A Comparison of Neural Network Algorithms for the Classification of Unbalanced Bank Marketing Data

Kieron Ellis

13560567

Abstract

Multilayer Perceptron Neural Network (MLP) and Convolutional Neural Network (CNN) algorithms are optimised and compared for the Classification of unbalanced Bank Marketing Data using Python's Sci-kit Learn (Sklearn) and Keras libraries. Three Classification metrics: Accuracy, Sensitivity and Specificity are used to evaluate results, creating a multi-objective optimisation problem.

A positive relationship between the complexity of MLP Hidden Layer Size and Training Accuracy is found but at the cost of Testing Accuracy. Larger Epochs and smaller Batch Sizes result in better predictive scores for CNNs.

Overall, Sklearn's optimised MLP had the highest Accuracy, Keras' MLP the highest Sensitivity and Keras' CNN had the highest Specificity.

1 Introduction

Artificial Neural Networks (NNs) are a set of Natural Computing (NC) algorithms which can be used for Classification, Clustering and Predicting (Brabazon et al., 2015). I chose to examine these algorithms in this paper because NNs have become increasingly popular in recent times, especially Deep NNs (NNs with many hidden layers) due to the increasing computational capabilities of modern computers (Orr and Müller, 2003).

I will compare and optimise MLPs and CNNs for the Classification of an unbalanced Bank Marketing Dataset. A selection of parameters for both algorithms will be optimised to maximise 3 Classification metrics: Test Data Accuracy, Sensitivity and Specificity. Where T=True, F=False, P=Positive and N=Negative:

- Accuracy = (TP+TN)/(TN+TP+FN+FP)
- Sensitivity = TP/(TP+FN)
- Specificity = TN/(TN+FP)

The remainder of the paper is set out as follows: Section 2 details the theory of the algorithms used, Section 3 looks at the application, Section 4 covers the experiments, Section 5 is for discussion, Section 6 gives conclusions and Section 7 lists references used.

2 Algorithms

NC is a family of three computing methods which either: distil the essence of natural phenomena and systems for creating nature inspired algorithms; use natural materials for computation; or simulate natural phenomena with computers for knowledge discovery purposes (Brabazon et al., 2015). This paper will focus on examining the first branch, i.e. NC algorithms which take inspiration from a multitude of biological, physical and chemical systems (Brabazon et al., 2015). This paper will look at biologically inspired Neurocomputing algorithms, specifically NNs, which draw inspiration from how the human brain works (Brabazon et al., 2015).

The human brain is made up of billions of neurons which are networked together by synapses between them. Each neuron is joined directly to thousands of others It "is constantly receiving electrical signals from them" and when the total incoming signal to a neuron reaches a certain value, "the neuron fires and produces an outgoing signal […] to other neurons" (Brabazon, 2015, p.222).

A NN is a weighted directed graph of perceptrons (artificial neurons) which can be formulated into models to be used for supervised and unsupervised learning (Brabazon, 2015), however, in this paper only supervised learning will be examined. In Classification, the NN iteratively tries to fit the Training Set by finding a function which minimises an error measure for linking inputs to outputs, with the goal of predicting outcomes for inputs not in the Training Set.

A MLP is the most commonly used NN (Brabazon, 2015). It contains an input layer, x, and an output layer, $\hat{y}$ (an approximation to the true y), both with the same number of neurons as the dimensions of the data being modelled. In-between the input and output layers can be many hidden layers with multiple hidden neurons in each. These hidden layers "are not directly visible or modifiable from outside the NN" (Brabazon, 2015, p.225). Neurons in one layer of the MLP are joined to neurons in the adjacent layers by directed edges called Synapses. These multiply their input value by a specific weight and output the result to the neurons they are directed towards. Neurons (excluding those in the input layer) take these outputs, sum them and apply and Activation Function (e.g. sigmoidal or Rectified Linear Unit (ReLU)) which is used by the MLP to model complex non-linear patterns that a simpler model might overlook.

MLPs are most commonly trained using Back Propagation, with some form of optimisation method like gradient descent (Brabazon et al., 2015). Back Propagation is a supervised learning method where the weights applied at the synapses of the NN are repeatedly changed to minimise an overall predictive error so that the optimal weights for the model can be found (Brabazon et al., 2015).

