Performance Analysis of Machine Learning Methods on Continuous and Binary Datasets

Bhargav Joshi[1], Armin Khayyer[1] and Ye Wang [1]

***Abstract*— In this paper, different machine learning methods are briefly explained and the results of their performance evaluation over continuous and a binary data sets are discussed. When a general regression network, radial basis functions neural network, SVM and feed-forward neural network were subjected to be trained using almost 1000 data points in the training data set of the Schaffer's F6 function, they showed different accuracy, precision and F1 score. These scores were improved by changing and fine-tuning the hyper-parameters of the methods.**

I. INTRODUCTION

Machine learning methods have been used for classification and regression problems. These approaches typically operate by storing each training instance in some form and then comparing new instances to those stored patterns [1]. In this paper, four major machine learning methods were subjected to train to output Schaffer's F6 function and in doing so improve the accuracy of the model by varying different training parameters. The remaining of this paper is structured as follows: Section II explains briefly the methodologies used in this paper, Section III explains the data sets and the used programming language, Section IV shows the achieved results of the mentioned tasks, Section V presents our conclusions, and finally Section VI explains the approach we took to find the population size which was used to generate the data set.

II. METHODOLOGY

A. General Regression Networks (GRNNs)

The GRNN is an instance-based method that provides estimates of continuous variables related in a linear or nonlinear way [2]. GRNN makes a single pass through a set of training instances and maps each instance t_i to a neuron in the network [1]. Each neuron has a weight equal to the desired output, d_i, of the instance. The predicted output for a given vector x is a distance-weighted average of the desired outputs of all neurons [2].

$$f(x) = \frac{\sum_{i=1}^{N} H(\overrightarrow{x}, \overrightarrow{t_i}) d_i}{\sum_{i=1}^{N} H(\overrightarrow{x}, \overrightarrow{t_i})} \tag{1}$$

where N is the number of training instances and $H(\cdot)$ is the hidden function. The hidden function for a GRNN compares a given input vector to a particular training instance and returns a value in the range [0, 1] indicating the degree of similarity between the two vectors. A common hidden function used for GRNNs is the Gaussian Function [1], which is defined as follows:

$$H(\overrightarrow{x}, \overrightarrow{y}) = e^{\frac{-||\overrightarrow{x}-\overrightarrow{y}||^2}{2\sigma^2}} \tag{2}$$

Here $||.||$ represents vector magnitude and σ represents the standard deviation. The GRNN may either use the same σ value for all neurons or a different σ for each neuron [1]. The following sub-sections explain the procedure of finding the best σ value for GRNN.

1) Evolve σ using a Steady-State Genetic Algorithm (SSGA): In order to find the best σ value, A SSGA algorithm has been utilized to evolve the σ value. A real-coded chromosome with the lenght of one is used for representing the σ values. Also the accuracy measure achieved from fitting GRNN for each σ value is used as fitness value of that specific individual (σ). At each evolutionary cycle, two parents are chosen using Binary tournament selection techniques and an offspring is created using the following crossover Operator [1].

$$child_{id} = \mu_d + \sigma_d N(0,1) \tag{3}$$

The newly created offspring replaces the worst fit individual from the population. Once the kid has been produced, its fitness over the evaluation set is calculated if its fitness value is higher than the best individual so far, it will be tested over validation set. If the validation performance exceeded the previous best validation performer, it replaced the old best validation individual. Otherwise, the previous best validation performer was stored. At the end of the run, we tested the best validation set individual on the test set and reported the performance over the test set.

