Constructing Extensions of Bayesian Classifiers with use of Normalizing Neural Networks

Dominik Ślęzak[1,2], Jakub Wróblewski[2], and Marcin Szczuka[3]

[1] Department of Computer Science, University of Regina
Regina, SK, S4S 0A2, Canada
[2] Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-008 Warsaw, Poland
[3] Institute of Mathematics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland
{slezak,jakubw}@pjwstk.edu.pl, szczuka@mimuw.edu.pl

Abstract. We introduce a new neural network model that generalizes the principles of the Naïve Bayes classification method. It is trained with use of backpropagation-like algorithm, in purpose of obtaining optimal combination of several classifiers. Experimental results are presented.

1 Introduction

In the classification tasks we search for a method that assigns class/decision value to each given example. In doing so we try to incorporate various kinds of information in the this process. We can look at, e.g., the most probable decisions or the entire vectors of probabilistic decision distributions, for some data-derived patterns. One of the best known classification models, in which the decision probabilities pop up quite naturally, is the Naïve Bayes classifier. It uses vectors of the attribute value probabilities subject to particular decision classes.

In this paper we construct a neural network based model that combines, compares and optimizes probabilistic components of the Naïve Bayes classifier. We propose the network architecture, which processes the whole vectors of the decision probabilities. We introduce a concept of normalizing neural network (NNN), where the neuron transition functions accept and return the vectors of real values instead of single values [9]. We provide the foundations, implement, and verify experimentally the NNN (backpropagation-like) learning algorithm.

2 The starting point – the Naïve Bayes classifier

We are dealing with the task of classification [5]. In the beginning we are given a set of examples (training sample) T drawn from some universe X. We assume that every example $u \in X$ is represented by a vector of attribute (feature, measurement) values $a_1(u), ..., a_n(u)$, where $a_i : X \to A_i$, $i = 1, ..., n$. The set A_i is referred to as the *attribute value space*. The examples are also labeled with

the value of decision d, treated as an additional attribute. We denote by $C_k \subset X$ the k-th decision class, i.e., the subset of examples labeled with decision value $k \in \{1, ..., r\}$.

Our goal in the classification problem is to find with some algorithm a hypothesis $h : X \to \{1, ..., r\}$, i.e. a mapping from X onto the set of decision values. Mapping h is often assumed to be highly consistent with the training sample T. In other words, one expects that $d(u)$ should be similar to $h(u)$ for $u \in T$. Mapping h should be also – what is far more important – inductively correct, which means that it should be properly applicable for new, not labeled examples. Consistency with the training data hardly provides the inductive correctness. It is often better to base on less accurate, but less complex models (cf. [6]).

An example of the model, which is approximately consistent with the training cases, is the Naïve Bayes classifier. Although it ignores the attribute dependencies derivable from the data, it is proven to behave in a way very close to optimal in many classification problems [2]. It establishes h on the basis of probabilities $Pr(\cdot)$ estimated from sample T. The estimates are very simple, based on counting the occurrence of the attribute-value patterns in data (cf. [5, 6]). We use them as follows:

$$h(u) = \arg\max_{k \in \{1,...,r\}} \Pr(d = k) \prod_{i=1}^{n} \Pr(a_i = a_i(u) | d = k) \qquad (1)$$

In the next sections, we are also going to refer to an extended version of Naïve Bayes classifier, which is more flexible and less dependent on the "Naïve" assumptions about data independence. Let us present it using (natural) logarithms of probabilities:

$$h(u) = \arg\max_k v_0 \log \Pr(d = k) + \sum_{i=1}^{n} v_i \log(\Pr(a_i = a_i(u) | d = k)) \qquad (2)$$

Weights v_i determine the importance of attributes a_i in classification process. One of contributions in this paper is a method for learning these weights using the backpropagation-like technique for the generalized neural network model.

