Data Discretization Using the Extreme Learning Machine Neural Network

Juan Jesús Carneros, José M. Jerez, Iván Gómez, and Leonardo Franco

Universidad de Málaga, Department of Computer Science, ETSI Informática, Spain
jcarnego@hotmail.com, {jja,ivan,lfranco}@lcc.uma.es

Abstract. Data discretization is an important processing step for several computational methods that work only with binary input data. In this work a method for discretize continuous data based on the use of the Extreme Learning Machine neural network architecture is developed and tested. The new method does not use data labels for performing the discretization process and thus is suitable for supervised and supervised data and also, as it is based on the Extreme Learning Machine, is very fast even for large input data sets. The efficiency of the new method is analyzed on several benchmark functions, testing the classification accuracy obtained with raw and discretized data, and also in comparison to results from the application of a state-of-the-art supervised discretization algorithm. The results indicate the suitability of the developed approach.

Keywords: Neural networks, Supervised learning, Extreme learning machine, Generalization, Discretization.

1 Introduction

Discretization techniques play an important role in the areas of data mining and knowledge discovery. Not only they help to produce a more concise data representation but also are a necessary step for the application of several classification algorithms, like Decision Trees, Logical Analysis of Data, DASG algorithm, etc. [1,8]. Discretization methods can be characterized according to at least five different features: supervised or unsupervised, static or dynamic, global or local, bottom-up or top-down, and direct or indirect methods. We will not do an in-depth analysis of these characteristics (see [6] for a more detailed analysis) but will mention the relevant ones in relationship to the method to be proposed below. The use or not of the label (or class) of the data for the discretization process distinguish a supervised from an unsupervised method; global methods use all set of features and data for the processing while local methods use only a portion of them; direct methods requires the user to decide on the characteristics of the output while, in general, incremental methods works by applying some predefined conditions towards the fulfillment of a goal. We present in this work a discretization algorithm constructed by using the Extreme Learning Machine (ELM) model that will be unsupervised, local and incremental. The ELM algorithm proposed by Huang et al. [5,4], as a fast and accurate neural network

T. Huang et al. (Eds.): ICONIP 2012, Part IV, LNCS 7666, pp. 281–288, 2012.

based classification system, construct feedforward neural network architectures containing a single hidden layer of neurons with the peculiarity that only the set of synaptic weights connecting the hidden layer to the output need to be learnt, as the connections between the input and the middle layer have randomly chosen values. In this way, not only the process is much faster but also can be performed deterministically by solving a system of equations, without depending on gradient descent based algorithms. Despite its simplicity, and the use of random weights in the first layer, the results obtained when using the ELM are surprisingly good, in most cases reaching a similar level of performance than more complex algorithms. The ELM normally uses continuos valued neurons in the middle layer but a version operating with discrete neurons has been proposed [4], and it is this version that we used to build a discretization algorithm.

2 Methods

2.1 A Discretizer Based on the Extreme Learning Machine

The Extreme Learning Machine is a neural based architecture for classification problems that can be trained in a supervised way. The network architecture contains a single layer of neurons that receive the information from the input neurons through random valued weights, and thus the training of the network only involves finding the value of the synaptic weights connecting the middle layer to the output neurons. This training process can be performed very fastly and straightforward as essentially consists in a matrix inversion.

For N input patterns $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in \mathbf{R}^n$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in \mathbf{R}^m$, the output vector $\mathbf{o}_i$ of an ELM with L neurons in the hidden layer and activation function $g(x)$ can be modelled mathematically as:

$$\sum_{i=1}^{L} \beta_i g\left(\mathbf{w}_i \,.\, \mathbf{x}_j + b_i\right) = \mathbf{o}_j, \qquad j = 1, \ldots, N, \tag{1}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^T$ is the vector of synaptic weights connecting the i^{th} hidden neuron to the input neurons, $\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the vector connecting the hidden neurons to the output ones, and b_i indicates the bias of the i^{th} hidden neuron. An ELM with m output neurons that can approximate N patterns with zero error implies that the $\sum_{i=1}^{N} \|\mathbf{o}_i - \mathbf{t}_i\| = 0$, expression that can be written in matrix form as:

$$\mathbf{H}\beta = \mathbf{T}, \tag{2}$$

where $\mathbf{H}$ represents the activity of the hidden neurons when an input pattern is presented, $\mathbf{T}$ represents the target outputs, and β are the weights to be find in order to verify the learning of the input-output relationship. The value of β with smallest norm that verifies in the minimum square sense the previous lineal system is:

