg AUC

The first tuning that was done was changing some of the rules the logistic regression tool was using. The class and solver specifically. The class weight was changed to ‘balanced’ which “uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data” (scikitLearn, n.d.). The effect of this is that weights are kept roughly proportional to the number of times they are applied, this helps with bias and not letting any particular weight measure skew the results by being referenced too often. The second change made was to adjust the kind of algorithm used by the logistical regression method. The default is ‘lbfgs’, which was changed to the ‘saga’ algorithm. This algorithm handles multinomial loss in multiclass problems. It also “supports non-strongly convex problems directly and is adaptive to any inherent strong convexity of the problem.” (SAGA, 2023)

Chart, treemap chart

Description automatically generated

Figure 9: Tweaked LogReg CM

Chart, line chart

Description automatically generated

Figure 10: Tweaked LogReg AUC

The accuracy was improved to 0.692. Changing the class weight

The second kind of tuning that could be done is to change the hyperparameters used. Reverting the changes just made, now instead what will change is the number of parameters used, as well as the size of the training batch. The features “Pre-screen”, “quality” and “AM/FM” were removed. On top of this, the training size was increased to be 90% of the data, rather than 80%

Chart

Description automatically generated

Figure 11: Altered LogReg CM

Chart, line chart

Description automatically generated

Figure 12: Altered LogReg AUC

Once again we see the accuracy has actually improved. Now it sits at 0.689. This shows that reducing the number of input features can have a big impact on the accuracy of a model. So does increasing the training size. This makes sense as by increasing the amount of data the network has to train with, the results are more accurate. The drawback to this however is that now there is not a huge amount of testing data to use to test.

There are other kinds of machine learning models that can be used to test different classifier architecture. Decision trees are a common kind of classifier for simpler problems but let’s see what happens if we use it with out dataset.

Graphical user interface

Description automatically generated with medium confidence

Figure 13:Decision Tree CM

This decision tree model actually performed far worse with an Accuracy of only 0.585. If this decision tree model was being made from scratch and not from a library, selecting the attribute with the most information gain would be a good start for improving accuracy, as it is unknown which attributes the library is choosing to use.

Neural networks already take inspiration from biological processes, so genetic algorithms are an obvious next step. Genetic algorithms ‘evolve’ their data through multiple generations. Similar to how weights are randomised and then elaborated upon, this technique uses sets of random solutions which it then treats to get a best result.

Solutions that are useful are duplicated, while those that are not useful are deleted, others have small changes made to them until a best solution is found. If one wanted to improve the artificial neural network , a genetic algorithm would eventually discover the best possible solutions through trial and error. The accuracy, F-score, and other matrix related metrics, should all be improved through the use of a genetic algorithm.

Diagram

Description automatically generated

Populations are created, evaluated, mutated and then evaluated again until a solution is found.

In order to get the network into the genetic algorithm there is a process of encoding, where each chromosome will have a series of ‘genes’. The genes will represent the weights between layers, forward and back. Each chromosome can have as many genes as it needs to complete the task, unlike actual chromosomes.

If a kind of genetic algorithm was to be used here, the NEAT algorithm (T.Kearney, 2016) would have been used, one of the largest developments in neuroevolutionary, this method tries to solve many problems of current genetic algorithms. In this method, new genes are assigned unique numbers, and current genes all have historical markers that can track the history of a gene. For crossover, where we combine pieces of solutions that have proved useful, genes in each genome are sorted depending on their ancestry, this makes finding trends and data easier without expensive topological analysis. Artificial synapsis are used to counteract the negative side effects caused by the competing conventions problem, where two genomes represent the same solution, but they are utilising different encodings.

The neural networks are converted into vectors for processing, and parameters like mutation percent and solutions per population are set. Every iteration, some data will be mutated. Genetic algorithms help optimise the neural network through evolution. ‘Genes’ are crossed over at random and mutated by a set amount and then a specific filter is used to replace old genes and implement a new one, insuring the most fit solutions survive.

# References

*SAGA*. (2023). Retrieved from paperswithcode: https://paperswithcode.com/method/saga

scikitLearn. (n.d.). *sklearn.linear\_model.LogisticRegression*. Retrieved from scikitLearn: https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html

T.Kearney, W. (2016). Using genetic algorithms to evolve artifical neural networks. *Colby College*.