A CNN is a NN whose structure is inspired by the connectivity pattern of the mammalian visual cortex (Ciresan, 2011). Unlike in a MLP where all the perceptrons in one layer are fully connected to the perceptrons in the next, in the Convolutional Layer of a CNN, a neuron can be connected to just a small region of neurons in the previous layer. This local connectivity between perceptrons in neighboring layers allows CNNs to make use of spatially correlated and temporal inputs (Wallach, 2015).

A Pooling Layer is commonly used immediately after a Convolutional Layer to reduce overfitting by reducing parameters and spatial size (Krizhevsky, 2012) and provide a form translation invariance (Orr and Müller, 2003). A ReLU layer applies a non-saturating activation function, $f(x)=\max(0,x)$, that does not require input normalization (Krizhevsky, 2012). ReLU layers increase the non-linear properties of the network and allow for faster training times than traditional saturating neuron activation functions e.g. tanh (Krizhevsky, 2012). A fully connected (Dense) CNN layer works like the fully connected layer of a MLP.

CNNs are used in this paper because they can exploit the temporal and spatial structure of the data (Wallach, 2015). Kim (2015, p.315) also used a CNN on this same dataset to leverage the "local and hierarchical properties" of the data.

3 Application

The dataset used in this experiment comes from a Portuguese Bank which ran a direct marketing campaign from 2008 until 2013, selling long-term deposits to its customers. The Bank's agents would either be successful or not at selling a to a customer and the result of each interaction is stored as a binary (successful or unsuccessful) variable. The dataset contains 45,211 customer interactions and is unbalanced, with approximately 12% of its instances being successful. The data was collected and publicly donated by Moro et al. (2014).

For every record in the dataset there exists a set of explanatory variables (bank account details, demographics, interaction information, social and economic indicators) and outcome of each interaction, the target variable for this experiment. The original dataset included 150 features, but Moro et al. (2014) used business and domain knowledge, and an automatic selection approach based on an adapted forward selection method to reduce this to 17 features: 6 categorical, 4 binary and 7 numerical variables.

4 Experiments

4.1 Setup

Python's Sklearn and Keras libraries were used for their MLP and CNN algorithms and a random seed of 1 was used for all experiments.

Categorical variables were transformed using one-hot encoding which creates a new binary column for all but one class and all numerical variables were scaled between 0 and 1. The dataset was split into a Training Set, used to train the NNs, and the Testing set, used to test the predictive power of the Models. To optimise results, 3 metrics were used: Accuracy, Sensitivity and Specificity.

The `hidden_layer_sizes`, `solver` and `alpha` parameters of Sklearn's MLPClassifier were tested, in that order, by first optimising one parameter before optimising the next. `hidden_layer_sizes` (the number of layers and neurons in each layer to be used) was tested by first optimising the neurons in the first layer before moving onto the next layer, etc. For `Solver` (the solver to be used for weight optimisation) all 3 solvers were tested: "adam", a stochastic gradient-based optimiser; "lbfgs", an optimiser in the family of quasi-Newton methods and "sgd", a stochastic gradient descent optimiser. For `alpha` (the regularisation parameter) a range of values from *1e-6 to 1e-0* were tested.

Keras' `epochs` and `batch_size` parameters were tested. `Epochs` is the number of epochs to be used to train the model, where an epoch is a full pass over all the Training Data. `batch_size` defines number of samples per gradient update to be propagated through the network. Using Keras, NNs were first tested with only Dense Layers, then a Convolutional Layer and Pooling Layer was added before the Dense Layers.

4.2 Results

4.2.1 Scikit-learn (MLP)

Figure 1. *Neurons in One Hidden Layer*

As seen in Figure 1, when the neurons in the first hidden layer increases, so too does the model's accuracy for the Training Set in an almost linear fashion. Unfortunately, this is not reflected in the model's Test Accuracy, which stays between 90% and 91%. The model's Test Specificity decreases slightly from 97% to 95% as the number of neurons increases. The Test Sensitivity however does seem to increase in a rather wild non-linear fashion as the number of neurons in the first hidden layer increases.

