Bruno Apolloni
Simone Bassis
Anna Esposito
Francesco Carlo Morabito
Editors

SMART INNOVATION, SYSTEMS AND TECHNOLOGIES ■ 19

Neural Nets and Surroundings

22nd Italian Workshop on Neural Nets, WIRN 2012, May 17–19, Vietri sul Mare, Salerno, Italy

SIREN

Springer

Smart Innovation, Systems and Technologies 19

Editors-in-Chief

Prof. Robert J. Howlett
KES International
PO Box 2115
Shoreham-by-sea
BN43 9AF
UK
E-mail: rjhowlett@kesinternational.org

Dr. Lakhmi C. Jain
Adjunct Professor
University of Canberra
ACT 2601
Australia
and
University of South Australia
Adelaide
South Australia SA 5095
Australia
E-mail: Lakhmi.jain@unisa.edu.au

For further volumes:
http://www.springer.com/series/8767

Bruno Apolloni, Simone Bassis, Anna Esposito,
and Francesco Carlo Morabito (Eds.)

Neural Nets and Surroundings

22nd Italian Workshop on Neural Nets,
WIRN 2012, May 17–19, Vietri sul Mare,
Salerno, Italy

Springer

Editors
Prof. Bruno Apolloni
Department of Computer Science
University of Milano
Milano
Italy

Dr. Simone Bassis
Department of Computer Science
University of Milano
Milano
Italy

Prof. Anna Esposito
Department of Psychology
Second University of Naples
Caserta
Italy

and

Institute for Advanced Scientific
Studies (IIASS)
Vietri sul Mare Salerno
Italy

Prof. Francesco Carlo Morabito
Department of Mechanics and Materials
Mediterranea University of Reggio Calabria
Reggio Calabria
Italy

ISSN 2190-3018 e-ISSN 2190-3026
ISBN 978-3-642-35466-3 e-ISBN 978-3-642-35467-0
DOI 10.1007/978-3-642-35467-0
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2012953656

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume collects a selection of contributions which has been presented at the 22nd Italian Workshop on Neural Networks, the yearly meeting of the Italian Society for Neural Networks (SIREN). The conference was held in Italy, Vietri sul Mare (Salerno), during May 17–19, 2012. The annual meeting of SIREN is sponsored by International Neural Network Society (INNS), European Neural Network Society (ENNS) and IEEE Computational Intelligence Society (CIS).

The workshop, and thus this book, is organized in three main components, two special sessions and a group of regular sessions featuring different aspects and point of views of artificial neural networks and natural intelligence, also including applications of present compelling interest.

More than 60 papers were presented at the Workshop, and most of them are reported here. The review process has been carried out in two steps, one before and one after the workshop in order to meet Publisher's requirements. The selection of the papers was made through peer-review process, where each submission was evaluated by at least two reviewers. The submitted papers were authored by peer scholars from different countries (the Italian component was anyway preponderant). The acceptance rate is thus high also because most of the attendees are involved in SIREN research and organization activities for more than 20 years. In addition to regular papers, the technical program featured keynote plenary lectures by some worldwide renowned scientist (Soo Young Lee, South Korea; Ganesh K. Venayagamoorthy, USA; Jacek Zurada, USA; Günther Palm, Germany; Alessandro Vinciarelli, UK; Danilo Mandic, UK). One of the two special sessions was supported by the EU-sponsored COST Action 2102 that closed his work on February 2011 even though the Members of the Action are still networking and collaborating in scientific activities.

The first Special Session explored the new frontiers and challenges in Smart Grid research and proposed a proficient discussion table for scientists joining the WIRN conference, whose expertise typically cover the research fields addressed in Smart Grid technology, as electrical and electronic engineering, computational intelligence, digital signal processing and telecommunications. The Session included two invited contributions and seven regular ones. The Session was particularly relevant because it introduced

some aspects of neural network applications not commonly known at the community in a field of growing interest.

The second Special Session was titled Computational Intelligence in Emotional or Affective Systems and was given in honour of John Taylor, the Editor-in-chief of the journal Neural Networks recently died. The Session featured two keynote lectures and 10 regular contributions. Computational Intelligence (CI) methods have shown great capabilities in modelling, prediction, and recognition tasks and a mature degree of understanding has been achieved in many application areas, in particular in complex multimodal systems supporting human-machine or human-human interaction. At the same time, the emotional issue has recently gained increasing attention in such complex systems due to its relevance in most common human tasks (like cognitive processes, perception, learning, communication and even "rational" decision-making) and therefore is highly relevant for the goal of human-like interaction with machines. The real challenge is taking advantage of the emotional characterization of humans to make the computer interfacing with them more natural and therefore useful. The scope of the session was to assess to what extent and how sophisticated computational intelligence tools developed so far might support the multidisciplinary research on the characterization of an appropriate system reaction to human emotions and expression in interactive scenarios.

We would like to thank all of the special sessions organizers, namely: Stefano Squartini, Rosario Carbone, Michele Scarpiniti, Francesco Piazza, Aurelio Uncini, Anna Esposito, Günther Palm.

The organization of an International Conference gathers for the efforts of several people involved. We would like to express our gratitude to everyone that has cooperate to the organization, by offering their commitment, energy and spare time to make this event a successful one.

May 2012

Bruno Apolloni
Simone Bassis
Anna Esposito
Francesco Carlo Morabito

Organization

WIRN 2012 is organized by the Italian Society of Neural Networks (SIREN) in co-operation with the International Institute for Advanced Scientific Studies (IIASS) of Vietri S/M (Italy).

Executive Committee

Bruno Apolloni	University of Milano, Italy
Simone Bassis	University of Milano, Italy
Anna Esposito	University Federico II of Napoli, Italy
Francesco Masulli	University of Genova, Italy
Francesco Carlo Morabito	University Mediterranea of Reggio Calabria, Italy
Francesco Palmieri	Second University of Napoli, Italy
Eros Pasero	Polytechnic of Torino, Italy
Stefano Squartini	Polytechnic University of Marche, Italy
Roberto Tagliaferri	University of Salerno, Italy
Aurelio Uncini	University "La Sapienza" of Roma, Italy
Salvatore Vitabile	University of Palermo, Italy

Program Committee

Conference Chair

Francesco Carlo Morabito	University Mediterranea of Reggio Calabria, Italy

Conference Co-Chair

Simone Bassis	University of Milan, Italy

Program Chair

Bruno Apolloni	University of Milan, Italy

Organizing Chair

Anna Esposito	Second University of Napoli, Italy

Special Tracks

Anna Esposito	Second University of Napoli, Italy
Stefano Squartini	Polytechnic University of Marche, Italy

Referees

G. Albano	S. Funari	M. Re
B. Apolloni	C. Furlanello	A. Rizzi
S. Bassis	G. L. Galliani	P. M. Ros
A. Borghese	S. Giove	S. Rovetta
F. Camastra	G. Ippoliti	A. Rozza
W. Capraro	F. La Foresta	M. Russolillo
R. Carbone	G. Lombardi	S. Scarpetta
M. Cardin	M. Lucchese	M. Scarpiniti
A. Ciaramella	D. Malchiodi	R. Serra
C. Claudio	U. Maniscalco	G. Spagnuolo
D. Comminiello	C. Marco	S. Squartini
V. d'Amato	F. Masulli	A. Staiano
R. de Rosa	L. Menconi	A. Uncini
F. Epifania	A. Micheli	G. Valentini
A. M. Esposito	F. C. Morabito	L. Valerio
A. Esposito	G. Palm	M. Villani
M. Faundez-Zanuy	F. Palmieri	S. Vitabile
A. Filisetti	E. Pasero	Q. Wei
M. Frasca	F. Piazza	A. Zippo

Sponsoring Institutions

International Institute for Advanced Scientific Studies (IIASS) of Vietri S/M (Italy)
Department of Psychology, Second University of Napoli (Italy)
Provincia di Salerno (Italy)
Comune di Vietri sul Mare, Salerno (Italy)

Contents

Part I: Algorithms

Part II: Signal Processing

Part III: Applications

Part V: Special Session on "Computational Intelligence in Emotional or Affective Systems"

Part I

Algorithms

Probability Learning and Soft Quantization in Bayesian Factor Graphs

Francesco A.N. Palmieri and Alberto Cavallo

Dipartimento di Ingegneria Industriale e dell'Informazione
Seconda Universitá di Napoli (SUN)
via Roma 29, 81031 Aversa (CE), Italy
{francesco.palmieri,alberto.cavallo}@unina2.it

Abstract. We focus on learning the probability matrix for discrete random variables in factor graphs. We review the problem and its variational approximation and, via entropic priors, we show that soft quantization can be included in a probabilistically-consistent fashion in a factor graph that learns the mutual relationship among the variables involved. The framework is explained with reference the "Tipper" example and the results of a Matlab simulation are included.

Keywords: Machine Learning, Factor Graphs, Bayesian Methods.

1 Introduction

Probability propagation on graphs is a very promising emerging paradigm for building intelligent signal processing systems [12]. Algorithms and applications are under development in many areas of research that range from communication and coding to signal processing and control. However, full use and development of artificial intelligence systems that operate with probability propagation techniques require refinements on a number of critical issues. Some of these are: 1. Propagation in graphs with cycles [1]; 2. Parameter learning [8]; 3. Graph-structure learning [13]; 4. Propagation and learning in hybrid graphs with both continuous and discrete variables; etc. In this paper we focus on learning the probability matrix in discrete-variable factor graphs [7] [6] pointing to a connection to variational learning [5] [3] [2] [18] [19]. We apply the idea to a generic block where the whole probability matrix is learned from examples. Recent development on inference based on entropic priors [15] [14] allows the introduction of soft quantization within the Bayesian graph framework much like in fuzzy logic [17]. Entropic priors allow to translate some of the successful heuristics typical of the fuzzy framework, into a probabilistically-consistent Bayesian learning paradigm on factor graphs. Soft logic formulated within standard probability theory [10] coupled with belief propagating on factor graphs represents a very promising framework to bring to a higher cognitive level many of the current signal processing problems. In our formulation we use factor graphs in Forney's normal

B. Apolloni et al. (Eds.): *Neural Nets and Surroundings*, SIST 19, pp. 3–10.
DOI: 10.1007/978-3-642-35467-0_1

form [11], because they are easier to handle in comparison to more traditional Bayesian graphs [16].

In this paper we first review the problem of learning the probability matrix pointing to a connection with variational message passing. Then we briefly introduce soft quantization with entropic priors and finally we apply the ideas to the well-known Tipper example. The results of a simulation show how this framework implements a very natural dynamic merge of inference and learning.

