Classification of Ordinal Data Using Neural Networks

Joaquim Pinto da Costa[1] and Jaime S. Cardoso[2]

[1] Faculdade Ciências Universidade Porto,
Porto, Portugal
jpcosta@fc.up.pt
[2] Faculdade Engenharia Universidade Porto / INESC Porto,
Porto, Portugal
jaime.cardoso@inescporto.pt

Abstract. Many real life problems require the classification of items in naturally ordered classes. These problems are traditionally handled by conventional methods for nominal classes, ignoring the order. This paper introduces a new training model for feedforward neural networks, for multiclass classification problems, where the classes are ordered. The proposed model has just one output unit which takes values in the interval [0,1]; this interval is then subdivided into K subintervals (one for each class), according to a specific probabilistic model. A comparison is made with conventional approaches, as well as with other architectures specific for ordinal data proposed in the literature. The new model compares favourably with the other methods under study, in the synthetic dataset used for evaluation.

1 Introduction

Many pattern recognition problems involve classifying examples into classes which have a natural ordering. Settings in which it is natural to rank instances arise in many fields, such as information retrieval [1], collaborative filtering [2] and econometric modelling [3].

Suppose that examples in a classification problem belong to one of K classes, numbered from 1 to K, corresponding to their natural order if one exists, and arbitrarily otherwise. The learning task is to select a prediction function $f(\mathbf{x})$ from a family of possible functions that minimizes the expected *loss*.

Although conventional methods for nominal classes could be employed, the use of techniques designed specifically for ordered classes results in simpler classifiers, making it easier to interpret the factors that are being used to discriminate among classes [3]. We propose a classifier for ordered classes based on neural networks by imposing that the output layer with K units has just one mode, corresponding to the predicted class. In fact the model is equivalent to a network with just one output unit, as the final layer serves only to compute the error and has no weights associated with its input values (see figure 1).

The novel proposed algorithm attains the best generalization capability for the group of methods evaluated.

J. Gama et al. (Eds.): ECML 2005, LNAI 3720, pp. 690–697, 2005.

2 A Neural Network Architecture for Ordinal Data

To use a neural network for classification, we need to construct an equivalent function approximation problem by assigning a target value for each class. For a two-class problem we can use a network with a single output, and binary target values: 1 for one class, and 0 for the other. The training of the network is commonly performed using the popular mean square error. For multiclass classification problems (1-of-K, where $K > 2$) we use a network with K outputs, one corresponding to each class, and target values of 1 for the correct class, and 0 otherwise. Since these targets are not independent of each other, however, it is no longer appropriate to use the same error measure. The correct generalization is through a special activation function (the softmax) designed so as to satisfy the normalization constraint on the total probability.

However this approach does not retain the ordinality or rank order of the classes and is not, therefore, appropriate for ordinal multistate classification problems.

Let us formulate the problem of separating K ordered classes $\mathcal{C}_1, \cdots, \mathcal{C}_K$. Consider the training set $\{\mathbf{x}_i^{(k)}\}$, where $k = 1, \cdots, K$, denotes the class number, $i = 1, \cdots, \ell_k$ is the index within each class, and $\mathbf{x}_i^{(k)} \in \mathbb{R}^p$. Let $\ell = \sum_{k=1}^{K} \ell_k$ be the total number of training examples.

Given a new query point $\mathbf{x}$, Bayes decision theory suggests to classify $\mathbf{x}$ in the class which maximizes the *a posteriori* probability $P(\mathcal{C}_k|\mathbf{x})$. To do so, one usually has to estimate these probabilities, either implicitly or explicitly. Suppose for instance that we have 7 classes and, for a given point $\mathbf{x}_0$, the highest probability is $P(\mathcal{C}_5|\mathbf{x}_0)$; we then attribute class $\mathcal{C}_5$ to the given point. If there is not an order relation between the classes, it is perfectly natural that the second highest *a posteriori* probability is, for instance, $P(\mathcal{C}_2|\mathbf{x})$. However, if the classes are ordered, $\mathcal{C}_1 < \mathcal{C}_2 <, \ldots, < \mathcal{C}_7$, classes $\mathcal{C}_4$ and $\mathcal{C}_6$ are closer to class $\mathcal{C}_5$ and therefore the second and third highest *a posteriori* probabilities should be attained in these classes. This argument extends easily to the classes, $\mathcal{C}_3$ and $\mathcal{C}_7$, and so on. This is the main idea behind the method proposed here, which we now detail.