2) Evolve both σ and whether a training instances is within the GRNN similar to EGRNN-I: This approach is similar to the previous one, except for each candidate solution, a GRNN was created using the training vectors specified by the masks in addition to the σ value. The first place of the chromosome of each individual is a real-coded value representing the σ value, the remaining places of the chromosome which are binary-coded are a boolean values that represents if an training instance is used in GRNN. A SSGA approach has been used for evolutionary cycles. at each cycle two parents are chosen using binary tournament selection, a Single Point Crossover(SPX) operator has been used for the boolean part of the chromosome, and finally formula 3 is used as crossover operator for the real-coded part of the chromosome which corresponds to the σ value. Then this GRNN was trained on those elements of the training set and

[1]Department of Computer Science and Software Engineering Auburn University Auburn, USA

tested on the evaluation set. Its performance was measured in terms of the prediction accuracy over the evaluation set, as well as the number of training instances that were specified by the mask. These values were used in the calculation of the fitness of the candidate solution. A criterion-based approach for the fitness assignment is used. Two candidate solutions x and y were compared first according to prediction accuracy on the test set, taking the larger of the two as the "winner". If the two candidate solutions had equal prediction accuracy, they were then compared according to the number of instance vectors used, taking the smaller of the two as the "winner". Finally, if both prediction accuracy and number of instance vectors were equal for x and y, the solution with the smaller MSE on the test set was judged to be the "winner" [1]. This comparison has been made twice during each evolutionary cycle, once when parents are selected in the binary tournament selection procedure, we choose winner individual according to the above mentioned procedure and once when we want to eliminate the worst fit individual from the population, we select the worst individual based on the above mentioned comparison [1].

B. Radial Basis Function Neural Networks (RBFNNs)

Radial basis function (RBF)[3] networks typically have three layers: an input layer, a hidden layer with a nonlinear RBF activation function and a linear output layer. The input can be modeled as a vector of real numbers $\mathbf{x} \in \mathbb{R}^n$.The output of the network is then a scalar function of the input vector, $\phi : \mathbb{R}^n \to \mathbb{R}$ and is given by

$$\phi(x) = \sum_{i=1}^{N} w_i \rho(||\mathbf{x} - \mathbf{c_i}||) \tag{4}$$

where N is the number of neurons in the hidden layer, $\mathbf{c_i}$ is the center vector for neuron i, and w is the weight of neuron i in the linear output neuron. $\rho(\cdot)$ is the the radial basis function is commonly taken to be Gaussian

$$\rho(||\mathbf{x} - \mathbf{c_i}||) = \exp(-||\mathbf{x} - \mathbf{c_i}||^2) \tag{5}$$

1) A RBFNN without Kohonen Unsupervised Learning and Backpropagation: To determine the K neurons, we randomly generate k clusters with centers and calculate the standard deviation σ, instead of using unsupervised learning algorithm. After that, we apply stochastic gradient descent to minimize the mean squared error.

$$\nabla \mathbf{w} = \rho_i(\mathbf{x_j}, \mathbf{c_i})(d_j - \phi(\mathbf{x_j})) \tag{6}$$

$$\mathbf{w_{i+1}} = \mathbf{w_i} + \eta \nabla \mathbf{w} \tag{7}$$

where $\mathbf{x_j}$ is a training instance, and $\mathbf{c_i}$ is cluster of each clusters.

2) A RBFNN with Kohonen Unsupervised Learning and Backpropagation: We employ Kohonen (Winner-Take-All) method to determine the K neurons using, which is unsupervised learning algorithm. For Kohonen (Winner-Take-All) method, we first randomly generate K vectors $(c_1, c_2, \cdots, c_k)$, and then for each training instance t_i, we calculate the Manhattan distance between the instance t_i and the centers.

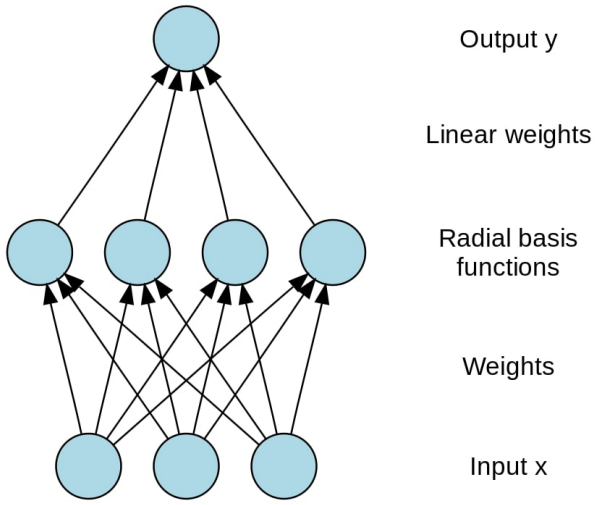

Fig. 1. Architecture of a radial basis function network. An input vector x is used as input to all radial basis functions, each with different parameters. The output of the network is a linear combination of the outputs from radial basis functions.