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{T}, \tag{3}$$

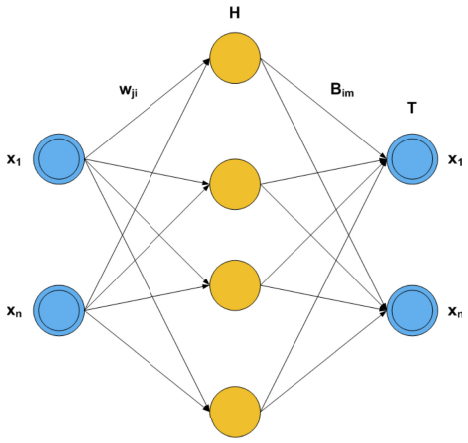

Fig. 1. Structure of an ELM used for data discretization. The number of output neurons is equal to the number of inputs and for each pattern of the data set the desired outputs (targets) are equal to the input values. Using discrete units in the hidden layer permits to obtain after training the network a discrete representation of the inputs.

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$ [5,4]. In the original ELM architecture [5] the activation function of the neurons in the middle layer is continuous valued but binary units can be used, and it has been shown that its functioning and the results that can be obtained are quite similar [4].

To build a discretizer based on the ELM, it is only necessary to use discrete valued hidden units, select the number of neurons to include in this hidden layer, and train the network as described above, with patterns that have the same output as the input. A network architecture of a discretizer constructed for the case of input vectors of length 2 is depicted in Figure 1, where the hidden layer, from which the discretized representation will be obtained, contains 3 neurons in this example. If the total error of the network, measured as the difference between input and output across all training patterns, is zero then the discretization obtained in the hidden layer contains all the information of the original input; in other cases the error quantifies the information loss during the discretization process.

2.2 The CAIM Discretization Algorithm

We briefly explain some characteristics of the CAIM discretization algorithm because it will be used as a reference for comparing the efficiency of the new developed algorithm. The CAIM algorithm is a supervised discretization method that tries to maximize the class-attribute interdependency by exploring a series of discretization points, to choose those that optimize an heuristic measure related to the dominancy of a given class in the created intervals, including also a factor so to minimize the number of intervals [6]. It is a local algorithm that consider the input attributes independently, usually generating for each input variable a

discrete representation with a length equal to the number of output classes (e.g., twice the original number of attributes for a binary output). The algorithm has been extensively tested leading to very good results and in several cases leading to classification rates larger than those obtained with the original data, fact that can be due to a noise-filtering process produced as a side-effect in the discretization process.

2.3 The C-Mantec Constructive Neural Network Algorithm

C-Mantec is a recently introduced neural based constructive algorithm for supervised learning tasks that generates very compact neural network architectures with high generalization capabilities [7,9] . A difference with previous constructive algorithms is that C-Mantec permits the architectures to grow as required by the learning of patterns but without freezing the values of existing synaptic weights, in a scheme where the neurons compete for the incoming knowledge and in which the learning at the level of individual neurons is performed by the thermal perceptron. The algorithm is used in this work to test the generalization ability of the several representation of the data obtained by applying the ELM discretizer because of its observed good generalization capabilities but also because training computational are much reduced in comparison to traditional MLPs.

