Improving multilayer perceptron classifiers AUC performance.

An approach in biomedical image analysis for breast cancer CAD supported by eInfrastructures.

Raúl Ramos Pollán

Faculty of Engineering, University of Porto

Department of Informatics Engineering

October 2011

Supervisors

Doctor Miguel Ángel Guevara López
Senior Researcher

Instituto de Engenharia Mecânica e Gestão Industrial
Faculty of Engineering
University of Porto

Doctor Eugenio de Costa Oliveira
Full Professor

Departamento de Engenharia Informática
Faculty of Engineering
University of Porto

A thesis submitted to the University of Porto in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Informatics Engineering

Porto, Portugal, October 2011

Abstract

This thesis addresses the problem of using efficiently Receiver Operating Characteristic (ROC) analysis in machine learning and, specifically, in multilayer perceptron based classifiers, both from a theoretical and a practical approach. It proposes a general formulation to improve the AUC performance (Area Under the ROC Curve) of existing machine learning methods that is affordable to implement and demonstrates experimentally its effectiveness with different kinds of multilayer perceptron training algorithms. As a means to this, a new computationally efficient method was devised to compute the AUC and two software frameworks were developed to facilitate the exploitation of distributed eInfrastructures for machine learning in general and ROC analysis in specific. Finally, these developments were applied to a real world case in the field of biomedical image analysis and the results obtained are herewith described.

ROC analysis is commonly used to judge the discrimination ability of a binary test for predictive purposes. A test might be, for instance, of chemical or biomedical nature but also a machine learning classifier aiming at distinguishing the two different classes of a binary dataset. It has traditionally been a tool in biomedical decision making and, during the last decade, ROC analysis has been increasingly used in machine learning, where it becomes especially useful when applied to biomedical data. ROC curves represent the trade-off between false-positives and true-positives of a classifier at different decision threshold levels, and the AUC is taken as a single scalar metric to compare classifier performance, seeking classifiers having greater AUC. This complements other commonly used metrics, such as *accuracy* (the percent of dataset elements correctly classified), *specificity*, *precision*, *recall*, etc.

Machine Learning (ML) algorithms have been historically devised to minimize some error rate loss function yielding better classifier accuracy. Although related, literature reports evidences that error rate minimization does not necessarily yield to AUC maximization and, in recent years, attempts have tried to use AUC in optimization problems requiring mostly designing new ML algorithms or heavily transforming existing ones for such task. The approach presented in this thesis provides a theoretical formulation to allow a straight forward integration of AUC optimization into existing ML algorithms, enabling the affordable reuse of the vast amount of techniques developed in the area to date. Then, it provides experimental evidence of its efficacy by using multilayer perceptrons with different training algorithms.

However, AUC calculation is a computationally expensive task that requires sorting the scores assigned by a certain classifier to the elements a dataset. As ML algorithms are already computationally intensive, calculating the AUC efficiently is key so that our approach does not render them impractical. Therefore, as part of this thesis and herewith used, an efficient error-bounded method has been devised to approximate the AUC with arbitrary precision, rendering its computational complexity linearly proportional to the number of dataset elements.

In addition, providing sufficient statistical evidence to these claims requires extensive use of computing resources to train and evaluate different configurations of datasets and ML methods. eInfrastructures such as computer clusters, Grids and, more recently, Clouds, provide vast amounts of computing resources, but their utilization is mostly tied to the access tools and methods offered by the specific infrastructure provider, middleware, etc. Furthermore, to gather the required evidence for our purposes, we need the ability to manage in an agile way many different configurations of third party ML algorithms over a diversity of datasets and training conditions, well beyond the capabilities of most user tools existing today to interact with eInfrastructures. To overcome this, two software frameworks were developed in this thesis. On one side, BiomedTK (the *Biomedical Data Analysis Toolkit*) allows the integration of third party ML algorithms, datasets and ROC analysis, enabling the systematic exploration of the space of their possible configurations. On the other side, C3 (the *Cloud Computing Colonies* framework) uses Java industry standards to provide a homogeneous way for any software (such as BiomedTK) to access computing resources scattered throughout eInfrastructures of different nature. C3 deploys colonies of *Job Agents* onto available eInfrastructures and provides a unified communications channel to send them data and deploy and run applications, regardless their actual access method.

At last, the results of this thesis have been successfully applied to obtain well performing classifiers offering automated second opinion for assisted diagnosis on breast cancer. In tight collaboration with specialized radiologists, datasets including their annotations have been analyzed as they were produced, yielding trained classifiers ready to be clinically validated for assisted diagnosis and integrated within their graphical workstations, medical workflows and informatics systems.

Resumo

Esta tese aborda o problema do uso eficiente da análise ROC (*Receiver Operating Characteristic*) em aprendizagem automática e, especificamente, em classificadores baseados em *perceptrons* multicapas, desde uma perspectiva teórica e prática. Uma formulação geral é proposta para melhorar o desempenho AUC (área sob a curva ROC) dos métodos de aprendizagem automática existentes, e se demonstra experimentalmente a sua eficácia com diferentes tipos de algoritmos de treinamento para *perceptrons* multicapa. Para isso, foi preciso desenvolver um novo método computacionalmente eficiente para calcular o AUC e dois ferramentas de software que facilitan a utilização de e-Infraestruturas distribuídas na aprendizagem automática em geral e na análise ROC em específico. Finalmente, estas contribuições foram aplicadas num caso real no campo da análise de imagens biomédicas e os resultados atingidos são aqui descritos.

A análise ROC é comummente usada para julgar a capacidade de discriminação de um teste binário para finalidades de predição. Um teste pode ser, por exemplo, de natureza química ou biomédica, mas também pode ser um classificador automático que tem como objetivo distinguir as classes de um *dataset* binário. A análise ROC tem sido tradicionalmente uma ferramenta na toma de decisões biomédicas e, durante a última década, é cada vez mais usado em aprendizagem automática, onde é especialmente útil quando é aplicado a dados biomédicos. As curvas ROC representam o balance entre falsos positivos e verdadeiros positivos de um classificador a diferentes limiares (*thresholds*) de decisão, e o AUC é empregado como uma métrica escalar para comparar o desempenho dos classificadores. Isto complementa outras métricas comummente usadas, tais como a taxa de erro, a precisão, a especificidade, o *recall*, etc.

Os algoritmos de aprendizagem automática têm sido historicamente concebidos para minimizar a taxa de erro dos classificadores. Apesar de estar relacionados, a literatura evidencia que a minimização da taxa de erro não significa necessariamente uma maximização do AUC e, nos últimos anos, tem existido tentativas de usar o AUC em problemas de optimização que requerem principalmente criar novos algoritmos ou transformar significativamente os já existentes. A abordagem apresentada nesta tese fornece uma formulação teórica que permite a integração imediata da optimização do AUC nos algoritmos já existentes, facilitando a reutilização das técnicas desenvolvidas na área até o momento. Após, ele fornece evidência experimental de sua eficácia usando *perceptrons* multicapa com diferentes algoritmos de treino.

No entanto, o cálculo da AUC é uma tarefa computacionalmente custosa que requer a ordenação dos *scores* atribuídos por um classificador a um *dataset*. Dado que os algoritmos de aprendizagem já são computacionalmente intensivos, um cálculo eficiente do AUC é fundamental para que a nossa abordagem não os torne impraticáveis. Portanto, nesta tese se proponde um método para aproximar eficientemente o AUC com precisão arbitrária, tornando a sua complexidade computacional linearmente proporcional ao tamanho do *dataset*.

Além disso, a obtenção de evidência estatística suficiente para apoiar estes argumentos requer o uso extensivo de recursos de computação para treinar e avaliar diferentes configurações de *datasets* e métodos de aprendizagem automático. As e-Infraestruturas, tais como clusters de computadores, computação em grelha e, mais recentemente, *clouds*, fornecem grandes quantidades de recursos computacionais, mas sua utilização é principalmente ligada à ferramentas propias da e-Infraestrutura específica, middleware, etc. Neste senso, para reunir a evidência necessária para os nossos propósitos, é precisa a capacidade de gerir de forma ágil muitas configurações de algoritmos através de uma diversidade de *datasets* e condições de treino, bem além das capacidades das ferramentas existentes hoje para interagir com as e-Infraestruturas. Para superar isso, dois produtos de software foram desenvolvidos neste trabalho. De um lado, o BiomedTK (*Biomedical Data Analysis Toolkit*) facilita a integração de algoritmos de aprendizagem automático, *datasets* e análise ROC, permitindo a exploração sistemática do espaço de suas possíveis configurações. Por outro lado, o C3 (*Cloud Computing Colonies framework*) utiliza os standards Java para fornecer uma forma homogénea de aceder a recursos de computação espalhados sob e-Infraestruturas diversas. O C3 mantém colónias de agentes (*Job Agents*) em e-Infraestruturas disponíveis e fornece um canal de comunicação unificado para enviá-los dados e executar aplicações, independentemente do seu método de acesso real.