Based on the Manhattan distance, we will find the closest c for that instance and label this c vector c_{winner}. Finally, we will update the c_{winner} by

$$\nabla c_{winner} = x_i - c_{winner} \tag{8}$$

$$c_{winner} = c_{winner} + \eta \nabla c_{winner} \tag{9}$$

To get optimal weights, we still employ stochastic gradient descent method to minimize the mean squre error like pervious section by backpropagation.

C. Support Vector Machines (SVMs)

Support-vector machines (SVMs, also support-vector networks[4]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis[4].

Given a training set of N data points $y_k, x_k{}_{k=1}^{N}$, where $x_k \in Rn$ is the kth input pattern and $y_k \in \mathbb{R}$ is the kth output pattern, the support vector method approach aims at constructing a classifier of the form:

$$y(x) = \text{sign}(\sum_{k=1}^{N} \alpha_k y_k \phi(x, x_k) + b) \tag{10}$$

where α_k are positive real constants and b is a real constant. For $\phi(\cdot,\cdot)$ one typically has the following choices:

1) Linear support vector machine

$$\phi(x, x_k) = x^T \tag{11}$$

2) Radial basis function support vector machine

$$\phi(x, x_k) = \exp(\frac{-||x - x_k||^2}{2\sigma^2}) \tag{12}$$

The classifier is constructed as follows. One assumes that

$$w^T \phi(x_k) + b \geq 1, \text{if} \quad y_k = +1 \tag{13}$$

$$w^T \phi(x_k) + b \leq -1, \text{if} \quad y_k = -1 \tag{14}$$

which is equivalent to

$$y_k[w^T \phi(x_k) + b] \geq 1, \quad y_k = 1, \cdots N, \tag{15}$$

where $\phi(\cdot)$ is a nonlinear function which maps the input space into a higher dimensional space.

D. Feedforward Neural Network(FFNN)

Feed forward neural network is also known as Multi-Layer Percepetron (MLP) neural network. A MLP network shows optimal results for classification problems as well as regression problems [5]. The name multi-layer percepetron comes from its architecture where the layered structure of percepetron neurons are interconnected with each other as shown in Figure 2.

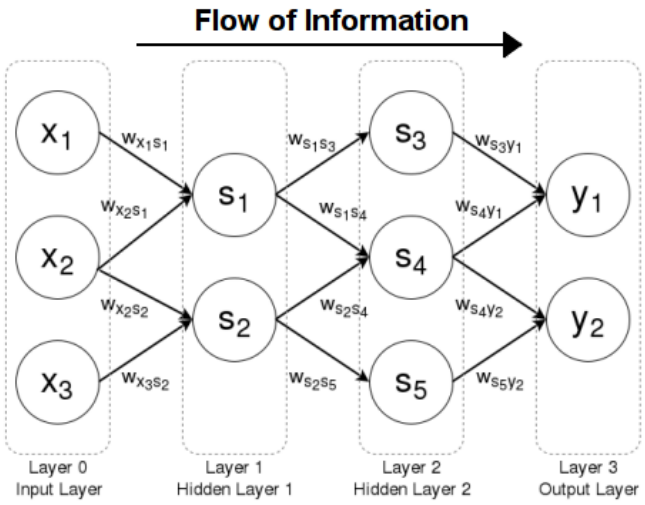

Fig. 2. A simple feedforward neural network architecture with multiple hidden layers [6]

The Feedforward networks have the flow of information in a single direction from the input layer through the hidden layers to the output layer. There are no feedback in the flow of information hence these types of neural networks are referred to as the feed-forward neural networks. The function of hidden neurons is to intervene between the external input and the network output in some useful manner. Existence of one or more hidden layers enable the network to extract higher-order statistics [7].