2.4 The Generalization Complexity Measure

The GC measure was introduced by Franco and colleagues [2] and was derived from evidence that pairs of bordering examples, those lying closely at both sides of the separating hyperplanes that classify different regions, play an important role on the generalization ability obtained in classification problems when neural networks are used as predictive methods. The idea behind the GC measure is to build an estimate of the generalization ability that can be obtained beforehand but also to be used to estimate the size of an adequate neural architecture [3]. As the GC measure has been defined only for binary input values, we used it in this work as an additional test of the quality of the representations obtained using the ELM discretizer developed.

3 Results

We have applied the ELM discretizer using several size architectures to a set of 20 benchmark data sets. To choose the number of neurons in the hidden layer of the ELM, i.e., the size of the representation, we took the number of neurons generated by CAIM and consider this value as a reference, and thus we try representations with the same number of units used by CAIM, twice and five times this number, and also consider a number of hidden units necessary to achieve a 0.01 level of error.

Table 1. Characteristics of the 20 data sets selected from the PROBEN1 benchmark set to carry out the tests (see the text for more details)

Id	Set	Patterns	Class distribution % (0/1)	Inputs	Neurons CAIM	Neurons ELM (Error < 0.01)
f_1	cancer1	699	65 - 35	9	18	527
f_2	card1	690	45 - 55	51	100	641
f_3	diabetes1	768	35 - 65	8	16	818
f_4	gene1a	3175	76 - 24	120	240	2977
f_5	gene1b	3175	76 - 24	120	240	2600
f_6	gene1c	3175	48 - 52	120	240	2600
f_7	glass1a	214	67 - 33	9	18	215
f_8	glass1a	214	64 - 36	9	18	243
f_9	glass1b	214	92 - 8	9	18	251
f_{10}	glass1c	214	94 - 6	9	18	235
f_{11}	glass1d	214	96 - 4	9	18	243
f_{12}	glass1e	214	86 - 14	9	18	251
f_{13}	heart1a	920	44 - 56	35	67	824
f_{14}	heartc1b	303	54 - 46	35	59	300
f_{15}	horse1a	364	38 - 62	58	116	347
f_{16}	horse1b	364	75 - 25	58	116	347
f_{17}	horse1c	364	85 - 15	58	116	347
f_{18}	thyroid1a	7200	97 - 13	21	42	538
f_{19}	thyroid1b	7200	95 - 5	21	42	529
f_{20}	thyroid1c	7200	7 - 93	21	42	520

Table 1 shows the characteristics of the 20 binary output data sets taken from the Proben benchmark set used for testing the discretization performance of the proposed algorithm. The first column of the table indicates the identification reference of the data set, and the rest of columns indicate its the name, the number of patterns available, the class distribution, the number of inputs, the number of discretization intervals (or equivalently the number of neurons) needed by the CAIM algorithm, and the number of neurons used in the developed discretizer needed to achieve a codification error below 0.01.

Table 2 shows the generalization ability results obtained in a ten fold cross-validation approach. The first column indicates the function used and the rest of columns the generalization ability obtained using the several representations of the function: the original continuous input data, the CAIM discretized inputs, the ELM based discretization with the same number of neurons as used with CAIM, twice and five times the previous value and finally with a number of attributes needed to achieve a 0.01 or less of training error (value indicated in the last column of Table 1).

Regarding the computational costs involved in the tests carried out, we report the average CPU times needed on an Intel Core 2 Duo E7300 PC running at 2.66GHz with Windows Vista OS for discretizing the 20 data sets analyzed. On average the CAIM algorithm needed 24.03 seconds per function, while the ELM based discretizer needed 0.05, 0.30 and 1.91 seconds respectively for the case of

Table 2. Generalization ability obtained for the 20 benchmark functions extracted from the Proben benchmark set. The function identifier is shown in the first column, and the rest of values shown indicate the generalization for the original continuous input data (no discretization applied), for the discretization obtained from CAIM, and for the ELM based discretizer with the different representations sizes considered. (See the text for details).