Figure 2 shows the average Test Accuracy, Sensitivity and Specificity after 5 experiments for three 1 layer MLPs: with 10, 50 and 85 neurons. As can be seen, the MLP with 50 neurons had the best Accuracy and Sensitivity, but was out performed by the 10 neuron MLP in terms of Specificity. I judged 50 Neurons to be the optimal number in the first hidden layer and began experimenting with the second hidden layer.

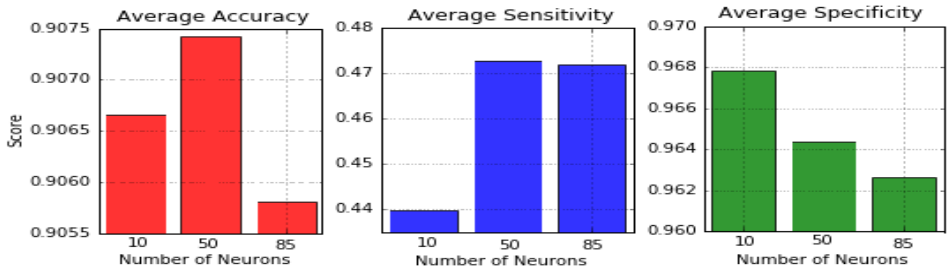

Figure 2. *Average Scores after 5 experiments*

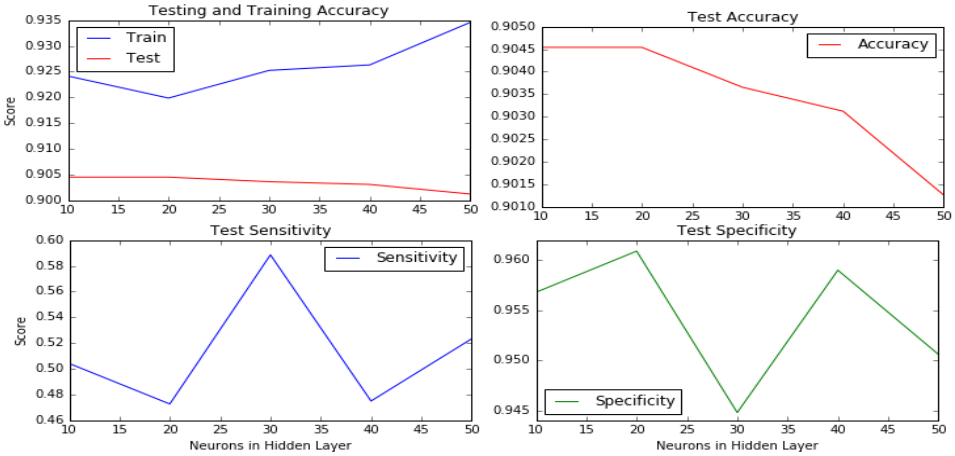

Figure 3. *Average Scores for the Second Hidden Layer after 6 experiments*

As seen in Figure 3, more neurons in the second hidden layer results in a higher Training Accuracy (just like with the first hidden layer), but a lower Testing Accuracy. 30 neurons in the second hidden layer has the lowest Specificity of the sizes tested, but the highest Sensitivity. Because of this I judged 30 neurons to be the optimal number of neurons for the second hidden layer because although the Accuracy drops by 0.1% when the number of neurons increases from 10 to 30, the Sensitivity increases by 8%.

50 and 30 neurons were used in the first and second hidden layers to test the third and fourth hidden layers. As seen in Figure 4, more neurons in the third hidden layer had little effect on Accuracy and Specificity, however Sensitivity was affected. I decided to use 15 neurons in the third layer to test the Fourth hidden layer. In the Fourth hidden layer, again Accuracy and Specificity were mostly unaffected, but for Sensitivity, Four was the optimum number of neurons, with a score of 60%.