Finalmente, os resultados desta tese têm sido aplicados com sucesso na obtenção de classificadores automáticos que permitem oferecer uma segunda opinião no diagnostico assistido do cancro da mama. Em colaboração estreita com radiologistas especializados, *datasets* foram analisados ao ser produzidos, gerando classificadores treinados prontos para ser validados clinicamente no diagnóstico assistido e integrados dentro de suas estações de trabalho gráficas, fluxos de trabalho médicos e sistemas informáticos.

Résumé

Cette thèse aborde le problème de l'utilisation efficace de l'analyse ROC (*Receiver Operating Characteristic*) dans l'apprentissage automatique et, plus précisément, dans classificateurs basées sur *perceptrons* multicouches, dans une approche à la fois théorique et pratique. On propose une formulation générale pour améliorer la performance AUC (area sous la courbe *ROC*) des méthodes d'apprentissage automatique existants qui est abordable d'implémenter et on démontre expérimentalement son efficacité avec différents types d'algorithmes d'entraînement pour *perceptrons* multicouches. Comme moyen de cela, une nouvelle méthode a été conçue pour calculer efficacement le AUC et deux logiciels ont été développés que facilitent l'exploitation des e-Infrastructures distribués pour l'apprentissage automatique en général et pour l'analyse ROC en particulier. Enfin, ces développements ont été appliqués à un cas réel dans le domaine de l'analyse d'images biomédicales et les résultats obtenus sont décrits ici.

L'analyse ROC est couramment utilisée pour juger de la capacité de discrimination d'un test binaire à des fins prédictives. Un test pourrait être, par exemple, de nature chimique ou biomédicale, mais aussi un classificateur automatique visant à distinguer les deux classes d'un *dataset* binaire. L'analyse ROC a toujours été un outil dans la prise de décision biomédicale et, pendant la dernière décennie, il a été de plus en plus utilisé dans l'apprentissage automatique, où il devient particulièrement utile lorsqu'il est appliqué à des données biomédicales. Les courbes ROC représentent le compromis entre faux positifs et vrais positifs d'un classificateur à différents niveaux de seuil de décision (*threshold*), et l'AUC est prise comme une métrique scalaire unique pour comparer sa performance, en cherchent des classificateurs qu'ont l'AUC plus élevée. Ça complément d'autres mesures couramment utilisées, telles que le taux d'erreur, la précision, la spécificité, le *recall*, etc

Les algorithmes d'apprentissage automatique ont été historiquement conçus pour minimiser le taux d'erreur des classificateurs. Bien que liés, la littérature rapport des évidences comme que la minimisation taux d'erreur ne conduit pas nécessairement à la maximisation de l'AUC et, ces dernières années, des tentatives ont essayé d'utiliser l'AUC dans des problèmes d'optimisation nécessitant la plupart d'eux la conception des nouveaux algorithmes ou la transformation significative de ceux existantes. L'approche présentée dans cette thèse propose une formulation théorique que permettre une

intégration de l'optimisation AUC dans les algorithmes existants, permettant de cette façon la réutilisation abordable de la grande quantité de techniques développées dans à ce jour. Ensuite, il fournit la preuve expérimentale de son efficacité en utilisant *perceptrons* multicouches avec des différents algorithmes d'entraînement.

Cependant, le calcul de l'AUC est une tâche coûteuse en ressources informatiques. Étant donné que les algorithmes d'apprentissage automatique sont déjà intensifs, le calcul efficace de l'AUC est clé pour que notre approche ne les rendre pas inutiles. Pour ce la, dans cette thèse, on a conçu une procédure pour approximer l'AUC avec une précision d'erreur arbitraire et bornée par l'utilisateur, en faisant sa complexité de calcul linéairement proportionnel au nombre d'éléments du *dataset*.

D'autre part, la collecte d'évidence statistique suffisent pour supporter ces arguments nécessite l'utilisation extensive des ressources informatiques pour entraîner et évaluer plusieurs configurations de *dataset*s et algorithmes. e-Infrastructures tels que des clusters d'ordinateurs, grilles et, plus récemment, *clouds*, offrent vastes quantités de ressources, mais leur utilisation reste principalement liée à des outils et des méthodes d'accès spécifiques des fournisseurs, middleware, etc. Aussi, pour rassembler l'évidence précise, on a eu besoin de gérer agilement nombreuses configurations des algorithmes sur une diversité d'ensembles de données et des conditions d'entraînement, bien au-delà des capacités de la plupart des outils existant aujourd'hui. Pour surmonter cette difficulté, deux logiciels ont été développés dans cette thèse. D'un côté, BiomedTK (*Biomedical Data Analysis Toolkit*) facilite l'intégration des algorithmes, des *datasets* et l'analyse ROC, permettant l'exploration systématique de leurs configurations. De l'autre côté, C3 (*Cloud Computing Colonies Framework*) utilise les standards Java pour fournir une façon homogène d'accéder aux ressources informatiques distribuées sur des e-Infrastructures. C3 déploie des colonies de *Job Agents* sur les ressources disponibles et fournit un canal de communications unifiée pour leur envoyer des données et exécuter des applications, indépendamment de leur méthode d'accès réel.

Finalement, les résultats de cette thèse ont été appliqués avec succès dans l'obtention des classificateurs automatiques pour offrir un deuxième avis dans le diagnostic assisté du cancer du sein. En utilisant les résultats de cette thèse, et en étroite collaboration avec des radiologues spécialisés, des *datasets* ont été produites et analysées obtenant classificateurs prêts à être validés cliniquement pour être utilisés dans le diagnostic assisté et intégrés dans leurs logiciels, *workflows* médicaux et systèmes informatiques.

Contents

Acknowledgements

This thesis is the result of an enriching journey that has taken me through landscapes of beauty, complexity and hard work. Throughout the challenges and difficulties encountered, the fellow coming out of this path is not anymore the same one that started walking. It is the true personal experience, devoid of any fears, that shapes one's life, and the souls who cared to walk along with generosity and lightness are those to whom I am most grateful. This journey would simply have not been possible without the support of many people to whom I will always owe my gratitude.

I would like to thank Professor Miguel Ángel Guevara López for his support and guidance in this work, but above all, for his friendship through many years, even before this journey began. Just as today's present was wisely unexpected as seen from past times, I am now certain that the future ahead of us will remain so.

To Professor Eugénio Oliveira, for his understanding and advice on key moments of this journey, for allowing me to discover FEUP and for sharing with me his wise and pragmatic insights on the academic world.

To INEGI[1], CETA-CIEMAT [2] and FMUP-HSJ[3] for creating the possibility for this thesis to exist and for providing the resources, the people, administrative support and understanding on the many issues without which this thesis could not have been developed.

To the many people with whom I have worked in the last years with direct or indirect implication in this thesis work.

To the many anonymous reviewers of my papers, for their constructive comments.

Finally, and most importantly, to all those who make part of the human being writing these words, family, friends, companionships, professors, colleagues ... all those with whom our paths have come to cross to any extent, more intimate, more circumstantial, more colorful, more enlightening, more mundane, more enduring ...

I am all those people. I am indebted to them all.

[1] Instituto de Engenharia Mecânica e Gestão Industrial – Faculdade de Engenharia da Universidade do Porto.

[2] Centro Extremeño de Tecnologías Avanzadas.

[3] Hospital de São João – Faculdade de Medicina da Universidade do Porto,

List of figures

List of tables

Glossary

ANN	Artificial Neural Network
API	Application Programming Interface
AUC	Area Under the Receiver Operating Characteristic Curve
BCDR	Breast Cancer Digital Repository
BiomedTK	The Biomedical Data Analysis Toolkit
C3	The Cloud Computing Colonies Framework
CADe	Computer Aided Detection
CADx	Computer Aided Diagnosis
CC	Mammogram containing a craniocaudal image view of a breast
CDF	Cumulative Density Function
CETA-CIEMAT	Centro Extremeño de Tecnologías Avanzadas, Spain
DRI	Digital Repositories Infrastructure
FMUP-HSJ	Hospital de São João–Faculty of Medicine at University of Porto
FPR	False Positive Rate
FROC	Free Response ROC Curve
HPC	High Performance Computing
HTC	High Throughput Computing
IaaS	Infrastructure as a Service
INEGI	Instituto de Engenharia Mecânica e Gestão Industrial, Portugal
IMED	The project named *Developing Algorithms for Medical Image Analysis*
ML	Machine Learning
MLC	Machine Learning Classifier
MLO	Mammogram containing a mediolateral oblique image view of a breast
MLP	Multilayer Perceptron
PaaS	Platform as a Service
PDF	Probability Density Function
ROC	Receiver Operating Characteristic
ROCCH	ROC Convex Hull
ROI	Region of Interest
SaaS	Software as a Service
SOA	Service Oriented Architecture
SRM	Structural Risk Analysis
STL	Statistical Learning Theory
SVM	Support Vector Machine

TPR	True Positive Rate
VC	Vapnik-Chervonenkis dimension
VM	Virtual Machine

Chapter 1

Introduction

This chapter presents the motivation and objectives of this thesis and states its theoretical and technological contributions. But first, in order to better understand their relevance, it briefly describes the conceptual background within which the work leading to the results herewith presented has been developed.