Function id	Continuos input	CAIM	ELM nh	ELM 2nh	ELM 5nh	ELM (Error < 0.01)
f_1	96.19	96.57	93.05	95.24	94.57	94.76
f_2	74.20	75.56	70.05	74.88	72.46	72.85
f_3	54.96	75.13	67.57	68.78	60.17	54.26
f_4	85.15	86.09	80.55	82.00	82.92	83.34
f_5	79.24	81.37	75.84	77.21	78.28	77.65
f_6	78.38	78.32	71.66	73.93	75.82	75.80
f_7	69.69	80.31	67.19	75.31	74.06	73.44
f_8	66.88	78.75	66.56	68.13	71.25	70.94
f_9	84.06	88.44	93.13	90.94	90.00	90.63
f_{10}	94.69	91.88	94.38	93.44	93.75	96.88
f_{11}	97.50	99.38	95.63	97.19	97.50	96.56
f_{12}	94.06	95.31	95.31	95.31	95.00	95.63
f_{13}	67.68	72.68	63.12	67.03	64.35	64.49
f_{14}	74.73	80.00	73.85	70.77	76.48	73.63
f_{15}	68.07	70.28	68.26	67.34	68.81	68.62
f_{16}	74.68	76.51	74.31	72.84	76.33	74.50
f_{17}	85.14	85.32	81.28	82.39	82.02	84.22
f_{18}	98.59	99.03	97.90	94.72	98.05	98.30
f_{19}	93.85	97.35	94.63	91.29	94.69	94.48
f_{20}	93.10	98.95	92.61	90.82	92.73	92.87
$Mean$	81.54	85.36	80.84	81.48	81.96	81.69

networks having the same number of outputs than the CAIM algorithm, twice and five times this value respectively.

Further, we have computed the Generalization Complexity (GC) of the several data sets used. This measure is related to the generalization ability that can be expected when a function is implemented in a neural network or other prediction system. The GC measure has been defined only for Boolean inputs and thus it is useful to analyze the values obtained with the different discretized representations considered previously. Table 3 shows the value obtained for the Generalization complexity for the 20 functions indicated in the first column. Columns 2 to 5 show the values of GC using the 5 considered discretization schemes: CAIM, ELM with same number of attributes than CAIM, twice and five times the previous value, and with a number of neurons enough to reduce the representation error below 0.01. The last row of the table shows the average of column values.

As the GC measure can only be computed for binary data, given a continuous input data it is not clear its true value as the data needs to be transformed.

Table 3. The Generalization Complexity values computed the 20 test data sets using 5 different discretization representation: CAIM, ELM nh, ELM 2nh, ELM 5nh, ELM with an error less than 0.01. Average column values are indicated in the last row.

Function id	CAIM	ELM nh	ELM 2nh	ELM 5nh	ELM (Error < 0.01)
f_1	0.0525	0.1440	0.0700	0.0473	0.0401
f_2	0.2033	0.2191	0.1969	0.2053	0.2109
f_3	0.3761	0.4481	0.4219	0.3356	0.3027
f_4	0.1617	0.2139	0.1860	0.1854	0.1761
f_5	0.1764	0.2334	0.2059	0.1792	0.1762
f_6	0.2417	0.2935	0.2763	0.2744	0.2601
f_7	0.3105	0.3860	0.3397	0.2823	0.2103
f_8	0.4048	0.4263	0.3765	0.2614	0.2240
f_9	0.1570	0.1543	0.1262	0.1262	0.1090
f_{10}	0.0591	0.1210	0.0944	0.0380	0.0506
f_{11}	0.0566	0.0768	0.0613	0.0389	0.0257
f_{12}	0.0573	0.1801	0.0491	0.0254	0.0288
f_{13}	0.2412	0.2562	0.2381	0.2579	0.2458
f_{14}	0.2333	0.2693	0.2294	0.2393	0.2525
f_{15}	0.3097	0.3300	0.3574	0.3810	0.3537
f_{16}	0.2454	0.2689	0.2889	0.2540	0.2807
f_{17}	0.1861	0.1832	0.1964	0.1882	0.1850
f_{18}	0.0120	0.0503	0.0271	0.0172	0.0130
f_{19}	0.0553	0.0915	0.0917	0.0919	0.0819
f_{20}	0.0543	0.1266	0.1269	0.1052	0.0878
$\overline{f}$	0.1797	0.2236	0.1980	0.1767	0.1657