3 Normalizing neural networks

Normalizing neural networks (NNNs) were introduced in [9]. It is an attempt to devise a mechanism for establishing compound decision distributions with use of an analogon of artificial neural network. Given the list of the classifier components (like e.g. single attributes in the Naïve Bayes method), we put to the input layer neurons responsible for processing their classification preferences. We assume that each component provides the vector of r real values expressing how it is likely to classify each given example to particular decision classes. The difference with respect to the standard artificial network is that now we are going to combine the vectors instead of single real values. The input to the neuron is a collection of vectors and the output is going to be a vector of r real values too. Abbreviation NNN comes from the fact that the weighted sums of vectors undergo normalization by means of the neuron transition functions $\phi : \mathbb{R}^r \to \triangle_{r-1}$ into the $(r-1)$-dimensional simplex of probabilistic distributions.

The structure of NNN with one hidden layer is presented in Figure 1. Vectors $x_i \in \mathbb{R}^r$ correspond to the classifier components for $i = 0, ..., n$. [1] Each j-th neuron in the hidden layer, for $j = 1, ..., m$, takes as input the vector $s_j \in \mathbb{R}^r$ and provides as output the vector $y_j = \phi(s_j)$, where $y_j \in \triangle_{r-1}$ and $\phi : \mathbb{R}^r \to \triangle_{r-1}$. The input to the output neuron is denoted by $t \in \mathbb{R}^r$ and its output takes the form of $h = \phi(t)$, which is the result of the NNN calculations. Vectors $s_1, ..., s_m, t \in \mathbb{R}^r$ are the weighted sums of the outcomes of previous layers, i.e.:

$$t = \sum_{j=1}^{m} w_j y_j \quad \text{and} \quad s_j = \sum_{i=0}^{n} v_{ij} x_i \quad \text{for } j = 1, ..., m \tag{3}$$

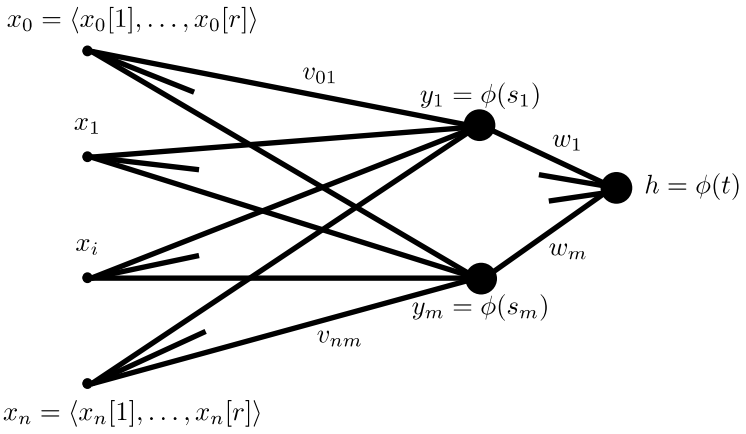

Fig. 1. The architecture of NNN with one hidden layer.

3.1 The NNN transition functions

Transition functions in NNN should be defined in a way that assures both the proper behaviour of calculation procedures and direct interpretation in extended neural network model. They should comply to some conditions, which generalize those formulated for classical transition functions (cf. [3–5]). In NN we use (mostly sigmoidal) monotone functions. In case of NNN, one can say that transition function $\phi : \mathbb{R}^r \to \triangle_{r-1}$ is monotone, if it satisfies the following:

1. Inequality $s[k] > s[l]$ between the input vector coordinates results in inequality $\phi(s)[k] > \phi(s)[l]$ between the output vector coordinates, for $k, l = 1, ..., r$.
2. The increase in the input vector coordinate $s[k]$ results in increase of the corresponding output vector coordinate $\phi(s)[k]$, as well as decrease of the other coordinates $\phi(s)[l]$, for $l \neq k$.