A binary classification task aims at labeling elements of dataset into two classes (positive/negative) by assigning a score to each element and then applying a threshold. ROC analysis (Metz 1978; Fawcett 2006) is a tool for assessing the discrimination ability of a binary classifier for predictive purposes when applied to a scored dataset at all meaningful thresholds. It provides, therefore, a more comprehensive understanding of classifier performance than other metrics (*accuracy*, *precision*, etc.) which are taken at a single threshold level (Fawcett and Provost 1997; Provost, Fawcett et al. 1998). Used primarily in signal detection theory (Egan 1975) to determine if an electronic receiver is able to satisfactory distinguish between signal and noise, usage of ROC analysis has reached different domains in medical decision making such as diagnostic systems, medical data mining, medical imaging, etc. (Swets 1988; Zweig and Campbell 1993; Hanley 1996; Swets, Dawes et al. 2000; Metz 2008; Iavindrasana, Cohen et al. 2009). In machine learning (ML), it is used since the early works of (Spackman 1989) mostly to evaluate and compare classifier performance (Bradley 1997; Fawcett 2003).

ML algorithms, such as for training multilayer perceptrons (MLP) or support vector machines (SVM), are typically designed to minimize some error rate, which measures

how far the score assigned by a classifier to each dataset element falls from its ideal value. Although related, literature reports sound evidence that error rate minimization does not necessarily yield to AUC optimization (Cortes and Mohri 2004) and, therefore, problems focused on ROC analysis are only partially addressed by existing ML algorithms, aimed at reducing error rates. Some examples of ML algorithms that have been devised for AUC optimization can be found at (Ferri, Flach et al. 2002; Rakotomamonjy 2004; Brefeld and Scheffer 2005; Calders and Jaroszewicz 2007; Castro and Braga 2008; Marrocco, Duin et al. 2008; Pahikkala, Airola et al. 2008; Takenouchi and Eguchi 2009). However, it would be desirable to be able to reuse for AUC optimization the vast amount of ML techniques developed to date and, rather than developing new ML algorithms, this thesis aimed at introducing AUC optimization into existing ones in an affordable manner, providing both the appropriate theoretical foundation and material tools to do so with a minimal impact in their implementation. The approach chosen for this aim has been (1) to define a general ROC based error measure to substitute traditional error rate measures, so that the core logic of existing ML algorithms remains intact requiring only replacing the error calculation methods; and (2) applying and validating it with different kinds of MLP training algorithms to show its effectiveness. This constitutes the central contribution of this thesis.

The term *eInfrastructures* refers to large amounts of distributed computing resources that are made available in a homogeneous manner to user communities. These include computing resources delivering CPU time or storage space, such as those typically found in data centers. An eInfrastructure is therefore regularly composed by a federation of data centers aggregating their computing resources to offer a large facility service. The affordability of computer hardware has led to a proliferation of data centers of many sizes and, since almost a decade now, both governments and private institutions are devoting a considerable amount of effort and funding to building trans-national eInfrastructures. Their generalized usage is to take academia and industry to new levels of reach in their research and production endeavors. However, their adoption has been slower than expected due to many factors (complexity of access tools, interoperability issues, cost of refactoring software, etc.) and there is a wide range of fields and specific problems that seldom benefit from eInfrastructures as much as they could. Machine learning is no exception to this and during the development of this thesis we faced the challenge to exploit eInfrastructures to validate our results. The technology developed to overcome the difficulties encountered constitutes the core of the technological contributions of this thesis.

From a global perspective, this thesis has been developed to address *classification problems* through the extensive usage of ROC analysis with machine learning classifiers based on MLPs (including the ones optimized for AUC) and SVMs supported on eInfrastructures. The practical utility of this thesis has been demonstrated in the fields of image analysis and pattern recognition for breast cancer CAD.

1.1 Background

This section provides the basic insight into the fields in whose intersection this thesis has been developed: Receiver Operating Characteristic analysis and machine learning, the usage of eInfrastructures for scientific production and Computer Aided Diagnosis for breast cancer.

1.1.1 Receiver Operating Characteristic (ROC) analysis

1.1.1.1 Performance metrics

Whenever a binary classifier or test is applied to a dataset a series of measures are produced to assess its performance and, therefore its efficacy. Figure 1 shows the fundamental metrics obtained after the elements of dataset have been classified. Particular values are typically obtained by a classifier that assigns scores to each dataset element and then sets a threshold level above which elements are classified (predicted) to be positive and the rest are classified as negative.

		actual value	
		P	**N**
classifier prediction	p	true positive (TP)	false positive (FN)
	n	false negative (FN)	true negative (TN)

P	Number of positive elements
N	Number of negative elements
p	Number of elements classified as positive
n	Number of elements classified as negative
TP	Number of positive elements classified as positive
TN	Number of negative elements classified as negative
FP	Number of negative elements classified as positive (Type I error)
FN	Number of positive elements classified as negative (Type II error)

Figure 1: Distribution of classified dataset elements

A diversity of measures derived from figure 1 are used in different fields according to particular interests. Some of these measures are shown in table 1. For instance, the Information Retrieval community usually deals with *Precision* and *Recall* (Manning,

Raghavan et al. 2008) whereas the signal processing and biomedical communities deal mostly with the TPR (True Positive Rate) and FPR (False Positive Rate) (Fawcett 2003) which somehow encompass all the rest.

Table 1: Threshold classifier metrics

$True\ Postive\ Rate\ (TPR)$	$= \frac{TP}{P} = \frac{TP}{TP + FN}$
$False\ Postive\ Rate\ (FPR)$	$= \frac{FP}{N} = \frac{FP}{FP + TN}$
$Specificity$	$= \frac{TN}{N} = \frac{TN}{FP + TN} = 1 - FPR$
$Accuracy$	$= \frac{TN + TP}{P + N}$
$Error\ Rate$	$= 1 - Accuracy$
$Positive\ Predictive\ Value\ (PPV)$	$= \frac{TP}{TP + FP}$
$Sensitivity = Recall$	$= True\ Postive\ Rate\ (TPR)$
$Precision$	$= Positive\ Predictive\ Value\ (PPV)$
$F\ measure$	$= \frac{2}{1/precision + 1/recall}$

Metrics in table 1 are known as *threshold metrics* since they are obtained after a threshold has been applied to a scored dataset. In addition, other kinds of metrics can be considered (Caruana and Mizil 2006) such as rank metrics (AUC –as defined in next section–, average precision, precision/recall break point, etc.) or probability metrics (squared error, cross entropy, etc.)

1.1.1.2 ROC Curves

In this work, we assume that element scores assigned by any classifier fall within the [0,1] interval. By varying the threshold level used by the binary classifier throughout such interval, we obtain different values for the above metrics. In specific, we obtain different TPR and FPR levels which can then be plotted in a ROC graph. If we start with a maximum threshold (at value 1) all elements are classified as negative therefore we have neither false positives nor true positives, represented as a point at the (0,0) coordinate on the graph. At a minimum threshold (value 0) all elements are classified as positive and TPR and FPR reach their maximum levels, represented as a point at

the (1,1) coordinate on the plot. As we gradually increase the threshold, each time a new dataset element falls below it, it generates a new TPR and FPR and therefore a new point in the ROC graph. Figure 2 reproduces an example in (Fawcett 2006). The ROC "curve" on the right is created by thresholding the dataset on the left. The dataset contains 20 elements and the table shows the score assigned by a classifier to each one along with its actual class. Each point in the ROC graph shows the threshold value that produces it and corresponds to a different FPR-TPR value pair. For instance, a threshold of 0.54 classifies elements 1 to 6 as positive and the rest as negative, producing a TPR=0.5 (out of the 10 positive elements of the dataset, 5 were classified as positive) and a FPR=0.1 (out of the 10 negative elements, 1 was classified as positive).

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1

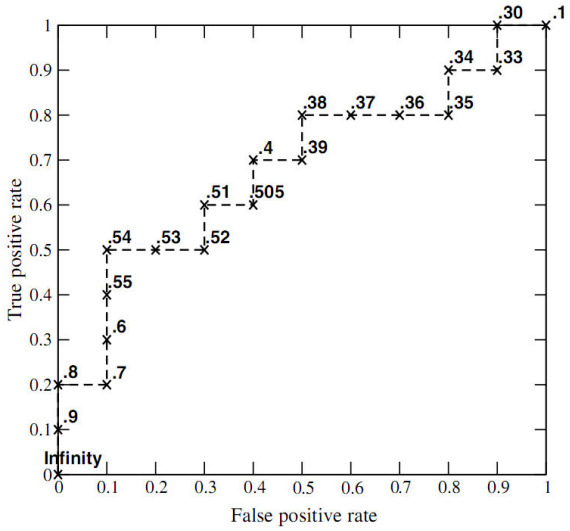


Figure 2: Example of ROC graph.

We are therefore in general interested in classifiers yielding ROC graphs bowing to the top left corner. One of the useful properties of ROC graphs is that they are insensitive to class skew and unbalanced datasets distributions (having, for instance, more positive elements than negative).