2 Learning the Probability Matrix

Probabilistic inference in factor graphs via message propagation is a relatively mature technique, at least in graphs with no cycles, when the conditional probability functions that make up the model are known [12]. A much harder problem is learning the model parameters on line, i.e. performing inference and learning at the same time. To focus on the specifics of this issue we start with the simplest (non trivial) factor graph of Figure 1 that models N independent realizations of two random variables $X \in \mathcal{X} = \{\xi_1, ..., \xi_d\}$ and $Y \in \mathcal{Y} = \{\eta_1, ..., \eta_m\}$. The variables are discrete and take values in the two alphabets $\mathcal{X}$ and $\mathcal{Y}$ and are related via the unknown conditional probability matrix

$$P(Y|X\Theta) = \begin{pmatrix} p(\eta_1|\xi_1) & ... & p(\eta_m|\xi_1) \\ p(\eta_1|\xi_2) & ... & p(\eta_m|\xi_2) \\ . & ... & . \\ p(\eta_1|\xi_d) & ... & p(\eta_m|\xi_d) \end{pmatrix} = \Theta = \begin{pmatrix} \Theta_{11} & ... & \Theta_{1m} \\ \Theta_{21} & ... & \Theta_{2m} \\ . & ... & . \\ \Theta_{d1} & ... & \Theta_{dm} \end{pmatrix}, \tag{1}$$

with $0 \leq \Theta_{ij} \leq 1$, $i = 1, ..., d$, $j = 1, ..., m$; $\sum_{j=1}^{m} \Theta_{ij} = 1$, $i = 1, ..., d$. The unknown parameters make up the matrix $\Theta \in \mathcal{T}$, where $\mathcal{T}$ denotes the set of all $d \times m$ stochastic matrices. Since the structure of Figure 1 may be part of a more complex network, we assume that information on $X[n]$ and $Y[n]$ is available in

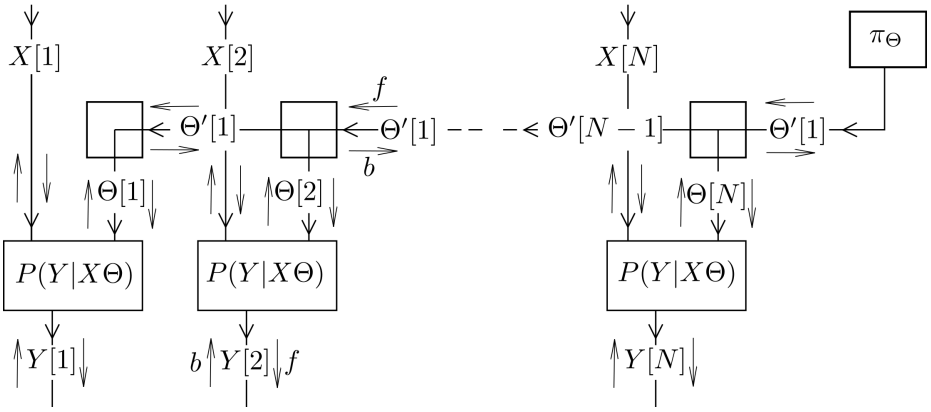

Fig. 1. The factor graph for N independent realizations of $(X[n], Y[n])$

soft form via forward and backward distributions $f_{X[n]}(x)$, $b_{X[n]}(x)$, $f_{Y[n]}(y)$ and $b_{Y[n]}(y)$, with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Also information about matrix Θ is carried by forward and backward messages $f_{\Theta[n]}(\theta)$ and $b_{\Theta[n]}(\theta)$ which are matrix functions. These messages are related to each other via marginalization as

$f_{Y[n]}(y) \propto \int_{\theta\in\mathcal{T}} \sum_{x\in\mathcal{X}} P(y|x\theta) f_{X[n]}(x) f_{\Theta[n]}(\theta) d\theta$;
$b_{X[n]}(x) \propto \int_{\theta\in\mathcal{T}} \sum_{y\in\mathcal{Y}} P(y|x\theta) b_{Y[n]}(y) f_{\Theta[n]}(\theta) d\theta$;
$b_{\Theta[n]}(\theta) \propto \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} P(y|x\theta) b_{Y[n]}(y) f_{X[n]}(x)$.

As usual in factor graphs, the notation $\propto$ means that the expressions are distributions except for proper normalization. The complete model is hybrid because $X[n]$ and $Y[n]$ are discrete and Θ is continuous and multi-dimensional. In a more compact matrix representation, forward and backward messages for $X[n]$ and $Y[n]$ are the column vectors

$\mathbf{f}_{X[n]} = (f_{X[n]}(\xi_1), ..., f_{X[n]}(\xi_d))^T$; $\mathbf{b}_{X[n]} = (b_{X[n]}(\xi_1), ..., b_{X[n]}(\xi_d))^T$;
$\mathbf{f}_{Y[n]} = (f_{Y[n]}(\eta_1), ..., f_{Y[n]}(\eta_m))^T$; $\mathbf{b}_{Y[n]} = (b_{Y[n]}(\eta_1), ..., b_{Y[n]}(\eta_m))^T$.

Therefore we can write

$\mathbf{f}_{Y[n]} \propto \int_{\theta\in\mathcal{T}} \theta^T \mathbf{f}_{X[n]} f_{\Theta[n]}(\theta) d\theta = F^T_{\theta[n]} \mathbf{f}_{X[n]}$;
$\mathbf{b}_{X[n]} \propto \int_{\theta\in\mathcal{T}} \theta \mathbf{b}_{Y[n]} f_{\Theta[n]}(\theta) d\theta = F_{\theta[n]} \mathbf{b}_{Y[n]}$,

where $F_{\theta[n]} = \int_{\theta\in\mathcal{T}} \theta f_{\Theta[n]}(\theta) d\theta$ is the mean forward matrix for $\Theta[n]$. The backward message for $\Theta[n]$ is the matrix function

$$b_{\Theta[n]}(\theta) \propto \mathbf{f}^T_{X[n]} \theta \mathbf{b}_{Y[n]} = \mathbf{f}^T_{X[n]} \begin{pmatrix} \theta_{11} & ... & \theta_{1m} \\ \theta_{21} & ... & \theta_{2m} \\ . & ... & . \\ \theta_{d1} & ... & \theta_{dm} \end{pmatrix} \mathbf{b}_{Y[n]}. \tag{2}$$

Messages for $\Theta[n]$ and $\Theta'[n]$ in the other branches are formally the result of the product rule $f_{\Theta[n]}(\theta) \propto f_{\Theta'[n]}(\theta) b_{\Theta'[n-1]}(\theta)$; $b_{\Theta'[n]}(\theta) \propto b_{\Theta[n]}(\theta) b_{\Theta'[n-1]}(\theta)$; $f_{\Theta'[n]}(\theta) \propto b_{\Theta[n+1]}(\theta) f_{\Theta'[n+1]}(\theta)$. Each message is a product of the type

$$\mu_\Theta(\theta) \propto \prod_l \mathbf{f}^T_{X[l]} \begin{pmatrix} \theta_{11} & ... & \theta_{1m} \\ \theta_{21} & ... & \theta_{2m} \\ . & ... & . \\ \theta_{d1} & ... & \theta_{dm} \end{pmatrix} \mathbf{b}_{Y[l]} = \prod_l \sum_{i=1}^{d} \sum_{j=1}^{m} b_{Y[l]}(\eta_j) f_{X[l]}(\xi_i) \theta_{ij} \tag{3}$$

If variables $X[n]$ and $Y[n]$ of block n are *instantiated*, i.e. forward and backward messages are delta functions, $f_{X[n]}(x) = \delta(x - \xi_i)$, $b_{Y[n]}(x) = \delta(y - \eta_j)$, backward information from block n is simply $b_{\Theta[n]}(\theta) \propto \theta_{ij}$. If also all variables from all n are instantiated, information exchanged among the blocks (except possibly for the prior on Θ) are exactly products of Dirichlet distributions

$$\mu_\Theta(\theta) \propto \prod_{i=1}^{d} \prod_{j=1}^{m} \theta_{ij}^{n_{ij}} \propto \prod_{i=1}^{d} Dir(\theta_{i1}, ..., \theta_{im}; n_{i1} + 1, ..., n_{im} + 1), \tag{4}$$

where n_{ij} are the integer numbers that represent the cumulative counts of the occurrences of pair (i, j) (*hard scores*). Unfortunately, in the general case we are

interested in with forward and backward messages carrying soft information, expression (3) becomes intractable. Hence we resort to a variational approximation [5] [3] [18] for $b_{\Theta[n]}(\theta)$ that gives

$$\begin{aligned} b^V_{\Theta[n]}(\theta) &\propto e^{\sum_{i=1}^{d}\sum_{j=1}^{m} b_{Y[n]}(\eta_j) f_{X[n]}(\xi_i) \log \theta_{ij}} = \prod_{i=1}^{d}\prod_{j=1}^{m} \theta_{ij}^{b_{Y[n]}(\eta_j) f_{X[n]}(\xi_i)} \\ &\propto \prod_{i=1}^{d} Dir(\theta_{i1}, ..., \theta_{im}; f_{X[n]}(\xi_i) b_{Y[n]}(\eta_1) + 1, ..., f_{X[n]}(\xi_i) b_{Y[n]}(\eta_m) + 1), \end{aligned} \tag{5}$$

which is again the product of d Dirichlet distributions. This is particularly interesting because the Dirichlet distribution, sometimes used as an assumption [7] [19], is exactly the variational approximation. Assuming that also the prior distribution π_Θ is a product of Dirichlet functions

$\pi_\Theta \propto \prod_{i=1}^{d} Dir(\theta_{i1}, ..., \theta_{im}; \alpha_{i1} + 1, ..., \alpha_{im} + 1)$.

A generic message in the upper branches has the form

$$\begin{aligned} \mu_\Theta \propto \prod_{i=1}^{d} Dir(&\theta_{i1}, ..., \theta_{im}; \\ &\alpha_{i1} + \textstyle\sum_l f_{X[l]}(\xi_i) b_{Y[l]}(\eta_1) + 1, ..., \alpha_{im} + \sum_l f_{X[l]}(\xi_i) b_{Y[l]}(\eta_m) + 1). \end{aligned} \tag{6}$$

A priori knowledge about the rule that maps X into Y can also be easily included in the coefficients of π_Θ. The exponential form for the variational approximation suggests that matrix variables $\Theta[n]$ and $\Theta'[n]$ could be replaced with *soft score* matrix variables $O[n]$ and $O'[n]$. Backward message from block n becomes matrix $b_{O[n]} = \mathbf{f}_{X[n]} \mathbf{b}^T_{Y[n]}$. Also all messages in the upper branches become $d \times m$ matrices with combination rules $f_{O[n]} = f_{O'[n]} + b_{O'[n-1]}$; $b_{O'[n]} = b_{O[n]} + b_{O'[n-1]}$; $f_{O'[n]} = b_{O[n+1]} + f_{O'[n+1]}$. Forward and backward messages for $Y[n]$ and $X[n]$ are respectively $\mathbf{f}_{Y[n]} \propto F^T_{O[n]} \mathbf{f}_{X[n]}$; $\mathbf{b}_{X[n]} \propto F_{O[n]} \mathbf{b}_{Y[n]}$,, where $F_{O[n]}$ is the row-normalized version of $f_{O[n]}$. Note that these propagation rules represent the learning steps for Θ as inference and learning happen at the same time. Recall that the various stages in the graph represent time-unfolded versions of the same block. Mode details and proofs will be reported in a longer paper.

3 Soft Quantization

Manipulation of discrete quantities in machine learning, also when the problem involves continuous variables, may be particularly handy, because a priori qualitative information can be more easily injected into the system. Fuzzy methods [17] have shown great success in merging soft knowledge with hard functions especially in control [9]. In [15] we have shown how the use entropic priors in the Bayesian framework allowing the introduction of soft membership information in a way that is consistent within standard probability theory. This is a crucial step to allow soft quantization and coherent use of probability propagation for inference and learning in systems that contain both continuous and discrete variables.

Figure 2 shows a quantization scheme for a continuous variable S_a. All the likelihoods are triangular, complementary and centered on the M nodes $\xi_1, ..., \xi_M$. Denoting the triangular function on a, b, c with $\Lambda(s_a; a, b, c)$, the M pdfs are

$$\begin{aligned}&\{\tfrac{2}{\xi_2-\xi_1}\Lambda(s_a;\xi_1,\xi_1,\xi_2), \tfrac{2}{\xi_3-\xi_1}\Lambda(s_a;\xi_1,\xi_2,\xi_3),\\ &..., \tfrac{2}{\xi_M-\xi_{M-2}}\Lambda(s_a;\xi_{M-2},\xi_{M-1},\xi_M), \tfrac{2}{\xi_M-\xi_{M-1}}\Lambda(s_a;\xi_{M-1},\xi_M,\xi_M)\},\end{aligned} \tag{7}$$

and are shown in Figure 2(a). The differential entropy [4] of $\Lambda(s_a; a, b, c)$ is easily computed to be $h(S_a) = \frac{1}{2} + \log\frac{c-a}{2}$. With entropic priors $\pi_i \propto e^{h(S_a|i)}$ [15], the prior-likelihood products, become equivalent to a set of functions with same height as in Figure 2(b). We recall that entropic priors are the distribution that maximize the joint entropy $H(S_a, S)$ for fixed likelihoods $(p_{S_a}(s_a|1), ..., p_{S_a}(s_a|M))$ [15]. The node distribution can be chosen according to the data points density, but the complementarity of the likelihoods guaratees that no information is lost after soft quantization. Figure 2(b) shows also how this kind of soft quantization can be drawn as a generative factor graph model that can be inserted into a larger factor graph. The backward message for S_a is a data point $b_{S_a}(s_a) = \delta(s_a - s_0)$. The backward message for S^1 in vector notation is $\mathbf{b}_{S^1} = (p_{S_a}(s_0|1), ..., p_{S_a}(s_0|M))^T$ that after combination with entropic priors becomes $\mathbf{f}_{S^2} = (p_{S_a}(s_0|1)\pi_1, ..., p_{S_a}(s_0|M)\pi_M)^T$. The soft quantization model satisfies a property of perfect recostruction because if $\mathbf{b}_{S^2} = \mathbf{b}_{S^1}$, we have $\mathbf{f}_{S^1} = \mathbf{f}_{S^2}$ and $f_{S_a}(s_a) = \delta(s_a - \mathbf{f}_{S^1}^T(\xi_1, ..., \xi_M)^T) = \delta(s_a - s_0)$ (lossless dequantization). More details about soft quantization with entropic priors will be reported in a longer paper elsewhere.