We will use the following monotone function $\phi_\alpha : \mathbb{R}^r \to \triangle_{r-1}$, where parameter $\alpha > 0$ determines the steepness of transition:

$$\phi_\alpha(s) = \left\langle \frac{e^{\alpha s[1]}}{\sum_{l=1}^{r} e^{\alpha s[l]}}, \ldots, \frac{e^{\alpha s[k]}}{\sum_{l=1}^{r} e^{\alpha s[l]}}, \ldots, \frac{e^{\alpha s[r]}}{\sum_{l=1}^{r} e^{\alpha s[l]}} \right\rangle \tag{4}$$

[1] Iteration $i = 0, ..., n$ is consistent with the application described in the next sections.

Behavior of ϕ_α is illustrated in Figure 2, for two decision classes, i.e. $r = 2$.

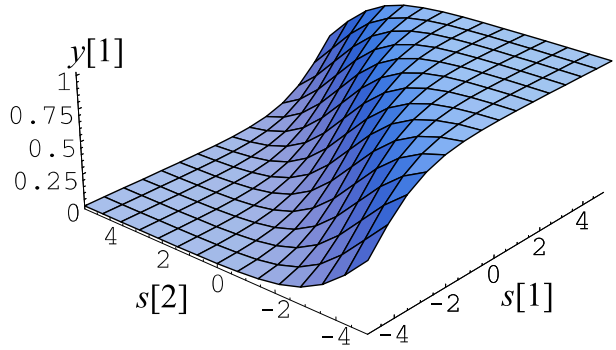

Fig. 2. Coordinate $y[1]$ of function $y = \phi_\alpha(s)$ for $\alpha = 1$ and $s \in \mathbb{R}^2$.

In the next section we generalize the backpropagation algorithm [3, 4, 6] in purpose of tuning the NNN weights. The advantage of using sigmoidal functions in this method is the way of calculating their derivatives. Figure 3 shows that ϕ_α generalizes the behavior of classical sigmoidal functions also at this level.

$$\alpha \cdot \begin{bmatrix} \phi_\alpha(s)[1](1-\phi_\alpha(s)[1]) & \dots & -\phi_\alpha(s)[1]\phi_\alpha(s)[k] & \dots & -\phi_\alpha(s)[1]\phi_\alpha(s)[r] \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ -\phi_\alpha(s)[k]\phi_\alpha(s)[1] & \dots & \phi_\alpha(s)[k](1-\phi_\alpha(s)[k]) & \dots & -\phi_\alpha(s)[k]\phi_\alpha(s)[r] \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ -\phi_\alpha(s)[r]\phi_\alpha(s)[1] & \dots & -\phi_\alpha(s)[r]\phi_\alpha(s)[k] & \dots & \phi_\alpha(s)[r](1-\phi_\alpha(s)[r]) \end{bmatrix}$$

Fig. 3. Derivative matrix $D\phi_\alpha(s)$ for the function $\phi_\alpha : \mathbb{R}^r \to \triangle_{r-1}$ defined by (4).

3.2 Backpropagation in NNN

The key issue is to equip the NNNs with an analogon of backpropagation procedure (cf. [3, 4]). In a nutshell, in the classical neural network model there exists effective method for calculating the error (gradient) ratios used in the weight updates. The error values for the output layer can be easily derived from the differences between the network outcomes and true answers for the training cases. For the hidden layers, the errors are calculated on the basis of linear combination of the error components propagated from the next layer.

The above method can also be applied in case of NNNs. Let us denote by $d = \langle d[1], \dots, d[r] \rangle$ the distribution, which we would like to obtain for a given training example. Let us consider the error function

$$E = \tfrac{1}{2} \textstyle\sum_{k=1}^{r} (h[k] - d[k])^2 \tag{5}$$

where $h = \langle h[1], \dots, h[r] \rangle$ is the output of NNN, as shown in Figure 1. Formula 5 is the normalized Euclidean distance between probabilistic distributions [8]. Its

upper bound equals 1 and it is reached only if the two distributions have ones at different positions (e.g., $\langle 0,0,1,0,0\rangle$, $\langle 0,1,0,0,0\rangle$), i.e. if the output h is totally incorrect in comparison to d for a given training example.