The Area Under the ROC Curve (abbreviated by AUC, as used in this thesis, or also Az) is then taken as a single measure to compare different classifiers, seeking classifiers having greater AUC. A fundamental result is that the AUC is the probability that a randomly selected positive element is ranked higher than a randomly selected negative element (Hanley and McNeil 1982) which corresponds to the Wilcoxon statistic (Wilcoxon 1945) or the Mann-Whitney test (Mann and Whitney 1947) as given by the following formula:

$$AUC(S,h) = \frac{\sum_{p \in S_P} \sum_{n \in S_N} \mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P| \cdot |S_N|} \quad (1.1)$$

which measures the empirical AUC of classifier h when used to classify dataset $S = S_P \cup S_N$ composed of positive instances (S_P) and negative instances (S_N) having $S_P \cap S_N = \emptyset$. Then, $h_{sc}(x)$ represents the classifier output (score) upon dataset element x and $\mathbf{1}[h_{sc}(n) < h_{sc}(p)]$ denotes the *indicator function*, yielding the value 1 if $h_{sc}(n) < h_{sc}(p)$ and 0 otherwise. Basically, for each positive element equation 1.1 counts how many negative elements have a lower score. A random classifier yields an AUC value of 0.5 (the probability for a randomly selected positive element to be ranked higher than a randomly selected negative element is the same as the probability to be ranked lower) which is represented in a ROC graph as the diagonal joining the 0,0 and 1,1 points in the FPR-TPR space, such as the dashed lines shown in figure 3.

From a statistical point of view, the scores of the positive and negative elements of a dataset can be considered to be drawn from different distributions whose probability density functions (PDF) are denoted by f^+ and f^-, and their respective cumulative density functions (CDF) are denoted by F^+ and F^-. Then, since we are assuming that the scores assigned by a classifier to dataset elements fall within the $[0,1]$ interval, the AUC can be defined in the continuous domain for known distributions as follows:

$$AUC(h,S) = \int_0^1 f^+(x) \cdot F^-(x) dx \quad (1.2)$$

which is the continuous equivalent to equation 1.1 and can be interpreted in the same way: for each point in the score space, whose probability of being a positive is given by $f^+(x)$, equation 1.2 accounts for the probability of having a negative element below it, which corresponds to the CDF of negative elements, $F^-(x)$. Since $f^+(x)$ and $f^-(x)$ are rarely known we are bound to use the empirical AUC in equation 1.1 on a limited amount of available samples (datasets) through statistical and machine learning methods. This formulation sets the foundation for further treatment of AUC, such as to obtain smooth curves (see figure 3), confidence intervals, assess their statistical significance, etc. (Hanley 1996; Jensen, Müller et al. 2000; Faraggi and Reiser 2002; Sorribas, March et al. 2002; Agarwal, Graepel et al. 2005; Macskassy, Provost et al. 2005; He, Lyness et al. 2009; Hanczar, Hua et al. 2010)

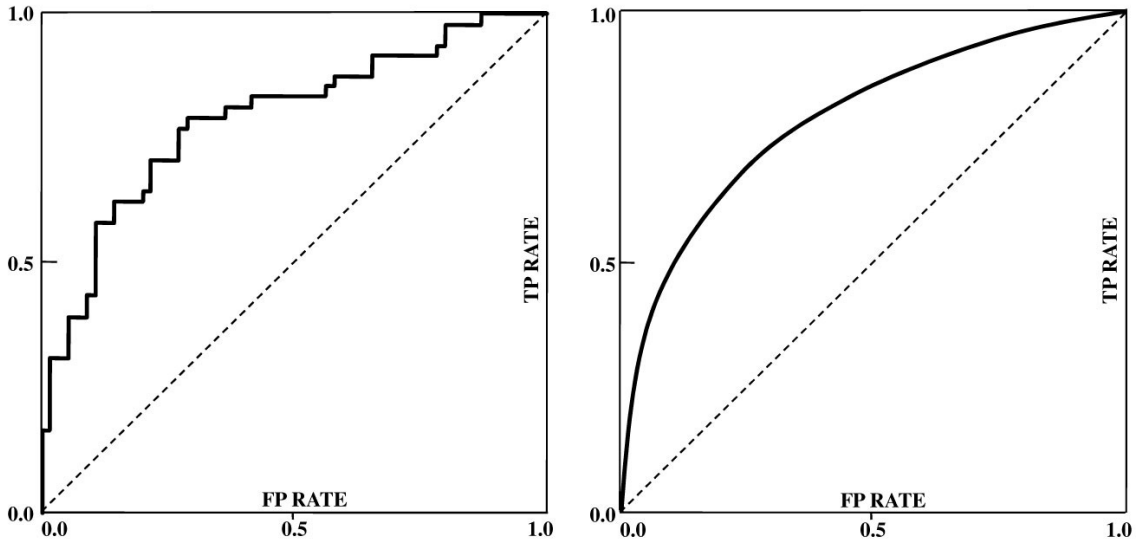


Figure 3: Smoothed ROC graph**.** ROC graph directly obtained by varying the threshold on dataset elements (left). Smoothed version for further mathematical treatment (right).

In addition, in medical imaging, a variation of ROC analysis is often also used when diagnosing lesions appearing on images. Free-Response Receiver Operating Characteristic analysis (FROC) (Bandos, Rockette et al. 2009) copes with cases where the same image might have several lesions or markings which might bias the AUC measure as explained above. A FROC curve plots the lesion localization fraction (LLF) against the non-lesion localization fraction (NLF). LLF is defined as the number of lesions detected divided by the number of total lesions, whereas NLF is the number of false lesions identified (false positives) divided by the total number of images processed.

Observe both from equation 1.1 and figure 2 the need to sort the dataset elements in order to compute the AUC, which makes it computationally expensive. From this perspective, an additional contribution of this thesis is on devising an efficient procedure to approximate the AUC of a score dataset with arbitrary precision, so that the user is able to select the maximum error incurred by the approximation whilst keeping the procedure computationally efficient.

1.1.2 Machine learning

Machine learning (ML) is about programming computers to improve some performance measure using example data or past experience. The definition in (Mitchell 1997) has become a widely used standpoint:

> ***Learning**: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*

"*Data*" or "*Experience*", in a broad sense, is in practice handled through *datasets* composed of *data elements*, which represent the specific objects presumed to contain the knowledge target for the learning task. These *elements* may be vectors with numeric features, annotated images, sound patterns, etc. Datasets are often split to form *train datasets,* used as input for ML algorithms, and *test datasets*, used to measure the generalization capability on unseen data. Section 1.1.1.1 above showed commonly used performance measures.

Many other fields in Artificial Intelligence and other areas overlap or fall under such broad definition, and a considerable wealth of methods and techniques are shared to different extents throughout all of them. It is intentionally left undefined what field contains what other field, since the literature shows differences in terminology across researchers and communities in this respect. For instance, *statistics* aims at understanding the phenomena that have generated the data, often with the goal of testing different hypotheses about those phenomena. *Data mining* aims at finding patterns in the data so that they are understandable by people. Even, *psychological studies* of human learning aim at understanding the mechanisms underlying the various learning behaviors exhibited by humans (concept learning, skill acquisition, strategy change, etc.). In contrast, ML is primarily concerned with the accuracy and effectiveness of the resulting computer systems.

The field of *pattern recognition* has historically produced many of the methods and applications used and addressed by ML. As defined in (Jain, Duin et al. 2000):

> *Pattern recognition is the study of how machines can observe the environment, learn to distinguish patterns of interest from their background, and make sound and reasonable decisions about the categories of the patterns.*

In order to later position appropriately the contributions resulting from this thesis, the following paragraphs describe briefly the kinds of problems addressed by ML, the methods and techniques mostly used and, finally, provide basic insight into pattern recognition processes for breast cancer CAD.

1.1.2.1 Learning tasks in ML

Whenever facing an ML problem (learning from data) it is said that the machine is about to carry out a *learning task*. Most of the problems addressed by ML can be catalogued within the following types, according to the nature of the learning task in hand, the data available and the learning goal. Detailed information and examples can be reviewed in (Mitchell 1997) and (Alpaydin 2010).

Supervised learning: Each element of a training dataset is given along with the desired output expected from a classifier. ML algorithms use this *a priori* knowledge to guide their learning processes and the output of ML classifiers must fall within the predefined classes or ranges. **Classification** is the supervised learning task where the allowed classifier outputs can only be from within a finite set of discrete class labels. A *Binary classification* problem is one where only two classes are allowed. **Regression** is the supervised learning task where the classifier is made of a continuous function and, therefore, its output can be any real number within a range. An example of classification is the problem of *credit score* (Hand 1998) where customers must be assigned to a set of classes (such as *low-risk*, *medium-risk*, *high-risk*) so that a decision on whether a loan is granted or not can be made based on customer data (income, savings, age, financial history, etc.). If we are looking for a ML system that can predict the price of a car based on its attributes (year, engine capacity, brand, mileage, etc.) then we are facing a regression problem. In addition, **cost-sensitive classification** (Elkan 2001) allows to take into account classification problems where different misclassification errors incur different penalties such as, for instance, the cost of a false-negative being much higher than the cost of a false-positive. See (Caruana and Mizil 2006) for further information on supervised learning.

Unsupervised learning: The training dataset is given with no class information and ML algorithms are to discover classes, patterns or relations in the data with no additional a priori knowledge. In statistics this is called *density estimation*. (Silverman 1986). Many unsupervised learning methods are based on the idea of **clustering**, where the aim is to find clusters or grouping of inputs through some similarity or distance metric. These include Independent Component Analysis (ICA), K-means, Factor Analysis, etc. See (Ghahramani 2004) for further information.