While solving regression problems using neural networks, it is necessary to figure out and implement appropriate number of hidden layers and the number of neurons in each hidden layer to prevent overfitting of the network since an overfitted network is unable to predict an output if the given input pattern to the neural network was not in the training dataset. The best approach to find the optimum number of neurons in a neural network is to use the trial and error [5] approach with a training dataset, a validation dataset and a test dataset. The training dataset is used to train the neural network. The validation dataset is used to tune the parameters of the nerual network and the test dataset is used to test the neural network for the performance matrix.

III. Experiment

In order to test the performance of the GRNNs, RBFNNs, and FFNNs tasks, a given dataset is used for training. The dataset is generated using a SSGA algorithm. It contains three columns. The first column shows the x value, the second column is the y value and the third column is the calculated Schaffer's F6 function. Several python packages were used to code the above mentioned tasks such as: Scikit-Learn [8]. The results of each task together with their fine-tuned hyper-parameters are elaborated in the following section. For testing the third task, SVMs, a second data set was created from the given original data set, where any training instance with a desired output > 0.5 is relabeled as 1.0 and any training instance that has a desired output <= 0.5 is relabeled as -1.0. The SVMs results are also presented in the following section, Section IV.

IV. Results

As shown in Table I, the radial basis function kernel support vector machine significantly outperforms other methods by achieving much higher accuracy, and GRNN-I achieves the second-best performance among all the other methods. Moreover, we visualize the prediction result of feedforward neural network with four hidden layers in Fig.3, as shown in the figure, FFNN can find a close function approximation mapping between the features, X and Y, and the Schaffer's F6 function value, Z. Also based on table I, one can observe that EGRNN-I can outperform the simple EGRNN both with respect to the accuracy measure and the number of training instances.

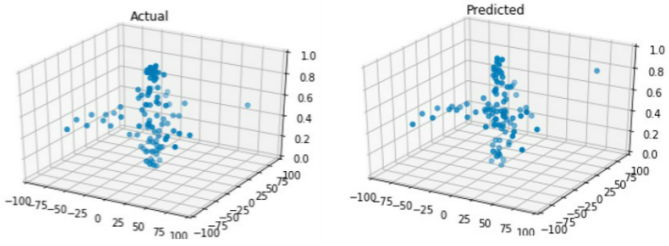

Fig. 3. True values in the test dataset vs predicted values using FFNN.

V. Conclusion

Our results demonstrate the strong performance of the RBF-SVMs with respect to accuracy and F1 score. The EGRNN-I achieves a good accuracy measure as well compared to the other methods and it also uses 477 instance to achieve this measure. The EGRRN-I represents a very promising approach for scalable instance-based learners, however when the population size and the number of function evaluation grows, the running time also grows too large to evolve to a good σ value. With the feedforward neural networks (FFNN), it showed less accuracy for one and two hidden layers. The accuracy started to improve as the third and the fourth layer added to the architecture. Increasing the number of neurons in each layer also improved the

TABLE I

PREDICTION PERFORMANCE FOR MACHINE LEARNING METHODS WITH DIFFERENT HYPER PARAMETERS

	EGRNN	EGRNN-I	RBFNN	RBFNN-K	SVM(Linear)	SVM(RBF)	NN(1 layer)	NN(2 layers)	NN(4 layers)
Parameters	$\sigma = 0.1$	$\sigma = 0.31$	$K = 8$	$K = 8$	None	None	$\alpha = 0.2$	$\alpha = 0.06$	$\alpha = 0.3$
	$N_{Max} = 500$	$N_{Max} = 500$	$I = 3500$	$I = 3500$	None	None	$I = 1e5$	$I = 5e5$	$I = 2e5$
	$pop_{size} = 50$	$pop_{size} = 50$	None	None	None	None	$N_k = (10)$	$N_k = (20, 10)$	$N_k = (15, 10)$
	700	$N_{instance} = 477$	None	None	None	None	None	None	$N_k = (10, 10)$
MSE	0.034	0.070	0.12	0.08	None	None	0.065	0.038	0.012
Accuracy	0.77	0.78	0.53	0.61	0.51	**0.83**	0.62	0.65	0.76
Recall	0.74	0.74	0.61 0.30	0.40	0.1	0.78	0.42	0.48	0.59
Precision	0.76	0.77	0.50	0.62	0.47	0.87	0.64	0.67	0.84
F1	0.75	0.76	0.38	0.47	0.15	0.82	0.51	0.58	0.69

accuracy, however after a certain point the decrease in accuracy and precision was noticed due to the over-fitting of the neural network. The RBF-SVM achieves the best prediction performance since it overcomes the non-linear separability by introducing the radial basis function, which allows an implicit projection into infinite dimensions.