Our method assumes that in a supervised classification problem with ordered classes, the random variable class associated with a given query $\mathbf{x}$ should be unimodal. That is to say that if we plot the *a posteriori* probabilities $P(\mathcal{C}_k|\mathbf{x})$, from the first $\mathcal{C}_1$ to the last $\mathcal{C}_K$, there should be only one mode in this graphic. Here, we apply this idea in the context of neural networks. Usually in neural networks, the output layer has as many units as there are classes, K. We will use the same order for these units and the classes. In order to force the output values (which represent the *a posteriori* probabilities) to have just one mode, we will use a parametric model for these output units. This model consists in assuming that the output values come from a binomial distribution, $B(K-1, p)$. This distribution is unimodal in most cases and when it has two modes, these are for contiguous values, which makes sense in our case, since we can have exactly the same probability for two classes. This binomial distribution takes integer values in the set $\{0, 1, \ldots, K-1\}$; value 0 corresponds to class $\mathcal{C}_1$, value

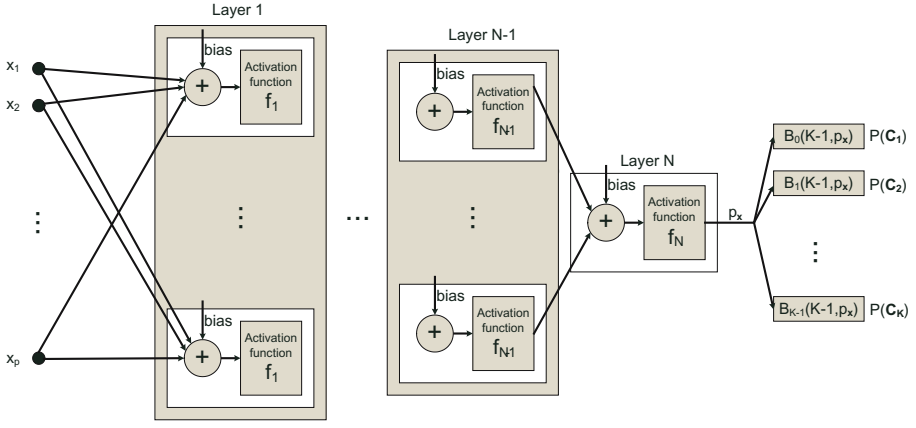


Fig. 1. unimodal neural network architecture

1 to class $\mathcal{C}_2$ and so on until value $K-1$ to class $\mathcal{C}_K$. As K is known, the only parameter left to be estimated from this model is the probability p. We will therefore use a different architecture for the neural network; that is, the output layer will have just one output unit, corresponding to the value of p – figure 1. For a given query $\mathbf{x}$, the output of the network will be a single numerical value in the range [0,1], which we call $p_{\mathbf{x}}$. Then, the probabilities $P(\mathcal{C}_k|\mathbf{x})$ are calculated from the binomial model:

$$P(\mathcal{C}_k|\mathbf{x}) \;=\; \frac{(K-1)!p_{\mathbf{x}}^{k-1}(1-p_{\mathbf{x}})^{K-k}}{(k-1)!(K-k)!},\; k=1,2,\ldots,K$$

In fact these probabilities are calculated recursively, to save computing time:

$$\frac{P(\mathcal{C}_k|\mathbf{x})}{P(\mathcal{C}_{k-1}|\mathbf{x})} \;=\; \frac{p_{\mathbf{x}}(K-k+1)}{(k-1)(1-p_{\mathbf{x}})},\text{ and so } P(\mathcal{C}_k|\mathbf{x}) \;=\; P(\mathcal{C}_{k-1}|\mathbf{x})\frac{p_{\mathbf{x}}(K-k+1)}{(k-1)(1-p_{\mathbf{x}})}.$$

We start with $P(\mathcal{C}_1|\mathbf{x}) \;=\; (1-p_{\mathbf{x}})^{K-1}$ and compute the other probabilities, $P(\mathcal{C}_k|\mathbf{x}),\;\; k=2,3,\ldots,K$, using the above formula.

When the training case $\mathbf{x}$ is presented, the error is defined as

$$\sum_{k=1}^{K}\left(P(\mathcal{C}_k|\mathbf{x})-\delta(k-\mathcal{C}_{\mathbf{x}})\right)^2$$

where $\delta(n)=\begin{cases}1 \text{ if } n=0\\ 0 \text{ otherwise}\end{cases}$ and $\mathcal{C}_{\mathbf{x}}$ the true class of $\mathbf{x}$. The network is trained to minimize the average value over all training cases of such error. Finally, in the test phase, we choose the class k which maximizes the probability $P(\mathcal{C}_k)$.