Reinforcement learning: When the output of a system is a sequence of actions, ML aims at optimizing the policy by which those actions are generated. In this sense,

ML methods should be able to learn from past good and bad sequences of actions in order to improve its action generating policy. In many cases, actions produce rewards or penalties through which actions and sequences of actions are assessed. *Game playing* is a good example for applying reinforcement learning, where a move is undertaken if it is part of a good policy for winning the game. Robot navigation is another good example.

1.1.2.2 Supervised Learning

Figure 4 below shows the general set up for supervised machine learning, as described in (Alpaydin 2010) and (Hastie, Tibshirani et al. 2009). Starting off from a dataset containing the experience to be learned from, an *ML Method* (such as Artificial Neural Networks (ANN), Support Vector Machines (SVM), Decision Trees, Bayesian Networks, etc.) undergoes a three step process to produce an *ML Model*, containing the results of the learning process so that it can be used in new prediction tasks.

1. **Model Fitting**: Encompasses the actual learning process that, starting from example data and a set of parameters specific to the *ML Method* used produces an *ML Model*. In general, this stage is performed many times for different sets of parameters producing several *ML Models*.
2. **Model Selection**: Estimates the performance of different *ML Models* in order to choose the best one.
3. **Model Assessment**: Having chosen a final model, it estimates its generalization error on new data.

From a statistical point of view, data can be viewed as random variables X and Y representing the input vectors and target (desired) outputs respectively, related through an unknown relation $f(X) = Y$. An *ML Model* is then a function $\hat{f}(X)$ representing a prediction model estimated from training data $\mathcal{T}$ obtained from sampling X and Y. The model $\hat{f}(X)$ is our estimation of the unknown underlying function $f(X)$. In classification tasks (such as the ones object of this thesis) an *ML Model* is referred to as an **ML Classifier** (MLC) and so is mostly used throughout the rest of this text.

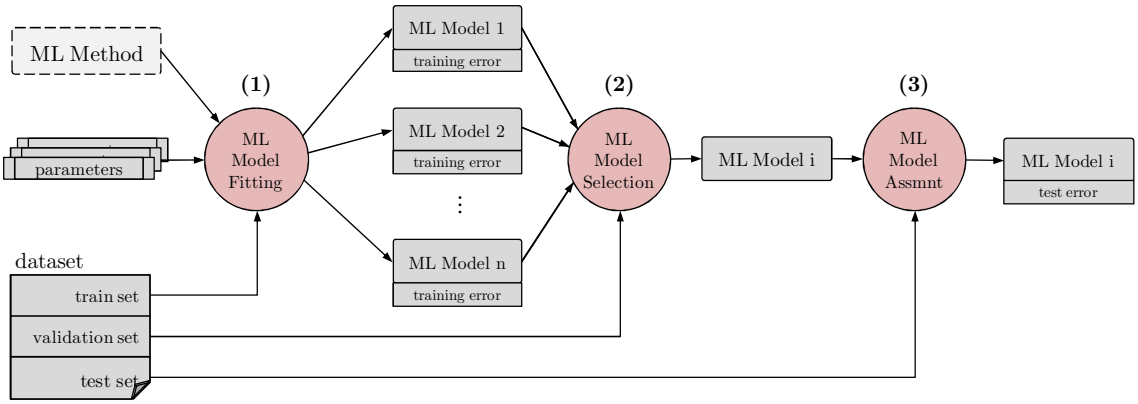


Figure 4: Elements of machine learning

A loss function is defined to assess model performance in the sense of the definition of learning given in page 8 measuring the error between $Y = f(X)$ and $\hat{f}(X)$, denoted by $L\left(Y, \hat{f}(X)\right)$. Typical choices for L are:

$$L\left(Y, \hat{f}(X)\right) = \begin{cases} \left(Y - \hat{f}(X)\right)^2 & squared\ error \\ \left|Y - \hat{f}(X)\right| & absolute\ error \end{cases} \tag{1.3}$$

In fact, one of the contributions of this thesis is on the usage of AUC as a loss function. Then, two error measures are defined and used at different stages. The *training error* (also named *empirical risk*) is the average loss over the training sample

$$\overline{err} = \frac{1}{N}\sum_{i=0}^{N} L\left(y_i, \hat{f}(x_i)\right) \tag{1.4}$$

The *test error*, also known as *generalization error* or *true risk*, is the *prediction error* over an independent test sample of the function $\hat{f}(X)$ as estimated over training data $\mathcal{T}$.

$$Err_{\mathcal{T}} = E\left[L\left(Y, \hat{f}(X)\right)|\mathcal{T}\right]$$

Where X and Y are drawn randomly from their joint distribution (population).

In a data rich situation, the best approach to ensure independence of the three stages depicted in figure 4 is to split the available data into three parts: a train set for model fitting, a validation set for model selection and a test set for model assessment. It is difficult to give a general rule for dataset splitting in this sense, since it is highly influenced by the signal-to-noise ratio of the data and the dataset size. A typical split might use 50% of data for model fitting (training), 25% for model selection (validation) and 25% for model assessment.

However, more than often there is not enough data for such split and the stages above need to be approximated either analytically or by efficient sample reuse. In both cases, the goal is to estimate the generalization error from the available data. Analytical methods are mostly based on the observation that increased model complexity tends to overfit training data while failing to generalize on test (unseen) data as illustrated in figure 5, obtained by (Hastie, Tibshirani et al. 2009).

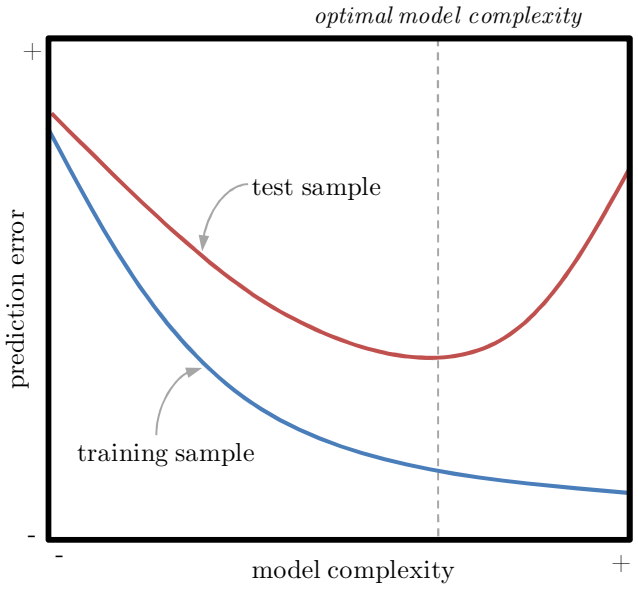


Figure 5: Test and train data vs. model complexity

Measuring model complexity is not straight forward. Using the number of parameters required by a model as a metric has been a first choice, but this has shown to be insufficient in many cases. In fact, rather than an absolute measure, it is thought that model complexity should somehow match the complexity of the data in hand. Different measures for model complexity have been devised in this sense, so that later can be used to estimate the generalization error from training data and improve model selection. Among them: the Bayesian Information Criterion (BIC) (Schwarz 1978), the Akaike Information Criterion (AIC) (Akaike 1974), the Minimum Description Length (MDL) (Rissanen 1983) and the VC (Vapnik-Chervonenkis) dimension (Vapnik 1998).

In addition, when data is scarce sampling methods can also be employed to reuse available data more efficiently. Among them, cross-validation and bootstrapping are widely used (Efron and Gong 1983; Kim 2009). In cross-validation, the generalization error is directly estimated by splitting the data in to k parts repeating the process above k times, each one using a different part of the split dataset for testing a model that has been fitted by using the remaining $k-1$ parts. Performance measures are then

averaged and a final estimate is given. Leave one out cross validation (LOOCV) refers to cross validation when k is equal to the number of elements in the dataset, and it is known to give an unbiased estimator of the generalization error (Efron 1983) at the expense of high variance and high computational cost (one fitting process per dataset element). For cross-validation, a widely used value for k is 10. On the other side, bootstrapping methods sample the dataset for training and testing with replacement.

The following two sections describe briefly the multilayer perceptrons (MLPs) and the support vector machines (SVMs) machine learning methods. A variety of training algorithms for MLPs are used in this research to validate experimentally its theoretical contributions. Then, together with SVMs, they are used extensively throughout the technological contributions herewith presented and their practical application in breast cancer CAD. Both MLPs and SVMs represent two significantly different approaches to supervised learning (kernel based and biologically inspired) and there is a wealth of literature and implementations upon which the work of this thesis can be solidly founded.

1.1.2.3 Multilayer perceptrons

A multilayer perceptron (MLP) is a feed forward artificial neural network model consisting of multiple layers of nodes in a directed graph which is fully connected from one layer to the next one (figure 6). The first layer constitutes the one accepting the input to the network and the last layer produces the output response. Nodes in intermediate layers are neurons with a non-linear activation function (such as tangential or sigmoid).