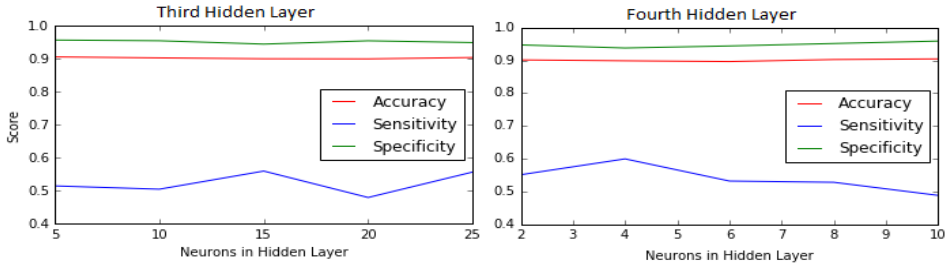

Figure 4. Average Scores in Third and Fourth Hidden Layers after 6 experiments

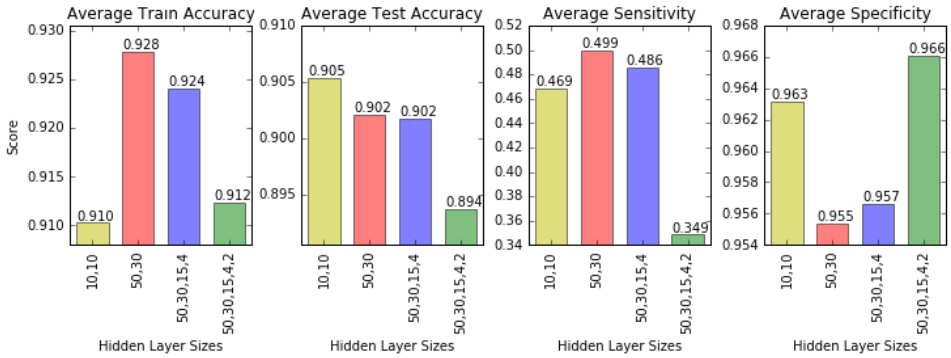

Figure 5. Scores after 6 experiments for various Hidden Layer Sizes

Figure 5 shows average scores across multiple Hidden Layer Sizes for two different Train Test Splits. As can be seen, the Deep MLP in green had the lowest Accuracy and Sensitivity but highest Specificity whereas MLPs with fewer layers had higher Accuracy and Sensitivity scores.

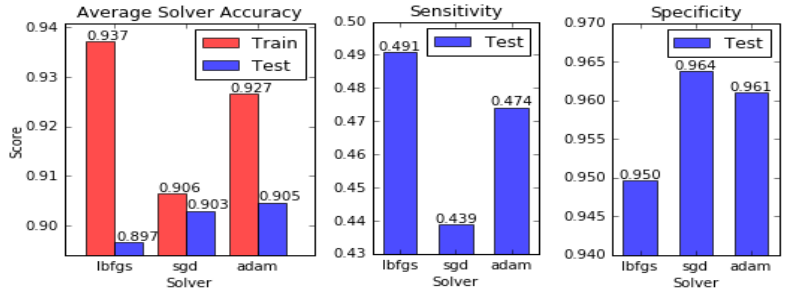

Figure 6. Average Solver Scores After 6 Experiments

As seen in Figure 6, "adam" has the highest Test Accuracy and second best Sensitivity and Specificity. "lbfgs" has a large gap between its high Training

Accuracy and low Testing Accuracy. “Sgd” performs the poorest in Sensitivity but has the highest Specificity and middling Accuracy.

The Alpha parameter was also tested with the values *1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e-0* each used in 6 experiments (Hidden Layer Size (50,30) and the “adam” solver were used). *0.1* had the highest Testing Accuracy (90.6%) and Sensitivity (50.2%), but lowest Specificity (95.9%) and vice versa for *1.0* (Accuracy 90.1%, Sensitivity 31.2% and Specificity 97.8%).

4.2.2 Keras (MLP and CNN)

Keras was firstly used with 3 Dense Layers. A multitude of Epochs between 5 and 25 and Batch Sizes between 10 and 125 had little impact on the MLP’s accuracy. However, Batch Sizes in this range did have a negative relationship with Sensitivity, which decreased by 12% as the batch size increased from 10 and 125.

Increasing the number of Epochs to 100 and using a Batch Size of 10 resulted in an Accuracy of 90.48%, Sensitivity of 58.85% and Specificity of 94.6%, which is a .29% increase in Accuracy, an 18.31% increase in Sensitivity and a 2% decrease in Specificity, compared to when 5 Epochs and a Batch Size of 125 was used. However, this comes with a cost of computation time, which drastically increased due to the increased Epochs and larger Batch Size.