3 Experimental Results

In this section we present experimental results for several models based on neural networks, when applied to a synthetic dataset.

3.1 Implementation Details

We compared the following algorithms:

- Conventional neural network (cNN). To test the hypothesis that methods specifically targeted for ordinal data improve the performance of a standard classifier, we tested a conventional feed forward network, fully connected, with a single hidden layer, trained with the traditional least square approach.
- Pairwise NN (pNN): Frank [4] introduced a simple algorithm that enables standard classification algorithms to exploit the ordering information in ordinal prediction problems. First, the data is transformed from a K-class ordinal problem to $K-1$ binary class problems. To predict the class value of an unseen instance the probabilities of the K original classes are estimated using the outputs from the $K-1$ binary classifiers.
- Costa [5], following a probabilistic approach, proposes a neural network architecture (itNN) that exploits the ordinal nature of the data, by defining the classification task on a suitable space through a "partitive approach". It is proposed a feedforward neural network with $K-1$ outputs to solve a K-class ordinal problem. The probabilistic meaning assigned to the network outputs is exploited to rank the elements of the dataset.
- proposed unimodal model (uNN), as previously introduced.

Experiments were carried out in Matlab 7.0 (R14), making use of the Neural Network Toolbox[1].

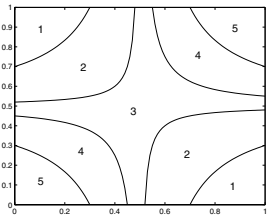
The first three models were configured with a single hidden layer and trained with Levenberg-Marquardt back propagation method, over 2000 epochs. The uNN model was also configured with a single hidden layer and trained with the `lsqnonlin` Matlab function over 1000 iterations.

The number of neurons in the hidden layer was experimentally set for the best performance.

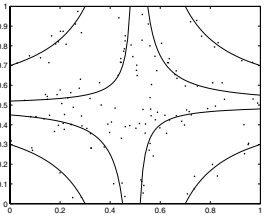
3.2 Measuring Classifier Performance

Having built a classifier, the obvious question is "how good is it?". This begs the question of what we mean by good. The obvious answer is to treat every misclassification as equally likely. This translates to adopting the non-metric indicator function $l_{0-1}(f(\mathbf{x}), y) = 0$ if $f(\mathbf{x}) = y$ and $l_{0-1}(f(\mathbf{x}), y) = 1$ if $f(\mathbf{x}) \neq y$, where $f(\mathbf{x})$ and y are the predicted and true classes, respectively. Measuring the performance of a classifier using the l_{0-1} loss function is equivalent to simply

[1] The code is available upon request to the authors.



(a) Classes' boundaries.



(b) Scatter plot of the data points wrongly ranked. Number of wrong points: 14.2%.

Fig. 2. Test setup for 5 classes in $\mathbb{R}^2$

considering the misclassification error rate (MER). However, for ordered classes, losses that increase with the absolute difference between the class numbers are more natural choices in the absence of better information [3].

The mean absolute error (MAE) criterion takes into account the degree of misclassification and is thus a richer criterion than MER. The loss function corresponding to this criterion is $l(f(\mathbf{x}), y) = |f(\mathbf{x}) - y|$.

A variant of the above MAE measure is the root mean square error (RMSE), where the absolute difference is replaced with the square of the difference, $l(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$.

Finally, the performance of the classifiers was also assessed with the Spearman coefficient (r_s), a nonparametric rank-order correlation coefficient, well established in the literature.

3.3 Experimental Methodology and Results

To check the adequacy of the proposed model we generated a synthetic dataset in a similar way to Herbrich [1].

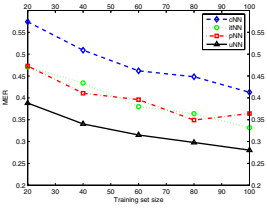
We generated 1000 example points $\mathbf{x} = [x_1\ x_2]^t$ uniformly at random in the unit square $[0,1] \times [0,1] \subset \mathbb{R}^2$. Each point was assigned a rank y from the set $\{1,2,3,4,5\}$, according to

$$y = \min_{r \in \{1,2,3,4,5\}} \{r : b_{r-1} < 10(x_1 - 0.5)(x_2 - 0.5) + \varepsilon \ < b_r\}$$

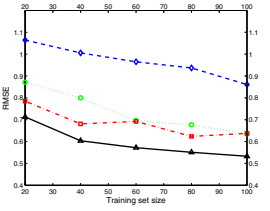
$$(b_0, b_1, b_2, b_3, b_4, b_5) = (-\infty, -1, -0.1, 0.25, 1, +\infty)$$

where ε is a random value, normally distributed with zero mean and standard deviation $\sigma = 0.125$. Figure 2(a) shows the five regions and figure 2(b) the points which were assigned to a different rank after the corruption with the normally distributed noise.