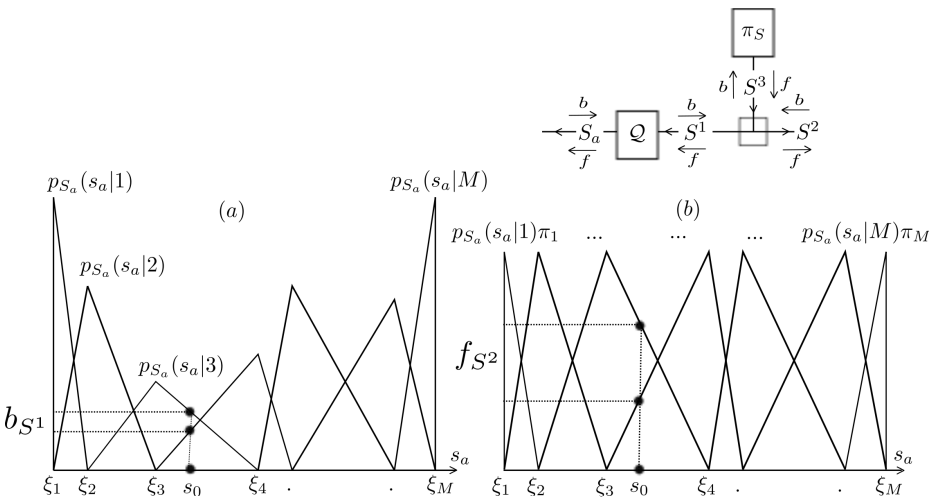

Fig. 2. Soft quantization on nodes $\{\xi_1, ..., \xi_M\}$. (a) The triangular likelihoods; (b) The entropic priors-likelihoods products.

4 The Tipper Example

In this paper we report some experiments with the variational learning rules of Section 2 and with the soft quantization scheme of Section 3 on the well-known "Tipper" example. In this problems there are three continuous variables: S_a (Service), F_a (Food) and T_a (Tip). The Tipper example, often used as a teaching example in control classes (there is a Matlab demo available in the Fuzzy Control Toolbox), is a typical case of mapping between two input variables (Service and Food) and a final one (Tip). In the fuzzy framework is also very easy to include soft rules and various design constraints. Our objective is here to traslate this typical approach into a probabilistically-consistent Bayesian framework. The underlying factor graphs shown in Figure 4, in which messages travel back and forth, allows simultaneous inference and learning with inputs and outputs that become essentially indistinguishable.

Even though a priori soft-logic rules can be easily included as contraints in the prior block π_Θ, we have assumed here no prior knowledge about the variables S_a, F_a and T_a. We have simply presented 50 realizations of the triplet as f_{S_a}, f_{F_a} and f_{T_a} and let the system learn (simulations with combinations of soft rules and examples will be reported elsewhere). The triplets were obtained from a blind run of the Matlab demo. Forward and backward messages carry information in various parts of the system and inferences can also be made backward on Service and/or Food from Tip.

The graph structure assumes that variables Service and Food are mutually independent and that the $N = 50$ realizations are also statistically independent. The three analog variables S_a, F_a and T_a are soft quantized from ranges $[0-10][0-10][5-25]$ with $M = 6$ uniformly spaced nodes each into the three discrete variables S, F and T. Entropic priors are imposed in π_S, π_F and π_T. The 36×6 matrix of conditional probabilities $P(T|SF\Theta)$ is learned via message propagations with the variational algorithm described in Section 2. The simulations let the messages propagate 300 steps which is enough to cover the graph diameter. The graph is clearly a tree and convergence is guaranteed. Figure 4 shows the comparison of forward and backward information at each stage n. The thre plots show the comparison of the actual value of each variable, as carried by the backward input message, with the value provided by the rest of the system, as carried by the forward output message that uses all the other inputs after learning and propagation. Note that learning and inference is all done at the same time since information about the parameter θ are also carried by travelling messages. The simulation is self-contained and implements our best use of the data because the inference, say on $T_a[n]$, is based on all the examples except the one on $T_a[n]$. This is because $f_{\Theta[n]}$ does not contain information coming from $b_{\Theta[n]}$. Hence each stage n uses a slightly different estimate for $P(T|SF\Theta)$ because the nth examples is automatically excluded. Therefore in inferring $T_a[n]$, values of $S_a[n]$ and $F_a[n]$ are used for inference, but not for learning. The same considerations apply to inferences on $S_a[n]$ and $F_a[n]$.

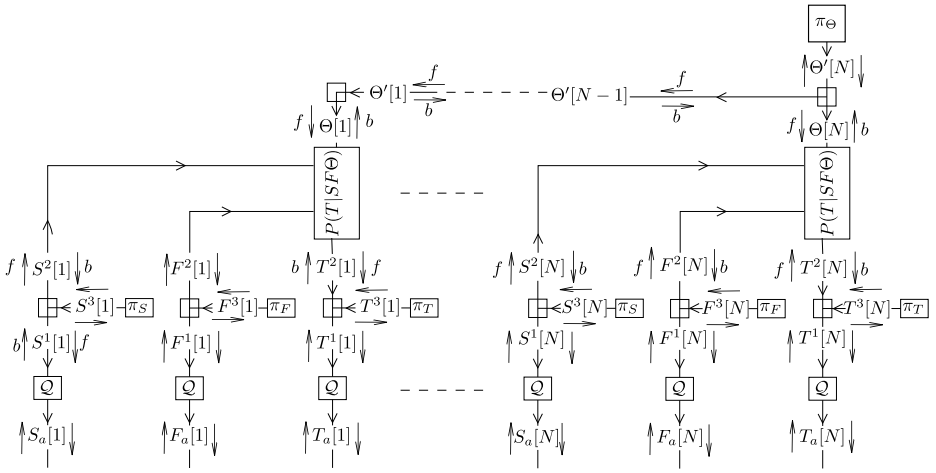

Fig. 3. The factor graph for the Tipper example

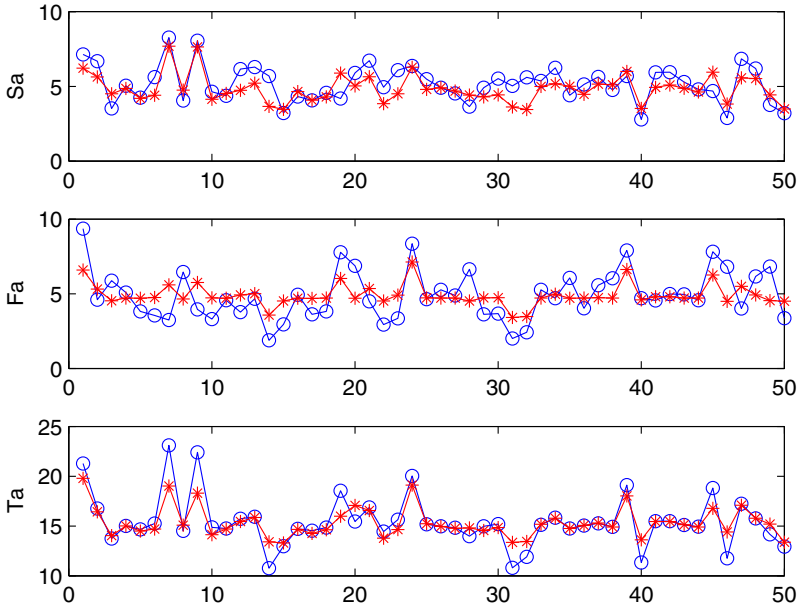

Fig. 4. Comparison of forward (inference) (*) and backward (true) (o) values for the three variables in the Tipper example

Conclusions

In this work we have reported partial results for a successful Bayesian paradigm that implements via message propagation on a factor graph simultanaous inference and learning. By means of an example, we have also proposed a quantization scheme, that via the introduction of entropic priors, allows us to build a probabilistically-consistent graph that can adapted with belief propagation. More work will be devoted to further understanding of the adaptation rules and on the inclusion of soft-logic contraints.

References

1. Graphical models emerge. new connections betweeen machine learning and signal processing. Signal Processing Magazine, 27(6) (2010)
2. Beal, M.J.: Variational algorithms for approximate bayesian inference. Ph.D. thesis, University of London (2003)
3. Beal, M.J., Ghahramani, Z.: Variational bayesian learning of directed graphical models with hidden variables. Bayesian Analysis 1, 1–44 (2004)
4. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley (2006)
5. Dauwels, J.: On variational message passing in factor graphs. In: ISIT2007, Nice, France, June 24-June 29 (2007)
6. Ghahramani, Z.: Unsupervised Learning. Springer (2004)
7. Heckerman, D.: A tutorial on learning with bayesian networks. Tech. Rep. MSR-TR-95-06, Microsoft Research (1996); March 1995 (Revised November 1996)
8. Dauwels, J., Eckford, A., Loeliger, S.K., Expectation, H.A.: xpectation maximization as message passing–part i: Principles and gaussian messages. arXiv:0910, 1–14 (2009); Submitted to IEEE Tr. on Information Theory
9. Jantzen, J.: Foundations of Fuzzy Control. Wiley (2007)
10. Jaynes, E.T.: Probability Theory: The Logic of Science. Cambridge University Press (2003)
11. Loeliger, H.A.: An introduction to factor graphs. IEEE Signal Processing Magazine 21(1), 28–41 (2004)
12. Loeliger, H.A., Dauwels, J., Hu, J., Korl, S., Ping, L., Kschischang, F.: The factor graph approach to model-based signal processing. Proceedings of the IEEE 95(6), 1295–1322 (2007)
13. Choi, M.J., Tan, V.Y.F., Anandkumar, A., Willsky, A.S.: Learning latent tree graphical models. Journal of Machine Learning Research 12, 1771–1812 (2011)
14. Palmieri, F.A.N., Ciuonzo, D.: Entropic priors for short-term stochastic process classification. In: 14th Int. Conf. on Information Fusion, Chicago, IL (2011)
15. Palmieri, F.A.N., Ciuonzo, D.: Objective priors from maximum entropy in data classification. In: Information Fusion (2012), doi:10.1016/j.inffus.2012.01.012
16. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc., San Francisco (1988)
17. Novak, V., Perfilieva, I., Mockor, J.: Mathematical Principles of Fuzzy Logic. Kluwer Academic Press (1999)
18. Winn, J., Bishop, C.M.: Variational message passing. Journal of Machine Learning Research 6, 661–694 (2005)
19. Winn, J.M.: Variational message passing and its applications. Ph.D. thesis, University of Cambridge (2004)

Rival-Penalized Competitive Clustering: A Study and Comparison

Alberto Borghese and Wiliam Capraro

Applied Intelligent Systems Laboratory, Dept. of Comupter Science, University of Milano
borghese@di.unimi.it, wiliam.capraro@studenti.unimi.it

Abstract. A major recurring problem in exploratory phases of data mining is the task of finding the number of clusters in a dataset. In this paper we illustrate a variant of the competitive clustering method which introduces a rival penalization mechanism, and show how it can be used to solve such problem. Additionally, we present some tests aimed at comparing the performance of our rival-penalized technique with other classical procedures.

1 Introduction

The term central clustering refers to a family of clustering algorithms that are based on moving a set of points, referred to as prototypes, inside the data space until their position minimizes a certain cost function, representing a measure of the goodness by which the prototypes represent the data points.

In the literature, approaches based on soft-clustering, like fuzzy c-means [14], Neural-Gas [13,4] or Self-Organizing Maps (SOM) [6,16], and competitive learning [5,15,12], have been proposed to partition a given dataset into a predefined number of clusters, each one represented by a prototype usually corresponding to the cluster centroid. All these algorithms suffer from several issues, most notably, the optimal value of the cost function is rarely reached. Only stochastic optimization [10], that is extremely costly, or a careful initialization of the prototypes allow escaping local minima. Although a few attempts have been proposed to derive a robust initialization (e.g. [11]), there seem to be no universal and reliable way to proceed, and some prototypes typically get stuck during the clustering process. These prototypes are referred to as "dead units" [13] and affect the proper operation of the algorithm and the quality of the result. As a consequence, the general approach is to repeat the clusterization process several times with different, random initializations of the prototypes, so as to allow the algorithm to escape from local minima from time to time.