When a Convolutional layer and then a Pooling layer were added to make a CNN with 5 Epochs and batch size 10, the Test Accuracy of the model fell to 89.55%, the Sensitivity was just 31.57% and its Specificity 97.12%. Using 100 Epochs and Batch Size 10 resulted in a Test Accuracy of 89.62%, Sensitivity of 53.03% and Specificity of 94.4%.

5 Discussion (Analysis)

There is a tradeoff between Sensitivity and Specificity as the number of neurons in the first hidden layer of the MLP increases. More neurons mean a lower Test Specificity score, but a higher Test Sensitivity score. The model’s overall Accuracy however remained mostly the same regardless of the number of neurons used. This was also true for the other layers tested. The more complex MLPs with more neurons and layers resulted in a higher training Accuracy, however this was not matched by the equivalent Testing Accuracy scores. As the MLP increased in complexity so too did its tendency to overfit the Training Data. It learned more complex relationships between variables in the Training

Set however these relationships were not present in the previously unseen Testing Set, resulting in a lack of generalization capability for the model and a lower Test Accuracy score.

There was a trade-off between increasing the Sensitivity of the model and its Accuracy. Generally, this meant trading a small amount of Accuracy for a larger amount of Sensitivity, and so parameters which resulted in higher Sensitivity were preferred.

Using Keras, the larger the number of Epochs used and the smaller the Batch Size used, the higher the Accuracy, Sensitivity and Specificity of the model. The addition of a Convolutional Layer to the model increased its Training Accuracy by 1.17% but reduced its Testing Accuracy by 0.86%, its Sensitivity by 5.82% and its Specificity by .2% suggesting that the model slightly overfit the training data more after the introduction of a Convolutional Layer. Overall, the effect of the model's Epoch and Batch Size parameters had more impact on the model's predictive power than the addition of a Convolutional Layer.

With optimised parameters Sklearn's MLP was outperformed by Keras' MLP and CNN in terms of Sensitivity, by 8.65% and 2.83% respectively. However, it had a higher Accuracy than Keras' MLP (0.12%) and Keras' CNN (0.98%) and higher Specificity than both at 1.3% and 1.5% respectively.

6 Conclusions

Optimising the parameters of the NNs studied generally resulted in a trade-off between increasing Accuracy, Sensitivity and Specificity. This type of multiple objective optimisation with conflicting objectives means that there are no perfect solutions, however, the unbalanced nature of the dataset meant that increasing Sensitivity usually resulted in only smaller decreases in Accuracy or Specificity and thus Sensitivity was generally preferred.

Deeper MLPs overfitted the Training Set, had a larger generalisation error and had more trouble classifying the smaller class in the unbalanced data whereas simpler MLPs had worse Training Accuracy but better generalisation.

Overall, Sklearn's optimised MLP had the highest Accuracy, Keras' MLP the highest Sensitivity and its CNN had the highest Specificity.

For future work, an Evolutionary Algorithm could be used to find the NNs parameters, this could result in interesting parameter choices, better results and could be much faster than manual coding. Strategies to overcome the unbalanced nature of the dataset could also be explored, such as rebalancing the training data, resampling or cross validation.

7 References

Brabazon, A., O'Neill, M. and McGarraghy, S., 2015. *Natural computing algorithms*. Berlin: Springer.

Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M. and Schmidhuber, J., 2011, June. Flexible, high performance convolutional neural networks for image classification. In Twenty-Second International Joint Conference on Artificial Intelligence.

Kim, K.H., Lee, C.S., Jo, S.M. and Cho, S.B., 2015, November. Predicting the success of bank telemarketing using deep convolutional neural network. In Soft Computing and Pattern Recognition (SoCPaR), 2015 7th International Conference of (pp. 314-317). IEEE.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

Moro, S., Cortez, P. and Rita, P., 2014. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, *62*, pp.22-31.

Orr, G.B. and Müller, K.R. eds., 2003. Neural networks: tricks of the trade. Springer.

Suryani, D., Convolutional Neural Network. [Online]. Published 27 February 2017. Bina Nusantara University School of Computer Science. Accessed 21/06/17. Available from: http://socs.binus.ac.id/2017/02/27/convolutional-neural-network/

Wallach, I., Dzamba, M. and Heifets, A., 2015. AtomNet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery. arXiv preprint arXiv:1510.02855.