Just like in the classical approach, we use negative gradient of E to tune the network weights. We treat gradient of (5) as the function of the weight vectors:

$$\frac{\partial E}{\partial w_j} = \left\langle h - d \,\middle|\, \frac{\partial h}{\partial w_j} \right\rangle \qquad \text{where} \qquad \frac{\partial h}{\partial w_j} = \left\langle \frac{\partial h[1]}{\partial w_j}, \ldots, \frac{\partial h[r]}{\partial w_j} \right\rangle \tag{6}$$

Let us recall that $h = \phi(t)$ for $\phi : \mathbb{R}^r \to \triangle_{r-1}$ and $t = \sum_{j=1}^{m} w_j y_j$. We obtain

$$\left[\frac{\partial h}{\partial w_j}\right]^T = D\phi(t)\,[y_j]^T \tag{7}$$

where $D\phi$ denotes the derivative matrix of ϕ. Further, let us consider

$$\frac{\partial E}{\partial v_{ij}} = \left\langle h - d \,\middle|\, \frac{\partial h}{\partial v_{ij}} \right\rangle \tag{8}$$

Let us recall that $y_j = \phi(s_j)$ for $s_j = \sum_{i=0}^{n} v_{ij} x_i$, $j = 1, ..., m$. We obtain

$$\left[\frac{\partial h}{\partial v_{ij}}\right]^T = D\phi(t) w_j D\phi(s_j)\,[x_i]^T \tag{9}$$

Formula (9) provides an interpretation similar to that concerning backpropagation, described at the beginning of this subsection. For the consecutive layers, the error vectors are calculated on the basis of the error components propagated from the next layer. The way of calculations of $D\phi(t)$ and $D\phi(s_j)$ depends on the choice of ϕ. In our research we apply function ϕ_α introduced in Section 3.1.

3.3 NNNs for Bayesian classification

Now we show how to implement the concept of NNN in connection with the Naïve Bayes scheme. We consider the NNN architecture described in Figure 1. Given an example $u \in X$ to be classified, for each $i = 1, ..., n$, we put

$$x_i = \langle \log \Pr(a_i = a_i(u)|d = 1), \ldots, \log \Pr(a_i = a_i(u)|d = r)\rangle \tag{10}$$

We also add a special input that corresponds to the bias connection in a classical multilayer, feedforward neural network:

$$x_0 = \langle \log \Pr(d = 1), \ldots, \log \Pr(d = k), \ldots, \log \Pr(d = r)\rangle \tag{11}$$

For each $j = 1, ..., m$, we get the following formula for the coordinates of the input $s_j \in \mathbb{R}^r$ to the j-th neuron in the hidden layer:

$$s_j[k] = v_{0j} \log \Pr(d = k) + \sum_{i=1}^{n} v_{ij} \log \Pr(a_i = a_i(u)|d = k) \tag{12}$$

It corresponds to the extended Naïve Bayes classifier (2). Since the NNN transition function ϕ is assumed to be monotone also in the sense of the properties

presented in Subsection 3.1, we obtain that $\arg\max_k s_j[k] = \arg\max_k y_j[k]$. Hence, if we classify cases using a single neuron with output y_j calculated from inputs (10,11), then the most probable decision class coincides with that given by (2).

Construction of the hidden layer with m neurons enables to learn automatically, using the generalized backpropagation introduced in Section 3.2, the coefficients of the ensemble of the weighted Naïve Bayes classifiers (2) and then – to synthesize them at the level of the output neuron $h = \phi(t)$. Such an approach closely follows the idea of classifier ensemble as introduced in [1].