A slightly different task is to find the number of clusters in a dataset (e.g. [3]). This is indeed a frequent problem in exploratory phases of data mining, and a straightforward approach is to adopt a parameterized version of a clustering algorithm using the desired number of data clusters K as parameter and to try a different clusterization for each possible value of the parameter. Subsequently, the best result would be chosen based on some validity measure or index.

However, it is impractical to run several random initializations of one algorithm for each possible values of its parameter, especially when the latter spans a wide interval of

B. Apolloni et al. (Eds.): *Neural Nets and Surroundings*, SIST 19, pp. 11–20.
DOI: 10.1007/978-3-642-35467-0_2

values. Alternatively, in some cases it is possible to exploit the intrinsic characteristics of some algorithms to produce dead units (e.g. [6]), to facilitate the search for a good solution.

Recently, in the framework of central clustering, a different approach has been proposed for moving the prototypes. The idea is that, while the winning prototype is attracted by the closest data point, other prototypes are moved in the opposite direction. This mechanism, known as rival-penalization [5,15], is somehow similar to the BCM model proposed by Bienenstock, Cooper and Munro [9] in a typical Hebbian learning fashion.

Rival-penalization clustering has been overlooked in the past. In this paper we present the results of some tests we performed, aimed at comparing the rival-penalization approach with classical clustering techniques, namely standard competitive learning and SOM. The ability of rival-penalization of discovering the proper number of clusters in a given dataset is analysed and discussed. Specifically, we show that, by introducing the rival penalization mechanism into a competitive learning setting, results comparable with soft-clustering can be achieved. Moreover, the number of clusters can be discovered in a robust and reliable way.

2 Algorithms

We'll consider a collection of N d-dimensional observations, $\{\xi_j\}$. Goal of clustering algorithms is to assign each observation to one of K clusters Ψ_i, according to a similarity measure with the other elements in the same cluster. Each cluster is represented by its centroid, ψ_i, which is also a point in $\mathbb{R}^d$.

The following subsections give a quick coverage of the algorithms we employed.

2.1 Competitive Learning and Rival Penalization

Competitive learning (CL) is an effective tool for data clustering, widely applied in a variety of signal processing problems such as data compression, classification, adaptive noise cancelation, image retrieval and image processing [2].

For the purposes of this contribution, a feed-forward neural network with a single layer consisting of K output units is used to achieve a K-cluster data partitioning. Each unit represents a cluster centroid ψ_i.

The training of the network proceeds as follows. At each iteration, each data point ξ is presented in turn to the network and a winning unit, w, is elected. This is the prototype whose Euclidean distance from the point is minimum:

$$w = \arg\min_i \|\xi - \psi_i\|. \tag{1}$$

Subsequently, the position of the winning unit is updated towards the data point using the following updating rule

$$\psi_{wj} = \psi_{wj} + \eta_{(t)}(\xi - \psi_w) \tag{2}$$

where j denotes a component of the prototype vector and $\eta_{(t)}$ is a learning rate parameter whose value decays as a function of the time t.

In a pure competitive learning setting, only the winning unit is updated. The procedure is repeated multiple times for each data point, until the prototypes converge to their final position—i.e. when the maximum difference in the position of any centroid in two successive iterations is smaller than a fixed tolerance ε, or when a maximum number of iterations is reached.

The prototypes are initialized using the "Forgy" approach [1]—i.e. K of the available data points are randomly chosen to serve as cluster prototypes. In this context, this is enough to guarantee that no dead unit will ever appear, as every prototype shall win the competition for at least one data point, that is, the prototype itself.

The rival-penalized competitive learning (RPCL) algorithm improves on the pure competitive learning approach by introducing a rival penalization mechanism, as proposed in [5] and [15]. With this approach, not only the position of the winning unit is updated towards the input vector, but additionally the position of its rival unit is updated in the opposite direction.

In order to find the winning unit and its rival, a relative winning frequency is introduced, which keeps track of how many times each unit happens to win a competition for some input vector. The relative winning frequency for unit i is defined as

$$\gamma_i = \frac{s_i}{\sum_{j=1}^{K} s_j} \tag{3}$$

where s_i is the number of times unit i was declared winner in the past. When $\sum_{j=1}^{K} s_j = 0$—i.e. initially, then $\gamma_i = 1$ in order to give every prototype a fair chance to win.

The winning unit w for an input vector ξ is now given by

$$w = \arg\min_i \gamma_i \|\xi - \psi_i\|. \tag{4}$$

Notice how the parameter γ_i acts as a "conscience" for the unit—if the unit has won too often in the past, its chances to win the competition for the current data point are reduced accordingly. Moreover, for each input vector ξ, the rival penalized competitive learning algorithm computes not only the winning unit w, but also a second winning unit, referred to as the rival, defined by

$$r = \arg\min_i \gamma_i \|\xi - \psi_i\|, \quad i \neq w. \tag{5}$$

Equation 2 is used to update both the winner and its rival. The latter, however, moves away its centroid from the input point with a de-learning rate β, which is related to η by

$$\beta_{(t)} = -c\eta_{(t)}\gamma_r \tag{6}$$

where γ_r is the relative winning frequency of the rival and $c = 1/10$ is a predefined constant. Unlike the implementation of [5], here β depends on both the learning rate η and the winning frequency γ_r, so that the rival is dynamically penalized according to γ_r even for constant η (which is not the case anyway).

In contrast to the CL algorithm, here a "Forgy" initialization of the prototypes is not enough to guarantee dead unit avoidance. In fact, even if the prototypes are initialized using the input data points, depending on the de-learning rate β, a rival unit may incur

considerable modification in the value of its prototype, and thus it can fail to win the competition even for the input data point to which it had been initialized.

What is interesting with this approach is that, as reported in [5], if the learning rate η is chosen to be at least one order of magnitude larger than β, then the adequate number of output clusters will be automatically found. In other words, assuming that the actual number of clusters is unknown and that the number of units K is chosen greater than the cluster number, the prototype vectors will converge towards the centroids of the actual clusters with few of them overlapping in space. In our implementation, this condition holds in each iteration as $c = 1/10$. In each iteration, the RPCL algorithm pushes away the rival, thus allowing for faster convergence, and invalidates extra prototypes by eventually making their cluster empty. Hence, the RPCL algorithm is believed to be able to perform appropriate clustering without knowing the cluster number.

2.2 SOM

The limitation of considering only one data point at a time in competitive learning has been overcome by soft-clustering approaches [2] in which the position of all the prototypes is updated for each data point. Among these approaches, Self-Organizing Maps (SOMs) represent an excellent tool in exploratory phases of data mining. They project the input space onto prototypes in a low-dimensional regular grid that can be effectively used to visualize and explore properties of the data. The SOM consists of a regular, one- or two-dimensional grid of units, with each unit i represented by its prototype vector ψ_i. Additionally, each unit i is assigned a place in the output grid, represented by its coordinates $r_i = (x_i, y_i)$, and the units are logically linked to adjacent ones by a neighborhood relation. During training, data points lying near each other in the input space are mapped to nearby units in the output hyperplane. Thus, the SOM can be regarded as a topology-preserving tool for mapping the input space onto the output grid.

The SOM is trained iteratively. At each training step, a data point ξ is randomly chosen from the input data set, and the distance between ξ and all the prototype vectors is computed. Subsequently, all prototype vectors are updated, each proportionally to the distance of the corresponding unit from the winning unit in the output grid:

$$\psi_{ij} = \psi_{ij} + \eta_{(t)} \Lambda(i,w)(\xi - \psi_i). \tag{7}$$

In the hereabove equation $\Lambda(i,w)$ denotes the value of the neighborhood function between unit i and the winning unit w, as given by

$$\Lambda(i,w) = \exp\left(-\frac{\|r_i - r_w\|^2}{2\sigma^2}\right) \tag{8}$$

where the parameter σ defines the radius of the neighborhood. $\Lambda(i,w)$ therefore defines a region of influence for the prototype w. Notice that the value of Λ is exactly 1 when $i = w$, and decreases as the distance of the prototype from the other data points increases. Also, it is useful to adjust the radius as well as the learning rate at each iteration, so that the influence region of a prototype decays with time as a function of σ and η.

In this work, we are mainly concerned with the SOM's ability to perform appropriate clustering of a given data set. Thus, only SOMs with one-dimensional output arrays are actually used. As stated in [16], this configuration is expected to produce better results as compared to the 2-dimensional grid configuration. This is due to the fact that the "tension" exerted in each unit by the neighboring units is much higher in the second configuration, and such a tension limits the plasticity of the SOM to adapt to the particular distributions of the dataset.

3 Experimental Setting and Test Results

In order to test the algorithms, we have generated a specific dataset containing 250 3-d data points distributed over 5 non-overlapping clusters. It has been generated by perturbating the centroid of each cluster with a Gaussian distribution with mean value 0 and variance 1.

Since the resulting clusterization depends strongly on the initialization of prototypes, it is essential that each algorithm be tested several times with different initializations. For our tests, 60 "Forgy" initializations (which we'll refer to as trials) have been generated and evaluated for each algorithm. This should be enough to overcome random fluctuations.

As to the tests we conducted, they can be divided into two types. In a first type thereof—we call it type-A experiment—we focused on a specific algorithm and tried to partition our dataset varying the cluster number from $K = 2$ to $K = 10$. (And for each K, 60 trials have been performed as described before.) On the other hand, in type-B experiments we tested 60 trials of an algorithm with a fixed value of K but varying the parameters of the algorithm instead.

In our tests, the validity of the resulting clusterization is evaluated by means of quality indexes, and the number of iterations required by the algorithm to converge was also measured. The quality indexes used are the Davies-Bouldin (DB) index and the mean quadratic error (MQE). The mean quadratic error is simply the ratio of the sum of all the squared distances of each data point from its cluster prototype to the total number of data points:

$$MQE = \frac{\sum_{i=1}^{K} \sum_{\xi \in \Psi_i} \|\xi - \psi_i\|^2}{\sum_{i=1}^{K} |\Psi_i|}. \tag{9}$$

The Davies-Bouldin index is defined as

$$DB = \frac{1}{K} \sum_{i=1}^{K} \max_{j \neq i} \left\{ \frac{S_i + S_j}{\|\psi_i - \psi_j\|} \right\} \tag{10}$$

where S_i is the within i-th cluster scatter, as given by

$$S_i = \sqrt{\sum_{\xi \in \Psi_i} \frac{\|\xi - \psi_i\|^2}{|\Psi_i|}}. \tag{11}$$

For a detailed review of these and other cluster validity measures see [8]. Notice that it is geometrically plausible to seek clusters that have minimum within-cluster scatter

and maximum between-class separation, so the number of clusters $\bar{K}$ that minimizes the Davies-Bouldin index can be reasonably taken as the optimal value of K. As reported in [8], for well-separated clusters, the Davies-Bouldin index is expected to decrease monotonically as K increases until the correct number of clusters is achieved.

In all the tests conducted, we fixed a tolerance of $\varepsilon = 0.001$ and the maximum number of iterations was set to 500. It should be enough for the algorithms to produce good clusterizations given the time-decay rule for η adopted, which is

$$\eta_{(t)} = \exp\left(-\frac{t}{50}\right) \cdot \eta_0 \tag{12}$$

where $\eta_0 = 0.1$ is the initial value and $t = 0, 1, \ldots$ is the iteration number.

In the remaining of this section we illustrate the results of our tests.

3.1 Competitive Learning

To begin with, we measured the effectiveness of the standard CL algorithm in partitioning our dataset by running a type-A test. The results can be used throughout the rest of this work as a reference for the other algorithms. For each value of K, Table 1 reports the Davies-Bouldin index and the quadratic error for the best outcome out of the 60 trials of the algorithm, along with the number of iterations performed.

As Table 1 shows, the algorithm succeeds in discovering the correct number of clusters: the DB index takes on its optimal value for $K = 5$.

As expected, the mean quadratic error is a decreasing function of the number of clusters (indeed one expects the within-cluster variance to decrease in this case), and hence it does not convey any useful information on the goodness of the result.

As a downside, the CL algorithm takes a considerable number of iterations to converge, as in each iteration only the winning unit is moved towards the current data point by a small, η-dependent, fraction of the distance. Moreover, in order to discover the optimal value of the parameter K, every possible value has to be investigated and the result evaluated. The average number of iterations is a decreasing function of K, which is rather obvious since, for small K, we expect the amount of modification in the position of each centroid as a function of the data points to be higher in each iteration as compared to when K is large.