Nevertheless, as the GC measure has been validated before [2] as a good predictor for the generalization ability, we computed the correlation between the generalization ability obtained using the original continuos data and the GC obtained from the several discretization schemes. The results show a strong correlation value in all cases (measured by the absolute value of the Pearson correlation coefficient), with largest values observed for the case of the CAIM discretized data (r=0.89) and also for the ELM based discretizer with twice as much bits as those generated by CAIM (r=0.86).

4 Conclusions

We have tested in this work the implementation of a non-supervised discretization algorithm based on the ELM neural network architecture and compared the results with those obtained using a supervised discretization method, named CAIM, known for its good performance. To compare the quality of the representations obtained we have analyzed the generalization ability using 20 benchmark data sets. The results show that the same of level of generalization than for the case of using the original continuous data can be obtained by using approximately five times more neurons than those selected by the CAIM algorithm,

that normally chooses a representation two times larger than the number of input features for binary classification problems, as it is the case of the analyzed data sets. Considering that the CAIM algorithm is a supervised method, the previous result can be considered useful, also because of the speed of the ELM based discretizer, several times faster in comparison to the CAIM algorithm. A further test of the quality of the discretization representation obtained with the ELM-based discretizer comes from an experiment carried to measure the value of the Generalization Complexity of the data sets, finding results quite close to those obtained when using the CAIM algorithm. As an overall conclusion we can state that the experiments carried out in this work shows the suitability of using the ELM as a discretization algorithm, highlighting that even if it generates larger data representations in comparison to a performant supervised algorithm like CAIM, the process is much faster and permits to work both with supervised and unsupervised data.

Acknowledgements. The authors acknowledge support from CICYT (Spain) through grants TIN2008-04985 and TIN2010-16556 (including FEDER funds) and from Junta de Andalucía through grant P08-TIC-04026.

References

1. Boros, E., Hammer, P.L., Ibaraki, T., Kogan, A., Mayoraz, E., Muchnik, I.B.: An Implementation of Logical Analysis of Data. IEEE Trans. Knowl. Data Eng. 12, 292–306 (2000)
2. Franco, L., Anthony, M.: The Influence of Oppositely Classified Examples on the Generalization Complexity of boolean functions. IEEE Trans. Neural Netw. 17(3), 578–590 (2006)
3. Gómez, I., Franco, L., Jerez, J.M.: Neural Network Architecture Selection: Can Function Complexity Help? Neural Proc. Lett. 30, 71–87 (2009)
4. Huang, G.B., Zhu, Q.Y., Mao, K.Z., Siew, C.K., Saratch, P., Sundararajan, N.: Can Threshold Networks be Trained Directly. IEEE Trans. Circuits Syst. II. 53, 187–191 (2006)
5. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks. In: Proc. Int. Joint Conf. Neural Networks, pp. 985–990 (2004)
6. Kurgan, L.A., Cios, K.J.: Caim Discretization Algorithm. IEEE Trans. on Knowl. and Data Eng. 16(2), 145–153 (2004)
7. Subirats, J.L., Franco, L., Jerez, J.M.: C-mantec: a Novel Constructive Neural Network Algorithm Incorporating Competition between Neurons. Neural Networks 26, 130–140 (2012)
8. Subirats, J.L., Jerez, J.M., Franco, L.: A New Decomposition Algorithm for Threshold Synthesis and Generalization of Boolean Functions. IEEE Trans. on Circuits and Systems 55-I(10), 3188–3196 (2008)
9. Urda, D., Cañete, E., Subirats, J.L., Franco, L., Llopis, L., Jerez, J.M.: Energy Efficient Reprogramming in wsn Using Constructive Neural Networks. International Journal of Innovative Computing, Information and Control 8 (2012)