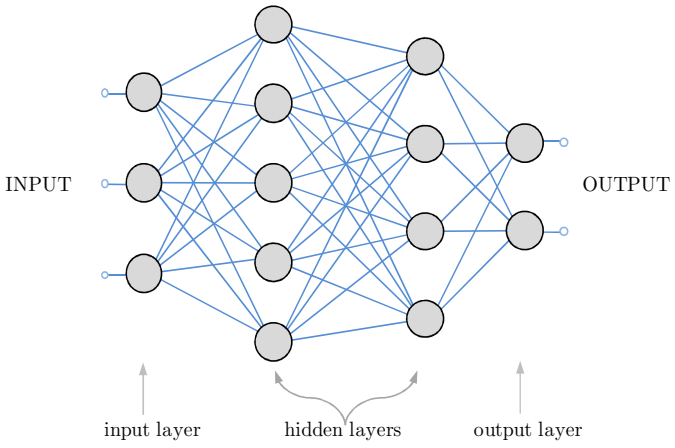


Figure 6: A mutilayer perceptron

MLPs are trained with a variety of ML algorithms, being feed forward backpropagation the most used one by far. As an example, table 2 below shows some of the algorithms to train MLPs included in the Encog toolkit (Heaton 2010), a well known implementation for ANNs.

Table 2: Some ML algorithms for multilayer perceptrons

Feed Forward Back Propagation (FFBP)	Per-element error measures at each output neuron are used to adjust neuron weights of the various layers of the MLP backwards from the output layer to the input layer, through a gradient descent method controlled by two user definable parameters: the *learning rate* and the *momentum*.
Feed Forward Resilient Propagation (FFRP)	A variation of FFBP where each neuron has its own set of independent parameters to control the gradient descent (similar to the FFBP learning rate and momentum) that the algorithm adjusts automatically throughout the training process
Feed Forward Simulated Annealing (FFSA)	The neuron weights of the MLP are taken through several "cooling" cycles. Starting at an initial top temperature, at each step in each cooling cycle the MLP weights are randomized according to the temperature (higher temperatures produce higher random variability) generating a new MLP. If the new MLP produces a lower error on the whole dataset, it is kept to the next cooling step. Otherwise it is discarded. Then, the temperature is lowered one step and the process continues. The user definable parameters it accepts are *start-temperature*, *end-temperature* and *number-of-cycles*.
Feed Forward Genetic Algorithms (FFGA)	The vector of MLP neuron weights is interpreted as a chromosome and a population of MLPs with identical structure and different weights is evolved through generations that mate and cross over. MLPs (chromosomes) yielding lower errors on the whole dataset are considered as best suited and, therefore, with a higher probability of survival and mating to the next generation. The user definable parameters it accepts are *population-size*, *mutation-percent* and *percent-to-mate-with*.

1.1.2.4 Support vector machines

Support Vector Machines (SVMs) can be seen as a learning technique that originated from the theoretical foundations of statistical learning theory (STL) (Vapnik 1998) and structural risk minimization (SRM) (Vapnik and Chervonenkis 1974). STL is a theoretical approach to understanding learning and the ability of learning machines to generalize, whereas SRM is an inductive principle to assess the choice of a learning model (machine) seeking those ones whose complexity is appropriate to describe the training data. In the simplest classification tasks, SVMs use a linear separating hyperplane to create a classifier with a maximal margin to discriminate between two classes (figure 7, right). When the classes cannot be linearly separated in the original input space (figure 7, left) as happens in virtually all practical problems, the SVM transforms it into a higher dimensional feature space where the classes might be separated. This is achieved by using a non-linear map (φ) such as a polynomial, sigmoidal, radial basis functions, etc. as shown in figure 7. The resulting linear hyperplane in the new higher dimensional feature space will be optimal in the sense of being a maximal margin classifier with respect to training data. This maximality condition is expected to convey good generalization properties on unseen (test) data. Additional theoretical instruments are added to this set up to cope with outliers and noise by allowing error bands when finding the maximal margin hyperplane.

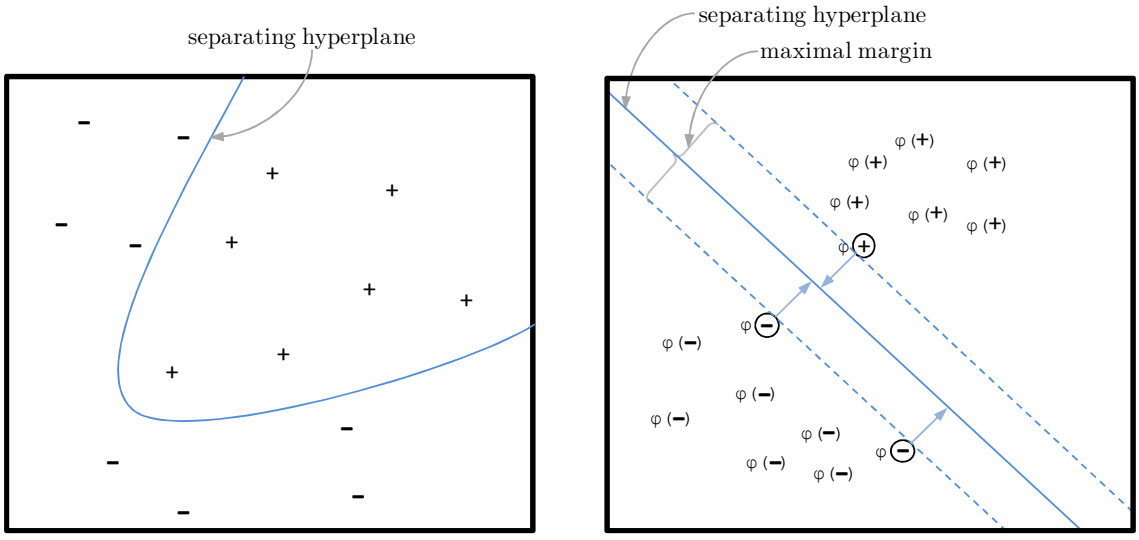


Figure 7: Separating hyperplanes on a Support Vector Machine. On the original input space (left) and on the higher dimensional feature space once the feature mapping (φ) has been applied (right). The circled dataset elements are the support vectors.

Choosing the right mapping (φ) or *kernel* function is therefore a key issue. The VC (Vapnik-Chervonenkis) dimension (Vapnik 1998) is a property of a set of approximating functions (such as polynomials, sigmoids or even artificial neural networks) which can be interpreted as a measure of the complexity of a family of functions with respect to a set of data points. Then, the ideas behind SRM are used to choose, from a set candidate models (learning machines), the one whose complexity is appropriate to describe the training data. With SVMs, this is done by minimizing both the VC dimension and the training error to tune the parameters of the kernel function chosen.

SVMs can be seen as part of a larger class of machine learning techniques called *kernel-based methods* where this idea of kernel substitution is applied to a wide range of data analysis methods, such as Fisher discriminant analysis or least squared approaches. These methods are used for supervised learning, but kernel substitutions can also be used in unsupervised scenarios such as *kernel PCA* (Principal Component Analysis) and *kernel CCA* (Canonical Correlation Analysis). Refer to (Campbell and Ying 2011) for detailed information.

1.1.3 eInfrastructures

The possibility offered by today's computing resources to develop science and engineering into new realms is well established. According to (NASA, NSF et al. 2008):

> *"Simulation Based Engineering and Science (SBE&S) today has reached a level of predictive capability that it now firmly complements the traditional pillars of theory and experimentation/observation. As a result, computer simulation is more pervasive today – and having more impact– than at any other time in human history. Many critical technologies, including those to develop new energy sources and to shift the cost-benefit factors in healthcare, are on the horizon that cannot be understood, developed, or utilized without simulation"*

This, together with the fact that computing resources are becoming rapidly cheaper and available, has drawn lots of efforts from all kind of institutions in the process of integrating distributed and dispersed computing resources to build *eInfrastructures* across the world with the aim of enabling science and engineering to use this new tool as a fundamental part of their development, without which such development would simply not exist.

The charts in figure 8 show the evolution of the supercomputers taking part in the **TOP500** (Top500 2011), which every six months lists the 500 most powerful supercomputers of the world. Each chart shows, historically, the nature of the supercomputers in the list according to different criteria.