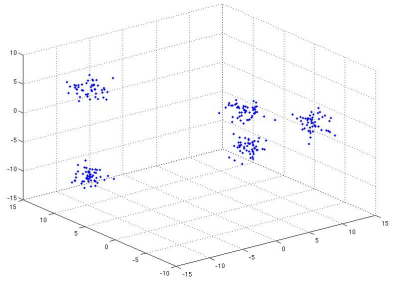

Fig. 1. Scatterplot of the dataset

Table 1. Type-A test results for CL

K	DB	MQE	it
2	0.712	49.642	344
3	0.396	27.360	344
4	0.429	10.583	328
5	0.298	2.816	298
6	0.690	2.648	298
7	0.751	2.603	298
8	0.768	2.540	298
9	0.746	2.435	298
10	0.809	10.197	328

Table 2. Type-A test results for SOM

(a) $\sigma = 0.5$

K	nD	DB	MQE	it
2	0	0.7123	49.6421	346
3	0	0.5779	32.8641	345
4	0	0.4291	10.5827	330
5	0	0.2983	2.8161	300
6	1	0.2983	2.8161	300
7	2	0.2983	2.8161	300
8	3	0.2983	2.8161	300
9	4	0.2983	2.8161	300
10	5	0.2983	2.8161	300

(b) $\sigma = 1$

K	nD	DB	MQE	it
2	0	0.7123	49.6421	346
3	0	0.5779	32.8641	345
4	0	0.4291	10.5827	330
5	0	0.2983	2.8161	300
6	1	0.2983	2.8161	300
7	2	0.2983	2.8161	300
8	3	0.2983	2.8161	300
9	3	0.7244	2.6374	294
10	3	1.0228	2.4724	294

(c) $\sigma = 1.5$

K	nD	DB	MQE	it
2	0	0.7123	49.6421	346
3	0	0.5779	32.8641	345
4	0	0.4291	10.5827	330
5	0	0.2983	2.8161	300
6	1	0.2983	2.8161	300
7	2	0.2983	2.8161	300
8	3	0.2983	2.8161	300
9	2	1.0505	2.5142	294
10	3	1.0710	2.5077	294

We have also investigated the role of the learning rate η in the learning process. To this end, a type-B test has been performed in which a value of $K = 5$ has been fixed and η takes on some values in the range (0.2-0.02). As before, 60 trials of the algorithm have been tested for each value of η, and the best outcome is considered. We do not report the results for space issues. We report, however, that the initial learning rate seems to play no crucial role in the learning process, since for every value of η the best value obtained for the DB index is the same as that of Table 1.

3.2 SOM

As our second test, we have investigated the performance of a SOM in achieving proper clusterizations of the dataset. As before we ran a type-A test using a 1-dimensional output array of units, as we are not interested in the spatial organization of the resulting cluster centroids. The distance of each prototype from its neighbor prototypes on the output array has been set arbitrarily to 1, which is also the initial value for the radius of the neighborhood σ. The general idea is that, by exploiting the SOM's inherent ability to produce dead units, it is possible to avoid testing every possible value of the parameter K provided it is chosen larger than the actual number of clusters. Results are reported in Table 2b, where nD represents the number of dead units and again each line represents the best outcome for all the 60 initializations, according to the Davies-Bouldin index.

As the table implies, the minimum of the DB index is obtained for values of K in the range between $K = 5$ and $K = 8$. The values reported confirm the ability of the SOM to produce good clusterizations of the dataset, with values comparable with that of the competitive learning approach for all values of the parameter K. The results also advocate the thesis that the SOM is able to invalidate extra clusters and discover the correct number of clusters if the parameter K is chosen in a neighborhood of its optimal value.

We did not expect the quality of the resulting clusterization to change considerably as a function of the initial learning rate η, so we did not conduct any test in this respect. It is interesting, however, to observe the behavior of the algorithm when the initial radius is enlarged or restricted. Tables 2a and 2c also show the result of type-A tests

Table 3. Type-B test results for RPCL

(a) $\eta = 0.1$

#	K	DB	MQE	it
1	5	0.2973	2.8166	298
2	8	14.1375	436.7141	499
3	5	0.2983	2.8161	298
4	7	0.8392	3.4301	337
5	5	0.2977	2.8162	298
6	5	0.2972	2.8165	298
7	5	0.2977	2.8162	298
8	5	0.2972	2.8168	298
9	8	1.2113	127.1700	485
10	5	0.2980	2.8161	298
11	9	1.9186	283.6699	486
12	5	0.2981	2.8161	298
13	9	1.5104	2.2541	285
14	10	1.4525	32.4955	438
15	8	1.8814	119.3174	478
16	9	3.2711	1289.8495	499
17	5	0.2981	2.8161	298
18	10	1.3259	2.0547	283
19	5	0.2964	2.8175	298
20	7	0.8711	2.4896	298
21	5	0.2961	2.8180	298
22	8	1.2890	187.3231	478
23	5	0.2983	2.8161	298
24	10	1.4116	2.0938	287
25	8	1.7381	87.3881	429
26	5	0.2959	2.8205	298
27	8	1.4844	786.6143	499
28	9	2.2929	931.2280	499
29	5	0.2983	2.8161	298
30	5	0.2967	2.8174	298
31	5	0.2970	2.8167	298
32	5	0.2975	2.8164	298
33	5	0.2983	2.8161	298
34	5	0.2959	2.8183	298
35	7	0.8889	2.5653	292
36	5	0.2983	2.8161	298
37	5	0.2973	2.8165	298
38	5	0.2979	2.8161	298
39	5	0.2973	2.8164	298
40	7	1.4684	206.7596	491
41	5	0.2975	2.8164	298
42	5	0.2959	2.8188	298
43	9	1.0727	2.5370	297
44	5	0.2975	2.8163	298
45	9	1.2583	69.7734	465
46	9	1.2287	150.8042	494
47	8	1.3926	246.4092	499
48	5	0.2959	2.8183	298
49	7	2.1666	222.3385	487
50	5	0.2983	2.8161	298
51	5	0.2958	2.8192	298
52	9	1.4211	51.5298	445
53	5	0.2976	2.8162	298
54	5	0.2970	2.8168	298
55	9	1.3163	988.7258	499
56	5	0.2982	2.8161	298
57	9	1.2233	2.2424	292
58	8	1.7596	428.0865	499
59	5	0.2978	2.8161	298
60	5	0.2982	2.8161	298

(b) $\eta = 0.3$

#	K	DB	MQE	it
1	5	0.3312	3.6632	366
2	5	0.2955	2.8196	353
3	5	0.2978	2.8162	353
4	6	1.0019	594.3351	499
5	10	3.2478	12092.0839	499
6	9	2.2782	27149.0575	499
7	7	2.2396	407.3565	499
8	1	n.a.	n.a.	499
9	9	2.0219	14926.2644	499
10	8	1.0686	111.9457	499
11	1	n.a.	n.a.	494
12	1	n.a.	n.a.	496
13	8	3.3781	736.2310	499
14	6	0.6007	2.6212	353
15	9	1.1331	2.4473	363
16	8	1.1660	254.0439	499
17	9	4.7230	68607.6355	499
18	1	n.a.	n.a.	499
19	1	n.a.	n.a.	486
20	1	n.a.	n.a.	459
21	5	0.2948	2.8272	353
22	1	n.a.	n.a.	499
23	8	1.4005	22.1895	463
24	7	0.8376	2.4683	345
25	9	1.1667	2.7230	374
26	1	n.a.	n.a.	495
27	9	1.1134	6.0580	410
28	8	1.1568	355.1591	499
29	6	1.4096	91.4646	443
30	7	1.1727	7076.8766	499
31	9	1.2331	22.9518	479
32	9	1.4621	599.8796	499
33	9	1.2037	2.8699	384
34	8	0.9067	2.8137	374
35	1	n.a.	n.a.	470
36	10	3.6183	406.4338	499
37	7	2.0509	729.5399	499
38	8	0.9922	2.6372	352
39	9	1.2467	32.5807	487
40	1	n.a.	n.a.	467
41	1	n.a.	n.a.	499
42	8	1.8052	5702.3187	499
43	9	1.0551	2.6401	363
44	1	n.a.	n.a.	499
45	9	1.6941	22.7070	461
46	1	n.a.	n.a.	499
47	7	1.7933	317.9108	499
48	9	1.1090	3.3827	386
49	9	1.1030	3.8625	396
50	7	1.0535	1192.2263	499
51	10	1.9694	1978.7778	499
52	5	0.2982	2.8161	353
53	8	5.0827	783.8008	499
54	7	1.9763	3376261.9184	499
55	9	1.2069	39.0237	497
56	5	0.2983	2.8161	353
57	5	0.2894	2.9133	354
58	7	1.0936	701.7895	499
59	1	n.a.	n.a.	499
60	8	2.7499	133.5437	499

(c) $\eta = 0.5$

#	K	DB	MQE	it
1	6	0.9961	1743.7830	499
2	7	1.1205	529.9871	499
3	1	n.a.	n.a.	499
4	9	1.0335	8.1133	440
5	8	2.0086	1834.3202	499
6	6	0.6528	2.9607	411
7	7	1.7469	27172.1130	499
8	1	n.a.	n.a.	499
9	1	n.a.	n.a.	499
10	1	n.a.	n.a.	499
11	3	1.3013	84.3499	454
12	7	1.0529	659.0174	499
13	8	1.5026	353.7500	499
14	5	0.2983	2.8161	379
15	7	1.3130	663.7360	499
16	1	n.a.	n.a.	499
17	4	0.9470	45.7815	432
18	9	8.7524	200.3968	499
19	8	1.3188	35309.9092	499
20	8	2.2503	712.0748	499
21	5	0.2983	2.8161	379
22	1	n.a.	n.a.	499
23	1	n.a.	n.a.	499
24	1	n.a.	n.a.	499
25	7	0.7805	3.0149	401
26	6	1.0100	270.6201	499
27	1	n.a.	n.a.	454
28	7	2.1072	12438.7022	499
29	1	n.a.	n.a.	499
30	7	1.0933	3285294.0383	499
31	8	0.9186	3.4217	410
32	9	2.0005	4802.1818	499
33	7	0.9568	2.8659	402
34	8	2.0441	15709.4822	499
35	9	8.6214	631.4390	499
36	7	1.0028	502.4863	499
37	1	n.a.	n.a.	499
38	1	n.a.	n.a.	499
39	7	2.0152	147.8561	482
40	2	0.7452	106.3408	445
41	8	2.7086	503.6955	499
42	6	2.6311	41958.7468	499
43	8	2.3926	889.5130	499
44	6	0.9915	904.8396	499
45	5	1.6410	564.5119	499
46	1	n.a.	n.a.	499
47	1	n.a.	n.a.	499
48	7	1.1248	208.1925	499
49	1	n.a.	n.a.	499
50	1	n.a.	n.a.	499
51	2	0.8228	102.4013	450
52	7	0.8945	2.7744	378
53	7	1.0473	1132.3572	499
54	1	n.a.	n.a.	499
55	6	1.0058	582.4626	499
56	1	n.a.	n.a.	499
57	10	6.9798	282791.2331	499
58	1	n.a.	n.a.	469
59	6	0.9993	464.6136	499
60	1	n.a.	n.a.	499

for $\sigma = 1.5$ and $\sigma = 0.5$: results show a general tendency of the radius to influence the ability of the SOM to kill extra units—this ability seems to increase as the radius of the neighborhood narrows.

3.3 Competitive Clustering with Rival Penalization

Lastly, we have analysed the performance of our RPCL implementation in discovering the correct number of clusters. As suggested in [5] we have chosen a number of clusters, $K = 10$, larger than the true number of clusters. Recall that the de-learning rate β is always at least one order of magnitude smaller than η. Once again we considered 60 "Forgy" initialization of the algorithm, and ran three type-B tests using different learning rates, namely $\eta = 0.1$, $\eta = 0.3$ and $\eta = 0.5$. Results are reported in Tables 3a through 3c, where we have indicated with k the number of partitions in the resulting clusterization, and with # the trial number. For each value of η, we have highlighted the best result according to the Davies-Bouldin index.