In the next section we show the experiments with the application of function ϕ_α. Used in the structure from Figure 1, ϕ_α results with the vector coordinates

$$y_j[k] = \frac{\Pr(d=k)^{\alpha v_{0j}} \prod_{i=1}^{n} \Pr(a_i = a_i(u)|d=k)^{\alpha v_{ij}}}{\sum_{l=1}^{r} \Pr(d=l)^{\alpha v_{0j}} \prod_{i=1}^{n} \Pr(a_i = a_i(u)|d=l)^{\alpha v_{ij}}} \tag{13}$$

at the level of the hidden layer, and with the final output coordinates

$$h[k] = \frac{\prod_{j=1}^{m} e^{\alpha w_j y_j[k]}}{\sum_{l=1}^{r} \prod_{j=1}^{m} e^{\alpha w_j y_j[l]}} \tag{14}$$

Vectors y_j can take the form of arbitrary elements of $\triangle_{r-1}$ except its vertices. h can approach a vertex of $\triangle_{r-1}$ only up to the vector of the form $\langle \frac{1}{e^\alpha+r-1}, ..., \frac{e^\alpha}{e^\alpha+r-1}, ..., \frac{1}{e^\alpha+r-1} \rangle$. This is why we decided to learn the NNNs using the reference vectors taking the following form for the training case $u \in T$:

$$d[k] = \begin{cases} \frac{e^\alpha}{e^\alpha+r-1} \text{ iff } d(u) = k \\ \frac{1}{e^\alpha+r-1} \text{ iff } d(u) \neq k \end{cases} \tag{15}$$

Using (15) decreases the risk of overfitting, what was confirmed by experiments.

4 Experiments

We implemented the generalized backpropagation algorithm described in Section 3.2 and applied it to learning the NNN model described in Section 3.3. Several data sets of different size and layout have been selected for this purpose (see Table 1). In most cases the split into training and test samples is inherited with the data set. The main observed value in all experiments is the ratio (percentage) of correctly classified objects in the training and testing sets. The presented results are averaged over several algorithm runs (usually 20 or more). They are compared with the results obtained with use of classical Naïve Bayes classifier. Experiments were also repeated with different choice of the NNN parameters.

The used data sets are taken from [10]. These are standard, well described benchmark data tables for which it is possible to find good reference results (see e.g. [6]). `DNA_small` is derived from the original table by taking only 20 attributes (out of 60), which are known to provide the largest amount of information (cf. [10]); `DNA_large` is the binary version of original data. `Soybean` and `primary_tumor` contain missing values.

The experiments were performed using the NNN network with one hidden layer composed 30-50 neurons in the hidden layer and the neuron transition function ϕ_α for $\alpha = 2$. In case of all data sets it was possible to obtain good classification results on training samples after reasonably small number of iterations of backpropagation algorithm (running about 2000 iterations by default). It confirms the idea of backpropagation presented in Section 3.2.

The results are summarized in Table 1 together with these obtained using the Naïve Bayes (NB) classifier. They are generally close to the best known results obtained for the data sets in discourse (cf. [6, 10]) and are significantly higher than some other classifier synthesis methods [9, 11]. In all cases the NNNs are noticeably better than NB classifiers on the training sets. In three out of four train/test cases (except `DNA_small`) the same may be told about the testing set.

Relatively high classification rate (comparing to the best results known so far) on the `primary_tumor` data is encouraging. It was obtained as an average of several 10-fold cross-validation runs.

Table 1. Description of data sets (number of objects, attributes and decision classes) and summary of experimental results ($\alpha = 2, m = 30$).