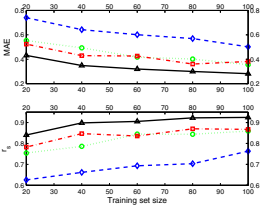
In order to compare the different algorithms, and similarly to [1], we randomly selected training sequences of point-rank pairs of length ℓ ranging from 20 to 100. The remaining points were used to estimate the classification error, which were



(a) MER criterion.

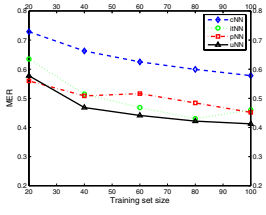


(b) RMSE criterion.

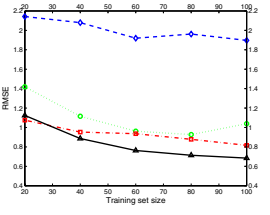


(c) MAE criterion and Spearman coefficient.

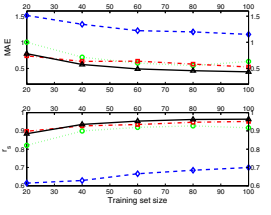
Fig. 3. Results for 5 classes in $\mathbb{R}^2$



(a) MER criterion.



(b) RMSE criterion.



(c) MAE criterion and Spearman coefficient.

Fig. 4. Results for 10 classes in $\mathbb{R}^2$

averaged over 10 runs of the algorithms for each size of the training sequence. Thus we obtained the learning curves shown in figure 3, for 5 neurons in the hidden layer.

Accuracy dependence on the number of classes. To investigate the relation between the number of classes and the performance of the evaluated algorithms, we also ran the four models on the same dataset but with 10 classes.

This time each point was assigned a rank y from the set $\{1,2,3,4,5,6,7,8,9,10\}$, according to

$$y = \min_{r\in\{1,2,3,4,5,6,7,8,9,10\}} \{r : b_{r-1} < 10(x_1 - 0.5)(x_2 - 0.5) + \varepsilon \ < b_r\}$$

$$(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}) = \\ (-\infty, -1.75, -1, -0.5, -0.1, 0.1, 0.25, 0.75, 1, 1.75, +\infty)$$

where ε is a random value, normally distributed with zero mean and standard deviation $\sigma = 0.125/2$.

The learning curves obtained for this arrangement are shown in figure 4 (again, for 5 neurons in the hidden layer).

Accuracy dependence on the data dimension. The described experiments in $\mathbb{R}^2$ were repeated for data points in $\mathbb{R}^4$, to evaluate the influence of data dimension on the models' relative performance.

Table 1. Results for 5 classes in $\mathbb{R}^4$ (MER;RMSE;MAE;r_s)

Set size	cNN	pNN	itNN	uNN
100	(0.64;1.60;1.12;0.28)	(0.59;1.14;0.81;0.58)	(0.61;1.39;0.96;0.42)	(0.63;1.58;1.12;0.48)
200	(0.61;1.53;1.05;0.36)	(0.53;0.97;0.66;0.70)	(0.53;1.13;0.75;0.63)	(0.47;1.03;0.64;0.75)
300	(0.57;1.44;0.94;0.44)	(0.51;0.91;0.61;0.75)	(0.45;0.97;0.60;0.73)	(0.39;0.80;0.46;0.84)
400	(0.52;1.36;0.85;0.52)	(0.48;0.86;0.57;0.77)	(0.35;0.76;0.42;0.84)	(0.33;0.68;0.37;0.89)
500	(0.51;1.27;0.80;0.57)	(0.44;0.79;0.50;0.82)	(0.29;0.65;0.32;0.88)	(0.31;0.67;0.35;0.89)

We generated 2000 example points $\mathbf{x} = [x_1\ x_2\ x_3\ x_4]^t$ uniformly at random in the unit square in $\mathbb{R}^4$.