Results reveal the following aspects. As expected, the algorithm exhibits a strong ability to invalidate extra units. Such ability appears to be stronger compared to the SOM, as suggested by the fact that the algorithm has always been able to obtain correct clusterizations of the dataset—i.e. five clusters, associated with extremely good values for the Davies-Bouldin index.

Moreover, this ability is only partially affected by the choice of the initial learning rate η—as Table 3 implies, the RPCL algorithm has been able to obtain correct partitionings of the dataset in the 56.67% of the trials for $\eta = 0.1$, 11.67% for $\eta = 0.3$ and 5% for $\eta = 0.5$. In this respect, a higher learning rate does augment the ability of the rival units to move in the data space, and hence the ability of the algorithm to invalidate extra clusters[1], but the success or failure of the algorithm is ultimately due to the goodness of the initialization of the prototypes. As a consequence, we expect the algorithm to succeed independently of the learning rate as long as the number of initializations tested is large enough.

4 Discussion and Conclusion

In conclusion, all the algorithms tested work reasonably well and produce good clusterizations of the dataset. However, if the main task is to make use of one such methods to discover the number of clusters in a given dataset, the rival-penalized competitive learning approach appears to be more robust and practical, since it exhibits a remarkable ability to invalidate extra units—i.e. clusters—depending on the prototype initialization, provided the number of initializations tested is large enough. Hence, this method comes in handy when the number of clusters of a dataset is unknown.

If the number of clusters is not known exactly but it is known to belong to a range of a few possible values, then the self-organizing map can also guess the correct number of clusters and yield a good clusterization, providing multiple, random initializations are tested and the prototypes are drawn from the input dataset. However, the performance

[1] Note that, in some cases, this ability has reached a point in which the algorithm produced a 1-cluster partitioning, for which the Davies-Bouldin index is structurally not defined and the quadratic error loses its significance.

of the SOM exhibits a strong dependency on the value of the parameters, and finding the optimal values for the radius and the step-length can be a challenging task. In this respect, the rival-penalized competitive learning approach is to be preferred over the SOM. Moreover, the SOM involves greater workload compared to the rival-penalized method or the standard competitive learning method, and hence its use in a context where the spatial organization of the output units is of little or no interest appears to be questionable.

Lastly, if the number of clusters is known in advance and the goal is simply to produce a clusterization of the dataset, the basic competitive learning algorithm works fairly well and is less susceptible to the initialization of the prototypes and does not suffer from the dead unit problem. Additionally, it has the highest performance-to-cost ratio, although the number of clusters and the learning rate can play a crucial role in this respect.

References

1. Forgy, E.W.: Cluster analysis of multivariate data: efficiency vs interpretability of classifications. Biometrics 21, 768–769 (1965)
2. Xu, R., Wunsch, D.: Survey of Clustering Algorithms. IEEE Transactions on Neural Networks 16, 645–678 (2005)
3. Sugar, C.A., James, G.M.: Finding the number of clusters in a dataset: an information theoretic approach. Journal of the American Statistical Association 98, 750–763 (2003)
4. Martinetz, T.M., Berkovich, S.G., Schulten, K.J.: "Neural Gas" network for vector quantization and its application to time-series prediction. IEEE Transactions on Neural Networks 4, 558–569 (1993)
5. Budura, G., Botoca, C., Miclău, N.: Competitive Learning Algorithms for Data Clustering. Electronics and Energetics 19, 261–269 (2006)
6. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics 43, 59–69 (1982)
7. Erwin, E., Obermayer, A., Schulten, K.: Self-organizing maps: ordering, convergence, properties and energy functions. Biological Cybernetics 67, 47–55 (1992)
8. Bezdek, J.C., Pal, N.R.: Some new indexes of cluster validity. IEEE Transactions on Systems, Man, and Cybernetics 28, 301–315 (1998)
9. Bienenstock, E.L., Cooper, L.N., Munro, P.W.: Neuroscience 2, 32–48 (1982)
10. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by Simulated Annealing. Science 220, 671–680 (1983)
11. Ferrari, S., Ferrigno, G., Piuri, V., Borghese, N.A.: Reducing and Filtering Point Clouds with Enhanced Vector Quantization. IEEE Transactions on Neural Networks 18, 161–177 (2007)
12. Uchiyama, T., Arbib, M.A.: An algorithm for competitive learning in clustering problems. Pattern Recognition 27, 1415–1421 (1994)
13. Fritzke, B.: A growing Neural Gas network learns topologies. Advances in Neural Information Processing Systems 7, 625–632 (1995)
14. Bezdek, J.: Pattern recognition with fuzzy objective function algorithms. Plenum Press, New York (1981)
15. King, I., Lau, T.-K.: Non-hierarchical Clustering with Rival Penalized Competitive Learning for Information Retrieval. In: Perner, P., Petrou, M. (eds.) MLDM 1999. LNCS (LNAI), vol. 1715, pp. 116–130. Springer, Heidelberg (1999)
16. Bação, F., Lobo, V., Painho, M.: Self-organizing Maps as Substitutes for K-Means Clustering. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2005. LNCS, vol. 3516, pp. 476–483. Springer, Heidelberg (2005)

An Interpretation of the Boundary Movement Method for Imbalanced Dataset Classification Based on Data Quality

Dario Malchiodi

Dipartimento di Informatica, Università degli Studi di Milano, Italy
malchiodi@di.unimi.it

Abstract. This paper describes how the classification of imbalanced datasets through support vector machines using the boundary movement method can be easily explained in terms of a cost-sensitive learning algorithm characterized by giving each example a cost in function of its class. Moreover, it is shown that under this interpretation the boundary movement is measured in terms of the squared norm of the separator's slopes in feature space, thus providing practical insights in order to properly choose the boundary surface shift.

1 Introduction

Many real-world problems cannot be solved directly because no formulation attempting to process them, even in an inefficient way according to the computational complexity theory [1], is known. Despite of this fact, such problem *instances* can be tackled *indirectly* because given a set of candidate solutions it is relatively easy to assert which ones will actually solve the problem and which won't. Refer to these two types of candidate solutions as *positive* and *negative examples*, respectively, and consider for instance the problem of face recognition [2]: while there is no agreement on the mechanisms underlying the way our brain recognizes a given face, it is fairly easy to state with good confidence if the face of known person, say Mr. White, appears on an image. Thus it is possible to build a *dataset* gathering several (positive and negative) examples. Such dataset can be processed by a *machine learning* algorithm having the aim of inferring an automatic classifier able to answer future queries related to the recognition of Mr. White.

Datasets such as those involved in face recognition often share the property of being *imbalanced*, as finding images not representing the face of a given person is far easier than finding images of that person. Thus examples from one class are represented in a sensibly lower quantity than those of the remaining classes. Many real-world instances of the classification problem fall into this special category, such as for instance happens in the fields of fraud detection or fingerprinting recognition. Indeed, it is expectable that when considering, e.g., payment requests issued by a credit card, only a small fraction of these will actually describe a fraud. Likewise, when comparing a latent fingerprint

B. Apolloni et al. (Eds.): *Neural Nets and Surroundings*, SIST 19, pp. 21–27.
DOI: 10.1007/978-3-642-35467-0_3

found on a crime scene against a forensic database, most of the available records will not refer to the found fingerprint.

The problem of imbalanced classification has been considered by various perspectives, including those:

- rebalancing positive and negative examples' representatives, either undersampling the over-represented class [3], oversampling the under-represented one [4], or performing both operations [5];
- post-processing of the classifiers in output of general-purpose learning algorithms [6];
- focusing only on the under-represented class, to be learnt through unsupervised techniques [7], so that the over-represented class is indirectly inferred through complementation of the under-represented one;
- tailoring learning algorithms for the special case of imbalanced data [8].

This paper focus on the relations between the second and the fourth approach when they are applied to the widely used classification methodology based on support vector machines [9]. When using these tools, the classifier inference is reformulated in terms of the solution of a constrained optimization problem depending on the original examples, and this solution is in turn translated into a hyperplane separating the images of examples in a suitable feature space. In particular, it is shown that the post-processing technique proposed in [6], consisting in shifting the threshold of previously mentioned hyperplane, constitutes the special case of a cost-sensitive algorithm for support vector classification (that is, an algorithm assigning distinct costs to each example and using these costs in order to build the optimization problem to be solved [10]). This equivalence will suggest a practical rule in order to set the shift for the hyperplane learnt through support vector classification in order to account for the amount of class imbalance.

The paper is organized as follows: Sect. 2 briefly reviews the employed cost-sensitive methodology tailored for support vector classification, while Sect. 3 describes the application of this methodology to the problem of imbalanced classification. Finally, Sect. 4 is devoted to concluding remarks.

2 Cost-Sensitive Classification through Support Vector Machines

The proposed approach uses as a starting point the cost-sensitive classification algorithm presented in [11]. This algorithm receives as input a sample

$$\{(x_i, y_i, r_i), i = 1, \ldots, m\}, \tag{1}$$

for some $m \in \mathbb{N}$, where the pair $(x_i, y_i) \in X \times \{-1, 1\}$ gathers a *pattern* and a *label*, thus corresponding to a labeled example for the classification problem while r_i (henceforth referred to as *quality*) quantifies the confidence that the association of label y_i to

object x_i in previous example is actually correct. The algorithm is formalized, as usual, through a constrained optimization problem that reads as follows:

$$\begin{aligned} &\min_{w,b} \frac{1}{2} w \cdot w + C \sum_{i=0}^{m} \xi_i \\ &\quad w \cdot \left(\Phi(x_i) - \frac{r_i}{2} w \right) + b \geq 1 - \xi_i \qquad \forall i : y_i = +1 \\ &\quad w \cdot \left(\Phi(x_i) + \frac{r_i}{2} w \right) + b \leq -1 + \xi_i \qquad \forall i : y_i = -1, \\ &\quad \xi_i \geq 0 \qquad \forall i = 1, \ldots, m. \end{aligned} \tag{2}$$

In the above formulation Φ is a mapping from X onto a space H, within which the optimization process aims at finding a hyperplane (having w and b respectively as slopes and threshold) separating the images through Φ of patterns associated to positive labels from those associated to negative labels. Slack variables $\xi_1, \ldots, \xi_m$ allow this hyperplane to perform mistakes in the separation process (precisely, pattern x_i is misclassified when $\xi_i > 1$), and the value of parameter $C > 0$ balances the two optimization components, namely the number of mistakes and the separator margin (the latter being related to the generalization capability of the inferred classifier [12]). The difference between (2) and the problem at the basis of classical support vector classifier [9] lies in the presence of r_i, whose effect is that of performing a *virtual shift* on the patterns images according to the related quality value. More precisely:

- when $r_i > 0$ point $\Phi(x_i)$ is shifted along the direction normal w.r.t. the separating surface, moving towards the latter with the effect of increasing the distance between the actual classifier and the original pattern position (see Fig. 1(a));
- when $r_i < 0$ the shift occurs in the opposite direction, so that if r_i is sufficiently large pattern x_i will be misclassified (see Fig. 1(b));
- finally, when r_i has a null value nothing changes in the problem formulation.