Name	train/test obj.	attr./dec.	NNN–test	NB–test
`DNA_small`	2000/1187	20/3	95.34%	95.62%
`DNA_large`	2000/1187	180/3	95.85%	93.34%
`Soybean`	307/376	35/19	91.14%	88.56%
`SAT`	4435/2000	36/6	82.96%	82.35%
`primary_tumor`	339/CV-10	17/21	45.55%	45.86%

The results of the NNN model are parameterized by the number m of neurons in the hidden layer and the value of parameter α. Figure 4 illustrates how the choice of these two parameters influences learning process and classification results. In case of the `DNA_small` data set, there is noticeable tendency suggesting that the larger number of neurons in the hidden layer contributes to the reduction of overfitting effect. These proves to be especially true for data sets with large number of decision values (small representation for each decision class) like `primary_tumor` and `Soybean`, for which the increase (from 30 to 40 and more) in the number of hidden neurons resulted in classification quality improvement. Optimal α value seems to be close to $\alpha = 2$, which can be deduced from Figure 4 and results (not presented) of the wider experiment, comparing the α values between 1.5 and 5.0. More massive experiments are obviously needed in purpose of finding optimal configurations of the NNN parameters.

5 Conclusions

We introduced a concept of normalizing neural network (NNN), where the probabilistic vectors appear both on inputs and output of every neuron. Description of the NNN architecture and the foundations of its learning algorithm

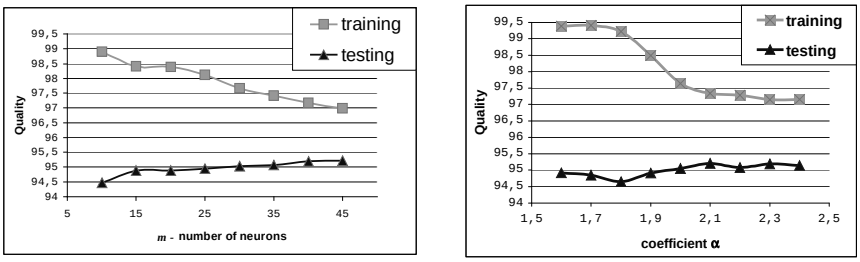

Fig. 4. Classification quality on `DNA_small` data with changing m and α.

(backpropagation) was presented as well. It effectively extends classical Bayesian approach to classification problems. Presented experimental results demonstrate the ability of NNN to both learn and generalize information. In several cases, Bayesian NNN classifier outperforms classical Naïve Bayes method.

Acknowledgements

Partially supported by the Polish State Committee for Scientific Research (KBN) grant No. 8T11C02519. The first author partially supported by the Research Center of PJIIT.

References

1. Dietterich, T.: Machine learning research: four current directions. *AI Magazine* **18/4** (1997) pp. 97–136.
2. Domingos, P., Pazzani, M.: On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. Machine Learning **29** (1997) pp. 103–130.
3. Hecht-Nielsen, R.: Neurocomputing, Addison-Wesley, New York (1990).
4. le Cun, Y.: A theoretical framework for backpropagation. In: Neural Networks – concepts and theory, IEEE Computer Society Press, Los Alamitos (1992).
5. Mitchell T.M., Machine Learning, McGraw Hill, Boston (1997)
6. Michie D., Spiegelhalter D.J., Taylor C.C.(eds), Machine Learning, Neural and Statistical Classification, Ellis Horwood, London (1994) `http://www.amsta.leeds.ac.uk/~charles/statlog/`
7. Ślęzak, D.: Normalized decision functions and measures for inconsistent decision tables analysis. *Fundamenta Informaticae* **44/3** (2000), pp. 291–319.
8. Ślęzak, D.: Various approaches to reasoning with frequency-based decision reducts: a survey. In: Polkowski, L., Tsumoto, S., Lin, T.Y. (eds): Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems. Physica Verlag, Heidelberg (2000) pp. 235–285.
9. Ślęzak D., Wróblewski J., Szczuka M., Neural Network Architecture for Synthesis of the Probabilistic Rule Based Classifiers, *ENTCS* **82.4**, Elsevier, (2003) `http://www.elsevier.nl/locate/entcs/volume82.html`
10. UCI Repository of ML databases, University of California, Irvine, (1998) `http://www.ics.uci.edu/~mlearn/MLRepository.html`.
11. Wróblewski J.: Ensembles of classifiers based on approximate reducts. Fundamenta Informaticae **47** (3,4), IOS Press (2001) 351–360.