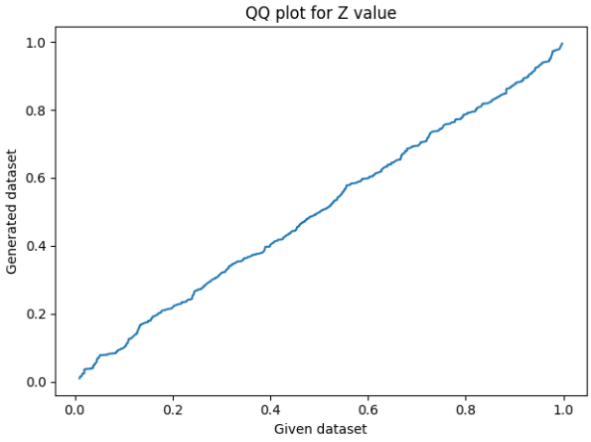

Fig. 4. QQ plot

VI. BREAKDOWN OF THE WORK

Each author coded one of the Tasks. However, since we are in group of three, Ye Wang also coded the SVMs task. At the end, all the authors reviewed the other authors' codes and results. Also, for writing the paper all the group members contributed equally.

APPENDIX

A. EXTRA CREDIT

In this Section we explain our methodology to find which of the population sizes was used to generate the given dataset(3, 12, 25, 50, 100) given a mutation amount of 0.01. For this purpose we used the codes for the first class assignment, the Simple Exploratory Attacker, We modified the code such that all the instances of the "ANINDIVIDUAL" class can be tracked and stored. Then we ran the code for different population sizes and collect the generated individuals during 1000 evolutionary cycles. Once the data sets are generated, we plot the QQ plot and run the student t-test for the z values to see whether the generated data set and the given data set are similar. T-test tells that whether there is a significant difference between the mean of the two data-set, while the QQ plot (figure 4) tells whether the data-sets follow the same distribution. T-test Result for a data-set generated using population size of 50 are as follows : (statistic=-0.070, p-value=0.943), the high p-value proves the fact that there is no significant difference between the mean of the two data-sets. Also based on figure 4, the slop of the line is close to one which is another evidence in support of our hypothesis that the population size is indeed 50.

REFERENCES

[1] K. Casey, A. Garrett, J. Gay, L. Montgomery, and G. Dozier, "An evolutionary approach for achieving scalability with general regression neural networks," *Natural Computing*, vol. 8, no. 1, pp. 133–148, Mar 2009. [Online]. Available: https://doi.org/10.1007/s11047-007-9052-x

[2] D. F. Specht, "A general regression neural network," *IEEE Transactions on Neural Networks*, vol. 2, no. 6, pp. 568–576, Nov 1991.

[3] S. E. Vt and Y. C. Shin, "Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems," *IEEE transactions on neural networks*, vol. 5, no. 4, pp. 594–603, 1994.

[4] B. Boser *et al.*, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[5] B. M. Wilamowski, "How to not get frustrated with neural networks," *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 5–11, 2011.

[6] M. John, G. José, Alonso, M. Saruque, K. Jimin, and Yushi, "Feedforward Neural Networks." [Online]. Available: https://brilliant.org/wiki/feedforward-neural-networks/

[7] M. H. SAZLI, "A brief review of feed-forward neural networks," *Communications, Faculty Of Science, University of Ankara*, no. January 2006, pp. 11–17, 2006. [Online]. Available: http://acikarsiv.ankara.edu.tr/browse/4026/3746.pdf?show

[8] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.