Applying standard results of duality theory [13] the solution of (2) is linked to that of the following problem:

$$\begin{aligned} &\max_{\alpha_1, \ldots, \alpha_m} \sum_i \alpha_i - \frac{1}{2(1 + \sum_l \alpha_l r_l)} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ &\quad \sum_i \alpha_i y_i = 0 \\ &\quad 0 \leq \alpha_i \leq C \qquad \forall i = 1, \ldots, m, \end{aligned} \tag{3}$$

where k is the kernel function associated to Φ, defined by $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$. More precisely, denoted by $\alpha_1^*, \ldots, \alpha_m^*$ the optimal values for (3), the solution of (2) occurs in correspondence of:

$$w^* = \frac{1}{1 + \sum_l \alpha_l^* r_l} \sum_i \alpha_i^* y_i \Phi(x_i), \tag{4}$$

$$b^* = y_i - w^* \cdot \Phi(x_i) + \frac{y_i r_i}{2} w^* \cdot w^* \quad \text{for } i \text{ such that } 0 < \alpha_i^* < C, \tag{5}$$

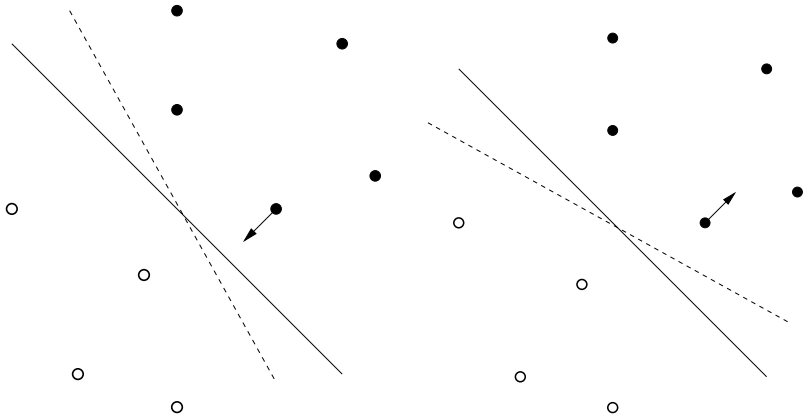

Fig. 1. Virtually shifting patterns in order to take into account the quality of their corresponding examples: (a) the image of a pattern such that $r_i > 0$ is shifted toward the separating surface of the original problem (plain line), with the effect of increasing the distance between the classifier (dashed line) and the original point position; (b) when $r_i < 0$ the image is shifted in the opposite direction, so that its distance w.r.t. the classifier is reduced.

so that the label of a new point x^{new} can be inferred as $y^{\text{new}} = \text{sign}(w^* \cdot \Phi(x^{\text{new}}) + b^*)$, which translates into:

$$y^{\text{new}} = \text{sign}\left(\frac{1}{1+\sum_l \alpha_l^* r_l} \sum_i \alpha_i^* y_i k(x_i, x^{\text{new}}) + y_j + \right.$$
$$\left. - \frac{1}{1+\sum_l \alpha_l^* r_l} \sum_i \alpha_i^* y_i k(x_i, x_j) + \frac{y_j r_j}{2} \frac{1}{(1+\sum_l \alpha_l^* r_l)^2} \sum_{i,h} \alpha_i^* \alpha_h^* y_i y_h k(x_i, x_h) \right),$$

where j in the second sum has been chosen so that $0 < \alpha_j^* < C$.

It has been shown that this approach promotes the correct classification of examples when r_i is positive and conversely tends to misclassify examples characterized by a negative r_i [11]. The evident drawback consists in the high nonlinearity of the objective function in (3), while in standard support vector classification this function is quadratic. This fact essentially precludes an efficient processing of the related optimization problem in order to numerically approximate its solution.

3 Classification of Imbalanced Data

An interesting application of the procedure described in Sect. 2 having the additional benefit of being formulated as a quadratic optimization concerns the classification of imbalanced data. Indeed, this can be easily accomplished through association of a fixed positive quality value to each example from the under-represented class and a fixed

negative quality value to the remaining examples. As a result, the classification algorithm will be pushed to correctly classify the under-represented class, possibly at the expense of an erroneous classification of the over-represented one. The simplest way to implement this strategy is that of setting $r_i = y_i$, so that (2) becomes

$$\begin{aligned} &\min_{w,b} \frac{1}{2} w \cdot w + C \sum_{i=0}^{m} \xi_i \\ &\quad y_i \left(w \cdot \left(\Phi(x_i) - \frac{1}{2} \right) + b \right) \geq 1 - \xi_i \qquad \forall i = 1, \ldots, m \\ &\quad \xi_i \geq 0 \qquad \forall i = 1, \ldots, m. \end{aligned} \tag{6}$$

Note how this formulation uses the "costs" r_i in order to directly shape the problem constraint instead of modifying its objective function as in other cost-sensitive approaches to support vector classification. The dual form of this problem simplifies to

$$\begin{aligned} &\max_{\alpha_1, \ldots, \alpha_m} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ &\quad \sum_i \alpha_i y_i = 0 \\ &\quad 0 \leq \alpha_i \leq C \qquad \forall i = 1, \ldots, m. \end{aligned} \tag{7}$$

The latter problem coincides with the original support vector classification algorithm [9] whose solution $(w^{\text{svc}}, b^{\text{svc}})$ in the primal space can be efficiently found using any of the available standard tools and techniques (refer for instance to the SMO algorithm [14]). Substituting $r_i = y_i$ in (4–5) shows that the the optimal solution of (6) occurs in correspondence of $w^* = w^{\text{svc}}$ and $b^* = b^{\text{svc}} + \frac{1}{2}||w^{\text{svc}}||^2$.

It is also easy to prove that the same analysis can be applied to the general case $r_i = r y_i$ for any $r > 0$, obtaining the optimization problem

$$\begin{aligned} &\min_{w,b} \frac{1}{2} w \cdot w + C \sum_{i=0}^{m} \xi_i \\ &\quad y_i \left(w \cdot \left(\Phi(x_i) - \frac{r}{2} \right) + b \right) \geq 1 - \xi_i \qquad \forall i = 1, \ldots, m \\ &\quad \xi_i \geq 0 \qquad \forall i = 1, \ldots, m, \end{aligned} \tag{8}$$

whose optimal values are:

$$w^* = w^{\text{svc}}, \tag{9}$$

$$b^* = b^{\text{svc}} + r \frac{1}{2} ||w^{\text{svc}}||^2. \tag{10}$$

The solution in (9–10) corresponds to rediscovering the so-called *boundary movement method* for the classification of imbalanced datasets with support vector machines [6], which captures the intuitive idea of suitably shifting the threshold value for the separating hyperplane in order to account for the under-representation of one class w.r.t. the remaining one. Equation (10) states that this shift can be measured in terms of the

squared norm of the separating hyperplane slopes. The latter quantity represents the optimal value of the first addend in the objective function of (8), which in turn can be expressed as a function of the maximized *margin* of the classifier [9] through an inverse relation property: precisely

$$\mu^* = \frac{2}{||w^*||}, \tag{11}$$

where μ^* denotes the maximal margin.

Note also that the shift is always positive: indeed it is expressed as a positive multiple of a squared norm in (10), meaning that the region of X associated to the under-represented class is being enlarged.

The obtained result implicitly sets the following rule in order to preliminary fix the shift amount when applying the boundary movement method:

1. solve the original formulation support vector classification problem (i.e. that not taking into account the data imbalance), obtaining w^{svc} and b^{svc} through standard tools;
2. compute the value $\frac{1}{2}||w^{\text{svc}}||^2$;
3. fix the initial shift as r times the quantity computed on previous point, with $r > 0$.

Elementary algebraic considerations involving equations 10 and 11 rediscover the natural assertion that when the optimal margin assumes a big value, a unit move of b^{svc} will correspond to shifting the hyperplane by a small (possibly fractional) number of margins and *vice versa*.

4 Conclusions

This work showed that the boundary movement method used as a post-processing technique for classifiers learnt through the support vector method in order to account for class imbalance can be viewed as a special case of a cost-sensitive modification of the original support vector learning algorithm. The formulation of the latter, where costs are used to shape the optimal problem's constraints rather than its objective function, suggests a rule for setting the boundary movement involving the classifier margin.

References

1. Papadimitriou, C.H.: Computational complexity. Addison-Wesley, Reading (1994)
2. Zhao, W., Chellappa, R., Phillips, P.J., Rosenfeld, A.: Face Recognition: A Literature Survey. ACM Computing Surveys 35(4), 399–458 (2003)
3. Tang, Y., Zhang, Y., Chawla, N.V., Krasser, S.: SVMs Modeling for Highly Imbalanced Classification. IEEE Transactions on Systems, Man, and Cybernetics, Part B 39(1), 281–288 (2009)
4. Chawla, N., Bowyer, K., Hall, L., Kegelmeyer, W.: SMOTE: Syntetic Minority Oversampling Techinque. Journal of Artificial Intelligence Reserarch 16, 321–357 (2002)
5. Stefanowski, J., Wilk, S.: Improving Rule-Based Classifiers Induced by MODLEM by Selective Preprocessing of Imbalanced Data. In: Proceedings of the Workshop RSKD at European Conference on Machine Learning and Principles of Knowledge Discovery in Databases, Warszawa, September 17-21, pp. 54–65 (2007)

6. Wu, G., Chang, E.: Class-Boundary Alignment for Imbalanced Dataset Learning. In: ICML 2003 Workshop on Learning from Imbalanced Data Sets (II), pp. 49–56 (2003)
7. Ben-Hur, A., Horn, D., Siegelmann, H.T., Vapnik, V.: Support Vector Clustering. Journal of Machine Learning Research 2, 125–137 (2001)
8. Wu, G., Chang, E.Y.: Adaptive Feature-Space Conformal Transformation for Imbalanced-Data Learning. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003), pp. 816–823 (2003)
9. Cortes, C., Vapnik, V.: Support-Vector Networks. Machine Learning 20, 121–167 (1995)
10. Elkan, C.: The Foundations of Cost-Sensitive Learning. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence, pp. 973–978 (2001)
11. Apolloni, B., Malchiodi, D., Natali, L.: A Modified SVM Classification Algorithm for Data of Variable Quality. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007, Part III. LNCS (LNAI), vol. 4694, pp. 131–139. Springer, Heidelberg (2007)
12. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2002)
13. Fletcher, R.: Practical Methods of Optimisations, 2nd edn. John Wiley & Sons, Chichester (1987)
14. Platt, J.: Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In: Schölkopf, B., Burges, C.J.C., Smola, A.J. (eds.) Advances in Kernel Methods – Support Vector Learning, pp. 185–208. MIT Press, Cambridge (1999)

Genetic Algorithm Modeling with GPU Parallel Computing Technology

Stefano Cavuoti[1], Mauro Garofalo[2], Massimo Brescia[1,3,⋆], Antonio Pescape'[2], Giuseppe Longo[1,4], and Giorgio Ventre[2]

[1] Department of Physics, University Federico II, Via Cinthia 6, I-80126 Napoli, Italy
brescia@oacn.inaf.it
[2] Department of Computer Engineering and Systems, University Federico II, Via Claudio 21, I-80125 Napoli, Italy
[3] INAF, Astronomical Observatory of Capodimonte, Via Moiariello 16, I-80131 Napoli, Italy
[4] Visiting Associate, California Institute of Technology, Pasadena, CA 91125, USA

Abstract. We present a multi-purpose genetic algorithm, designed and implemented with GPGPU / CUDA parallel computing technology. The model was derived from a multi-core CPU serial implementation, named GAME, already scientifically successfully tested and validated on astrophysical massive data classification problems, through a web application resource (DAMEWARE), specialized in data mining based on Machine Learning paradigms. Since genetic algorithms are inherently parallel, the GPGPU computing paradigm has provided an exploit of the internal training features of the model, permitting a strong optimization in terms of processing performances and scalability.

Keywords: genetic algorithms, GPU programming, data mining.

1 Introduction

Computing has started to change how science is done, enabling new scientific advances through enabling new kinds of experiments. They are also generating new kinds of data of increasingly exponential complexity and volume. Achieving the goal of being able to use, exploit and share most effectively these data is a huge challenge. The harder problem for the future is heterogeneity, of platforms, data and applications, rather than simply the scale of the deployed resources. Current platforms require the scientists to overcome computing barriers between them and the data [1].

The present paper concerns the design and development of a multi-purpose genetic algorithm implemented with the GPGPU/CUDA parallel computing technology. The model comes out from the machine learning supervised paradigm, dealing with both regression and classification scientific problems applied on

⋆ Corresponding author.

B. Apolloni et al. (Eds.): *Neural Nets and Surroundings*, SIST 19, pp. 29–39.
DOI: 10.1007/978-3-642-35467-0_4

massive data sets. The model was derived from the original serial implementation, named GAME (Genetic Algorithm Model Experiment) deployed on the DAME [8] Program hybrid distributed infrastructure and made available through the DAMEWARE [9] data mining (DM) web application. In such environment the GAME model has been scientifically tested and validated on astrophysical massive data sets problems with successful results [2]. As known, genetic algorithms are derived from Darwin's evolution law and are intrinsically parallel in its learning evolution rule and processing data patterns. The parallel computing paradigm can indeed provide an optimal exploit of the internal training features of the model, permitting a strong optimization in terms of processing performances.

2 Data Mining Based on Machine Learning and Parallel Computing

Let's start from a real and fundamental assumption: we live in a contemporary world submerged by a tsunami of data. Many kinds of data, tables, images, graphs, observed, simulated, calculated by statistics or acquired by different types of monitoring systems. The recent explosion of World Wide Web and other high performance resources of Information and Communication Technology (ICT) are rapidly contributing to the proliferation of such enormous information repositories. Machine learning (ML) is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data. A *learner* can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. These data form the so called Knowledge Base (KB): a sufficiently large set of examples to be used for training of the ML implementation, and to test its performance. The DM methods, however, are also very useful to capture the complexity of small data sets and, therefore, can be effectively used to tackle problems of much smaller scale [2].