For 5 classes each point was assigned a rank y from the set $\{1, 2, 3, 4, 5\}$, according to

$$y = \min_{r\in\{1,2,3,4,5\}} \{r : b_{r-1} < 1000 \prod_{i=1}^{4} (x_i - 0.5) + \varepsilon \ < b_r\}$$

$$(b_0, b_1, b_2, b_3, b_4) = (-\infty, -2.5, -0.5, 0.5, 3, +\infty)$$

where ε is a random value, normally distributed with zero mean and standard deviation $\sigma = 0.25$.

Finally, for 10 classes the rank was assigned according to the rule

$$y = \min_{r\in\{1,2,3,4,5,6,7,8,9,10\}} \{r : b_{r-1} < 1000 \prod_{i=1}^{4} (x_i - 0.5) + \varepsilon \ < b_r\}$$

$$(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}) = (-\infty, -5, -2.5, -1, -0.4, 0.1, 0.5, 1.1, 3, 6, +\infty)$$

where ε is a random value, normally distributed with zero mean and $\sigma = 0.125$. The results are presented in a tabular form, tables 1 and 2.

Network complexity One final point to make in any comparison of methods regards complexity. The number of learnable parameters for each model is presented in table 3.

Table 2. Results for 10 classes in $\mathbb{R}^4$ (MER;RMSE;MAE;r_s)

Set size	cNN	pNN	itNN	uNN
100	(0.81;3.42;2.54;0.25)	(0.79;2.10;1.60;0.66)	(0.76;2.83;1.99;0.49)	(0.79;3.25;2.36;0.34)
200	(0.78;3.24;2.32;0.33)	(0.74;1.73;1.31;0.80)	(0.68;2.28;1.50;0.66)	(0.65;2.16;1.41;0.72)
300	(0.74;3.04;2.14;0.42)	(0.70;1.55;1.14;0.84)	(0.54;1.37;0.84;0.88)	(0.58;1.74;1.07;0.80)
400	(0.74;3.08;2.11;0.44)	(0.67;1.42;1.03;0.87)	(0.47;1.13;0.66;0.92)	(0.51;1.15;0.71;0.91)
500	(0.70;2.88;1.95;0.49)	(0.63;1.29;0.92;0.89)	(0.42;0.96;0.55;0.94)	(0.47;1.01;0.61;0.93)

4 Conclusion

This study presents a new approach to neural networks training for ordinal data. The main idea is to retain the ordinality of the classes by imposing a parametric model for the output probabilities.

Table 3. Number of parameters for each neural network model

Model	cNN	pNN	itNN	uNN
$\mathbb{R}^2$, $K = 5$	45	21×4	39	21
$\mathbb{R}^2$, $K = 10$	75	21×9	69	21

Model	cNN	pNN	itNN	uNN
$\mathbb{R}^4$, $K = 5$	165	97×4	148	97
$\mathbb{R}^4$, $K = 10$	250	97×9	233	97

The study compares the results of the proposed model with conventional neural network for nominal classes, and two models proposed in the literature specifically for ordinal data.

Simple misclassification, mean absolute error, root mean square error and spearman coefficient are used as measures of performance for all models and used for model comparison. This new method is likely to produce a simpler and more robust classifier, and compares favourably with state-of-the-art methods.

Other directions in future work include the use of similar networks with two or more output units and more flexible models other than the binomial. We think this increased flexibility might improve further the results for more complicated problems, like when we increase the number of classes and of dimensions in our experiments. Another idea which we will consider consists in using these type of models in conjunction with other learning algorithms, like for instance SVMs. We plan also to investigate the performance of this type of models for non ordinal classes.

References

1. Herbrich, R., Graepel, T., Obermayer, K.: Regression models for ordinal data: a machine learning approach. Technical Report TR-99/03, TU Berlin (1999)
2. Shashua, A., Levin, A.: Ranking with large margin principle: Two approaches. In: Neural Information and Processing Systems (NIPS). (2002)
3. Mathieson, M.J.: Ordinal models for neural networks. In Refenes, A., Abu-Mostafa, Y., Moody, J., eds.: Neural Networks in Financial Engineering, World Scientific, Singapore (1995)
4. Frank, E., Hall, M.: A simple approach to ordinal classification. In: Proceedings of the 12th European Conference on Machine Learning. Volume 1. (2001) 145–156
5. Costa, M.: Probabilistic interpretation of feedforward network outputs, with relationships to statistical prediction of ordinal quantities. International Journal Neural Systems **7** (1996) 627–638