DM on Massive Data Sets (MDS) poses two important challenges for the computational infrastructure: asynchronous access and scalability. With synchronous operations, all the entities in the chain of command (client, workflow engine, broker, processing services) must remain up for the duration of the activity: if any component stops, the context of the activity is lost.

Regarding scalability, whenever there is a large quantity of data, the more affordable approach to making learning feasible relies in splitting the problem in smaller parts (parallelization) sending them to different CPUs and finally combine the results together. So far, the parallel computing technology chosen for this purpose was the GPGPU.

GPGPU is an acronym standing for General Purpose Computing on Graphics Processing Units. It was invented by Mark Harris in 2002, [3], by recognizing the trend to employ GPU technology for not graphic applications. With such term we mean all techniques able to develop algorithms extending computer graphics but running on graphic chips. In general the graphic chips, due to

their intrinsic nature of multi-core processors (many-core) and being based on hundreds of floating-point specialized processing units, make many algorithms able to obtain higher (one or two orders of magnitude) performances than usual CPUs (Central Processing Units). They are also cheaper, due to the relatively low price of graphic chip components.

The choice of graphic device manufacturers, like NVIDIA Corp., was the many-core technology (usually many-core is intended for multi-core systems over 32 cores). The many-core paradigm is based on the growth of execution speed for parallel applications. Began with tens of cores smaller than CPU ones, such kind of architectures reached hundreds of core per chip in a few years. Since 2009 the throughput peak ratio between GPU (many-core) and CPU (multi-core) was about 10:1. Such a large difference has pushed many developers to shift more compu-ting-expensive parts of their programs on the GPUs.

3 The GAME Model

An important category of supervised ML models and techniques, in some way related with the Darwin's evolution law, is known as evolutionary (or genetic) algorithms, sometimes also defined as based on genetic programming [4]. The slight conceptual difference between evolutionary and genetic algorithms is that the formers are problem-dependent, while the latters are very generic.

GAME is a pure genetic algorithm specially designed to solve supervised optimizations problems related with regression and classification functionalities, scalable to efficiently manage MDS and based on the usual genetic evolution methods (crossover, genetic mutation, roulette/ranking, elitism). In order to give a level of abstraction able to make simple to adapt the algorithm to the specific problem, a family of polynomial developments was chosen for GAME model. This methodology makes the algorithm itself easily expandable, but this abstraction requires a set of parameters that allows fitting the algorithm to the specific problem.

From an analytic point of view, a pattern, composed of N features contains an amount of information correlated between the features corresponding to the target value. Usually in a real scientific problem that correlation is *masked* from the noise (both intrinsic to the phenomenon, and due to the acquisition system); but the unknown correlation function can ever be approximated with a polynomial sequence, in which the degree and non-linearity of the chosen function determine the approximation level. The generic function of a polynomial sequence is based on these simple considerations:

Given a generic dataset with N features and a target t, pat a generic input pattern of the dataset, $pat = (f_1, ..., f_N, t)$ and $g(x)$ a generic real function, the representation of a generic feature f_i of a generic pattern, with a polynomial sequence of degree d is:

$$G(f_i) \cong a_0 + a_1 g(f_i) + ... + a_d g^d(f_i) \tag{1}$$

Hence, the k-th pattern (pat_k) with N features may be represented by:

$$Out(pat_k) \cong \sum_{i=1}^{N} G(f_i) \cong a_0 + \sum_{i=1}^{N} \sum_{j=1}^{d} a_j g^j(f_i) \tag{2}$$

Then target t_k, concerning to pattern pat_k, can be used to evaluate the approximation error of the input pattern to the expected value:

$$E_k = (t_k - Out(pat_k))^2 \tag{3}$$

If we generalize the expression (2) to an entire dataset, with NP number of patterns ($k = 1, ..., NP$), at the end of the *forward* phase (batch) of the GA, we obtain NP expressions (2) which represent the polynomial approximation of the dataset.

In order to evaluate the fitness of the patterns as extension of (3), the Mean Square Error (MSE) or Root Mean Square Error (RMSE) may be used.

Then we define a GA with the following characteristics:

- The expression (2) is the fitness function;
- The array $(a_0, ..., a_M)$ defines M genes of the generic chromosome (initially they are generated random and normalized between -1 and +1);
- All the chromosomes have the same size (constrain from a classic GA);
- The expression (3) gives the standard error to evaluate the fitness level of the chromosomes;
- The population (genome) is composed by a number of chromosomes imposed from the choice of the function $g(x)$ of the polynomial sequence.

About the last item, this number is determined by the following expression:

$$NUM_{chromosomes} = (B \cdot N) + 1 \tag{4}$$

where N is the number of features of the patterns and B is a multiplicative factor that depends from the $g(x)$ function, which in the simplest case is just 1, but can arise to 3 or 4 in more complex cases. The parameter B also influences the dimension of each chromosome (number of genes):

$$NUM_{genes} = (B \cdot d) + 1 \tag{5}$$

where d is the degree of the polynomial. For example if we use the trigonometric polynomial expansion, given by the following expression (hereinafter polytrigo),

$$g(x) = a_0 + \sum_{m=1}^{d} a_m \cos(mx) + \sum_{m=1}^{d} b_m \sin(mx) \tag{6}$$

in order to have 200 patterns composed by 11 features, the expression using (2) with degree 3, will become:

$$Out(pat_{k=1...200}) \cong \sum_{i=1}^{11} G(f_i) \cong a_0 + \sum_{i=1}^{11}\sum_{j=1}^{3} a_j \cos(jf_i) + \sum_{i=1}^{11}\sum_{j=1}^{3} b_j \sin(jf_i) \quad (7)$$

In the last expression we have two groups of coefficients (sin and cosine), so B will assume the value 2. Hence the generic genome (population at a generic evolution stage), will be composed by 23 chromosomes, given by equation (4), each one with 7 genes $[a_0, a_1, a_2, a_3, b_1, b_2, b_3]$, given by equation (5), with each single gene (coefficient of the polynomial) in the range $[-1, +1]$.

In the present project, the idea is to build a GA able to solve supervised crispy classification and regression problems, typically related to an high-complexity parameter space where the background analytic function is not known, except for a limited number of couples of input-target values, representing valid solutions to a physical category of phenomena. A typical case is to classify astronomical objects based on some solution samples (the KB) or to predict new values extracted by further observations. To accomplish such behavior we designed a function (a polynomial expansion) to combine input patterns. The coefficients of such polynomials are the chromosome genes. The goal is indeed to find the best chromosome so that the related polynomial expansion is able to approximate the right solutions to input pattern classification/regression. So far, the fitness function for such representation consists of the training error, obtained as absolute difference between the polynomial output and the target value for each pattern. Due to the fact that we are interested to find the minimum value of the error, the fitness is calculated as the complement of the error (i.e. 1-error) and the problem is reduced to find the chromosome achieving the maximum value of fitness.

4 The GPU-Based GAME Implementation

In all execution modes (use case), GAME exploits the polytrigo function (6), consisting in a polynomial expansion in terms of sum of sins and cosines. Specifically in the training use case, corresponding to the GA building and consolidation phase, the polytrigo is used at each iteration as the transformation function applied to each chromosome to obtain the output on the problem input dataset, and indirectly also to evaluate the fitness of each chromosome. It is indeed one of the critical aspects of the serial algorithm to be investigated during the parallelization design process.

Moreover, after having calculated the fitness function for all genetic population chromosomes, this information must be back-propagated to evolve the genetic population. This back and forth procedure must be replicated as many times as it is the training iteration number or the learning error threshold, both decided and imposed by the user at setup time of any experiment. The direct consequence of the above issues is that the training use case takes much more execution time than the others (such as test and validation), and therefore is the one we are going to optimize.

Main design aspect approaching the software architecture analysis for the GPU is the partition of work: i.e. which work should be done on the CPU vs. the GPU. We have identified the time consuming critical parts to be parallelized by executing them on the GPU. They are the generation of random chromosomes and the calculation of the fitness function of chromosomes. The key principle is that we need to perform the same instruction simultaneously on as much data as possible. By adding the number of chromosomes to be randomly generated in the initial population as well as during each generation, the total number of involved elements is never extremely large but it may occur with a high frequency. This is because also during the population evolution loop a variable number of chromosomes are randomly generated to replace older individuals. To overcome this problem we may generate a large number of chromosomes randomly *una tantum*, by using them whenever required. On the contrary, the evaluation of fitness functions involves all the input data, which is assumed to be massive datasets, so it already has an intrinsic data-parallelism. Since CUDA programming involves code running concurrently on a host with one or more CPUs and one or more CUDA-enabled GPU, it is important to keep in mind that the differences between these two architectures may affect application performance to use CUDA effectively. The function polytrigo takes about three-quarters of the total execution time of the application, while the total including child functions amounts to about 7/8 of total time execution. This indeed has been our first candidate for parallelization. In order to give a practical example, for the interested reader, we report the source code portions related to the different implementation of the polytrigo function, of the serial and parallelized cases.

C++ serial code for polytrigo function (equation 6):

```
for (int i = 0; i < num_features; i++) {
   for (int j = 1; j <= poly_degree; j++) {
    ret +=  v[j] * cos(j * input[i]) + v[j + poly_degree] *
            * sin(j * input[i]); } }
```

CUDA C (Thrust) parallelized code for polytrigo function (equation 6):

```
struct sinFunctor {  __host__ __device__
 double operator()(tuple <double, double> t) {
    return sin(get < 0 > (t) * get < 1 > (t)); }};
struct cosFunctor {  __host__ __device__
 double operator()(tuple <double, double> t) {
    return cos(get < 0 > (t) * get < 1 > (t)); }};
thrust::transform(thrust::make_zip_iterator(
    thrust::make_tuple(j.begin(), input.begin())),
    thrust::make_zip_iterator(
       thrust::make_tuple(j.end(), input.end())),
       ret.begin(), sinFunctor(), cosFunctor());
```

Noting that, while the vector $v[]$ is continuously evolving, $input[]$ (i.e. the elements of the input dataset) are being used in calculation of ret at each iteration but they are never altered. We rewrite the function by calculating in advance the sums of sins and cosines, storing the results in two vectors and then use them in the function polyTrigo() at each iteration. This brings huge benefits because we calculate trigonometric functions, which are those time consuming, only once instead of at every iteration and exploit the parallelism on large amount of data because it assumes that we have large input datasets.

From the time complexity point of view, by assuming to have as many GPU cores as population chromosomes, the above CUDA C code portion would take constant time, instead of polynomial time required by the corresponding C++ serial code.

5 The Experiment

In terms of experiments, the two CPU versions of GAME, the original and an optimized version of the serial algorithm (hereinafter serial and Opt respectively), together with the final version for GPU (hereinafter ELGA), have been compared basically by measuring their performance in terms of execution speed, by also performing an intrinsic evaluation of the overall scientific performances. The optimized algorithm is the serial version adapted by modifying the code portions which are candidate to be parallelized in the final GPU release.

Initially, the tests have been organized by distinguishing between classification and regression functional modes. By analyzing early trials, however, it resulted that the performance growth was virtually achieved in both cases. So far, we limit here the discussion details to a classification experiment, done in the astrophysical context.

The scientific problem used here as a test bed for data mining application of the GAME model is the search (classification) of Globular Cluster (GC) populations in external galaxies [2]. This topic is of interest to many astrophysical fields: from cosmology, to the evolution of stellar systems, to the formation and evolution of binary systems.

The dataset used in this experiment consists in wide field HST observations of the giant elliptical NGC1399 in the Fornax cluster, [5]. The subsample of sources used to build our Base of Knowledge, to train the GAME model is composed by 2100 sources with all photometric and morphological information, [2]. Finally, our classification dataset consisted of 2100 patterns, each composed by 11 features (including the two targets, corresponding to the classes GC and not GC used during the supervised training phase).

The performance was evaluated on several hardware platforms. We compared our production GPU code with a CPU implementation of the same algorithm. The benchmarks were run on a 2.0 GHz Intel Core i7 2630QM quad core CPU running 64-bit Windows 7 Home Premium SP1. The CPU code was compiled using the Microsoft C/C++ Optimizing Compiler version 16.00 and GPU benchmarks were performed using the NVIDIA CUDA programming toolkit version