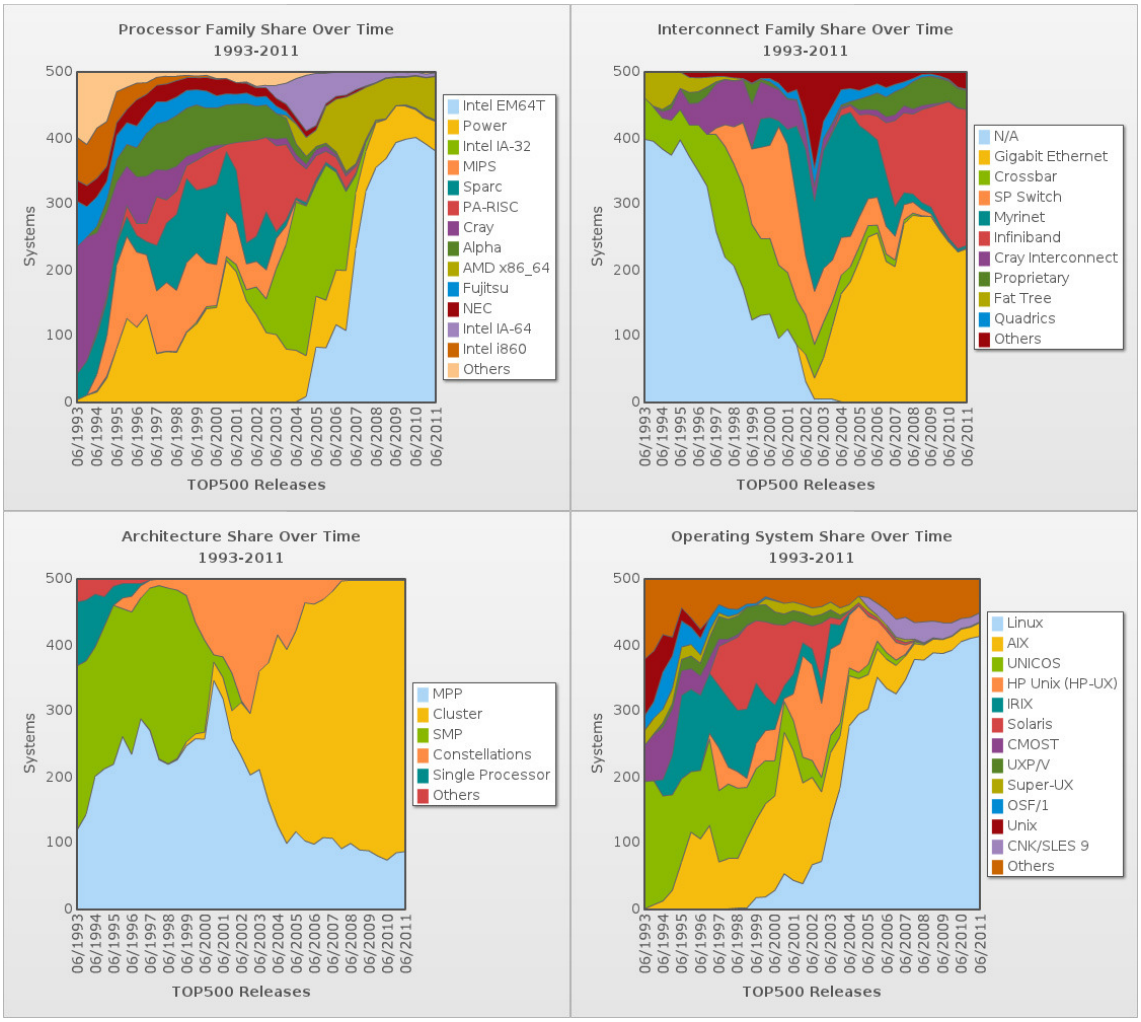


Figure 8: Historical distribution of the TOP 500 by hardware and OS.

As it can be seen, the supercomputers taking part on the list are increasingly of similar nature as more than 80% of the TOP 500 supercomputers are devised as cluster architectures, interconnected with Infiniband or Gigabit Ethernet, with Intel or AMD 64bit processors, run by Linux operating systems. This means that computer centers are becoming more a *commodity*, being built out of hardware widely available. It is straight forward to infer that, outside the TOP 500, this picture lies at the root of the current proliferation of computer centers of all sizes across the globe.

These two issues: the usage of computing resources as a fundamental part of scientific and engineering developments and the proliferation of computer centers, is

one major reason behind the sustained funding and efforts to build eInfrastructures happening in different parts of the world.

eInfrastructures are made available through a certain computing model and access method. According to these, three major kinds of eInfrastructures are available today: Grids, Clouds and Supercomputers. Each one of them is addressed to different kinds of problems and users and is based on different underlying technologies, etc. as summarized in table 3.

Table 3: Comparison of eInfrastructures

	Grids	**Clouds**	**Supercomputers**
Computing Model	Batch uncomunicating jobs	Virtual machines decoupled from physical infrastructure	Batch intercommunicating jobs
Access Method	Middleware to federate distributed storage and batch queue systems	Virtualization	Batch queue systems
Technologies	gLite, Globus, UNICORE	VMWare, E2C, OpenNebula, Eucalyptus	SGE, PBS, LSF
Target applications	Sequential algorithms parallelizable by parameter sweeps or data partitioning	Whatever can be encapsulated within a virtual machine. IaaS, PaaS, SaaS	Parallelizable algorithms
Current users	Mostly academia	Mostly industry	Academia & Industry

1.1.4 Breast Cancer CAD

Machine learning methods produce models that can explain complex relationships in the data and, therefore, they seem suitable for biomedical data analysis, often consisting on high dimensional quantitative data provided by the state-of-the-art medical imaging and high-throughput biology technologies. The general strategy relies on expert-curated ground truth datasets that provide the categorical associations of all available data samples, constituting the basis for supervised learning as explained in previous subsections.

In statistical pattern recognition (Jain, Duin et al. 2000) a pattern is represented by a set of d features, or attributes and viewed as a d-dimensional feature vector. In a preprocessing stage, the pattern of interest is segmented out from the background, noise is removed, the pattern is normalized, etc. Then, a set of features is

extracted/computed representing the segmented pattern and a classifier is trained to partition the feature space. When an appropriate classifier has been found, then it can be applied to features vectors to provide automatic classification of the pattern. This process can be undertaken in a supervised or unsupervised manner, depending on the nature of the problem in hand.

Breast Cancer CAD is a supervised pattern recognition task. With some ambiguity, the literature uses the CAD term to refer both to Computer Aided Detection (CADe) and the Computer Aided Diagnosis (CADx). While CADe is concerned with locating suspicious regions within a certain medical image (such a mammogram), CADx is concerned with offering a diagnosis to a previously located region. In general, starting from a digital mammogram, the CAD process is performed through the following stages, as illustrated in figure 9:

1. ***Region of Interest (ROI) selection:*** the specific image region where the lesion or abnormality is suspected to be (which can be manual, semiautomatic or automatically selected).
2. ***Image Preprocessing:*** the ROI pixels are enhanced so that, in general, noise is reduced and image details are enhanced.
3. ***Segmentation:*** the suspected lesion or abnormality is marked out and separated from the rest of the ROI by identifying its contour or a pixels region. Segmentation can be fully automatic (the CAD system determines the segmented region), manual or semi-automatic, where the user segments the region assisted by the computer through some interactive technique such as deformable models (Chenyang and Prince 1998) or intelligent scissors (livewire) (Liang, McInerney et al. 2006).
4. ***Features Extraction***: quantitative measures (features) of different nature are extracted out from the segmented region to produce a features vector. These might include representative measures of the image region statistics *(skewness, kurtosis, perimeter, area, etc.)*, shape *(elongation, roughness, etc.)* and texture *(contrast, entropy, etc.)*
5. ***Automatic Classification:*** this last step is the one that finally offers a diagnostic to be used as a second opinion, by assigning the vector of extracted features to a certain class, corresponding to a lesion type and/or a benignancy/malignancy status.

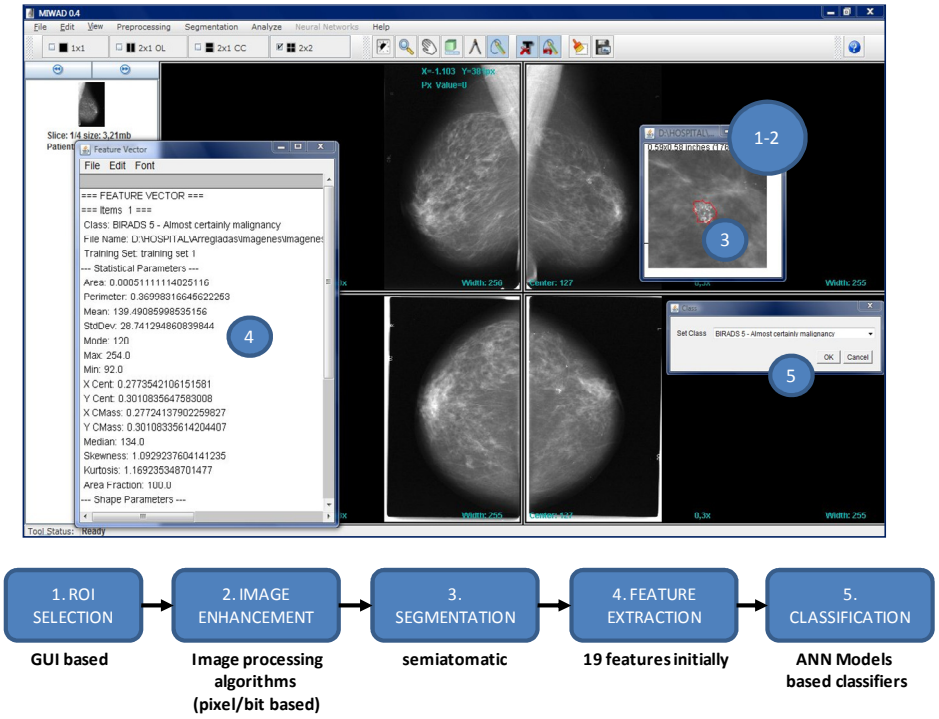


Figure 9: Breast Cancer CAD cycle

1.2 Motivation and objectives

The present PhD work emerges in the context of the IMED Project (for *Development of Algorithms for Medical Image Analysis*) being carried out between INEGI (*Instituto de Engenharia Mecânica e Gestão Industrial*, at University of Porto, Portugal) and CETA-CIEMAT (*Centro Extremeño de Tecnologías Avanzadas*, Ministry of Science and Innovation, Trujillo, Spain) since 2008 where a mammograms repository is being built and deployed at FMUP-HSJ (*Hospital de São João–Faculty of Medicine* at University of Porto, Portugal) and CAD methods are being developed by exploiting the data collected in the repositories. The project is founded on preliminary works to use Grid infrastructures for hosting medical image repositories (Calanducci, Ramos-Pollan et al. 2008; Barbera, Ramos-Pollan et al. 2009), retrieving and analyzing data (Ramos-Pollan and Barreiro 2009; Risk, Ramos-Pollan et al. 2009), but soon focused on medical imaging for breast cancer to deliver software platforms that include graphical user interfaces (for manipulating and classifying mammograms) and interfaces to Grid storage and local resources for managing the image repository (Ramos-Pollan, Guevara López et al. 2009). Through the IMED project, a pilot platform was deployed at FMUP allowing:

1. Feeding the data repository with scanned or digital mammograms and patient clinical data
2. Having specialized radiologists segmenting regions in mammograms (through a graphical user interface, see figure 9) and issuing a standardized diagnosis.
3. Building training datasets that include specialists' assessment, patient information and features extracted from segmented regions.

Confronted with the necessity to find well performing ML classifiers for the constructed datasets and offer them to radiologists as second opinion assessment to be used in their diagnosis and patient management decisions, the project team identified the need to (1) systematically use ML for AUC optimization as required by the medical environment within which the project is being developed, specially for multilayer perceptrons as they are commonly used in medical image analysis and (2) exploit eInfrastructures to perform massive explorations of ML classifiers configurations in an agile manner as data was being generated by specialized radiologists. Soon, it became evident that results in these aims could be beneficial for ML in general and therefore a thesis project was devised to channel the work to be performed and herewith described. This led to establishing the following objectives for this thesis:

> **Objective 1**: to provide a general formulation that integrates AUC optimization in existing ML algorithms in an affordable manner (with minimal programming effort and impact in source code), and validate it with different training algorithms on multilayer perceptrons based classifiers.
>
> **Objective 2**: to enable the usage of eInfrastructures for (1) the exploration of the search space of possible configurations of ML algorithms with different datasets and AUC metrics, and (2) the validation of new ML algorithms and classification methods (in particular, the multilayer perceptron based classifiers mentioned in Objective 1).
>
> **Objective 3**: to demonstrate the usefulness of the above results in the area of breast cancer CADx

With respect to the IMED project, the results of this thesis constitute stage 5 as described in figure 9, receiving as input the datasets containing the extracted features and the diagnosis information given by the specialized radiologists through the usage of the project tools; and providing as output a set of trained ML classifiers to be

integrated within the graphical workstations for delivering second opinion assessment to radiologists.

1.3 Thesis statement

The above objectives and consequent work was therefore carried out to prove the following hypothesis:

> **Hypothesis 1**: Multilayer perceptrons can be improved through new AUC based error measures, providing guidance to existing training methods to yield better AUC classifier performance.
>
> **Hypothesis 2**: Computing power harnessed by eInfrastructres enables systematic exploration of search spaces of machine learning classifiers configurations for given datasets, specifically biomedical datasets.
>
> **Hypothesis 3**: Computing power harnessed by eInfrastructures enables thorough validation of new classification methods.
>
> **Hypothesis 4**: Given the above three hypothesis, it is possible to develop more precise and robust breast cancer CADx methods.

1.4 Summary of contributions

The following theoretical and technological achievements were obtained as part of this thesis work.

1.4.1 Theoretical contributions

> **Contribution 1**: A new AUC based error definition (loss function) for machine learning algorithms.
>
> **Contribution 2**: An efficient error-bounded AUC approximation method with arbitrary precision
>
> **Contribution 3**: A methodology to integrate the AUC based error definition and the AUC approximation procedure into existing multilayer perceptrons, applicable to other ML methods.

1.4.2 Technological contributions

Contribution 4: A software framework for integrating third party machine learning classifiers, enabling the exploration of the search space formed by the possible parameters configurations of the model fitting processes implemented by the integrated classifiers.

Contribution 5: A software framework developed upon industry standards allowing (1) launching and maintaining colonies of Job Agents over heterogeneous computing resources and (2) submitting jobs to the Job Agents colonies through command line and API interfaces.

Contribution 6: An exploration methodology for the rational usage of the software frameworks (produced in contributions 4 and 5) over local and distributed computing resources.

Contribution 7: An application of the above contributions to search for well performing ML classifiers for breast cancer CADx based on medical data extracted from mammograms.

1.5 Thesis outline

This thesis is, therefore, structured as follows:

- **Chapter 1** is this introduction, which aimed at providing a general background for ROC analysis, machine learning, eInfrastructures and breast cancer CAD. It established the context within which this thesis was motivated, its objectives and summarized its contributions.
- **Chapter 2** describes the current state of the art of the three specific areas in whose intersection this thesis contributes: (1) the usage of ROC analysis in machine learning, (2) machine learning for breast cancer CAD and (3) the usage of eInfrastructures for scientific production.
- **Chapter 3** details the contributions of this thesis to enable AUC optimization in machine learning. In specific, it describes the algorithm developed to efficiently approximate the AUC within user defined error bounds and the method proposed to generalize AUC optimization in multilayer perceptrons.

- **Chapter 4** describes the two software frameworks constructed to enable an efficient usage of eInfrastructures for machine learning research and development. The Biomedical data analysis Toolkit (BiomedTK) allows researchers to manage large number of configurations of ML classifiers and train them with different ML algorithms integrating ROC analysis. The Cloud Computing Colonies framework (C3) deploys colonies of *Job Agents* into existing eInfrastructures enabling efficient usage of available computing resources to applications such as BiomedTK.
- **Chapter 5** shows a practical application of the contributions above in the field of breast cancer CAD, using AUC optimization in machine learning over eInfrastructures to find well performing ML classifiers to provide radiologists an automated second opinion for their diagnosis and patient management decisions.
- **Chapter 6** finally summarizes the conclusions of this thesis outlining future lines of work opened by its contributions.

Chapter 2

State of the Art

The major theoretical contributions of this thesis are in the area of ROC analysis for machine learning. From a technological perspective, the contributions of this thesis enable the effective utilization of eInfrastructures for massive exploration of machine learning methods and provide a practical application of these in breast cancer CADx. This section, therefore, outlines current developments in these three areas.

2.1 ROC analysis and machine learning

ROC analysis is used in machine learning since the works of (Spackman 1989) and (Bradley 1997). ROC curves are believed to be a more powerful tool than *accuracy* or other threshold measures to assess classifier performance mostly due to the fact that they constitute a richer object conveying more comprehensive information (Provost, Fawcett et al. 1998; Ling, Huang et al. 2003). In particular, *accuracy* tends to ignore skewed class distributions (more elements in one class than in another) and different misclassification costs, as usually arise in real work problems. Take for example a dataset with 1000 elements of which 900 are positive and 100 are negative. A naïve classifier labeling all elements as positive will achieve 90% accuracy and yet it is useless. In addition it seems that maximizing accuracy does not always result in AUC optimization. The work of (Cortes and Mohri 2004) formalized this intuition: minimizing the error rate ($= 1 - Accuracy$) as the vast majority of ML methods do, does not necessarily yield to AUC optimization. In addition, some results provide

evidence that AUC optimization might even yield to better accuracy (Ling, Huang et al. 2003). Although these results have still to be interpreted with caution when applied to specific application domains, they do indicate that a generalized usage of ROC analysis in ML might be beneficial. Since then, a few ML methods have been redesigned for AUC optimization, but that is a lengthy and costly task and the majority of techniques embedded within existing and newly developed ML methods aim at error rate minimization, while ROC analysis remains mostly limited to using the AUC for classifier comparison. Results of this thesis enable generalized AUC usage in the model fitting stage in machine learning.

The following subsections offer a perspective on how ROC analysis is used in different stages of machine learning, starting with a statistical view on ROC curves.

2.1.1 Obtaining ROC metrics

From a statistical perspective it is commonly assumed that the scores assigned by a classifier or a diagnostic test to elements of a dataset are instantiations of two random variables X^+ and X^- for positive and negative elements respectively. Functions f^+ and F^+ denote the PDF and CDF of the distribution of X^+, and analogously for X^-. The actual distributions f^+ and f^- are unknown and the researcher is faced with the problem to estimate the ROC curve and/or associated measures (such as AUC) based on the available observations (classifier output of a dataset). Regular statistical techniques are applied to obtain such estimates and literature is relatively extensive on this subject, since many results come from medical areas where ROC analysis is being used since earlier than in machine learning. Estimating ROC curves is also known in the literature as *curve fitting* (Metz 1978; Centor and Schwartz 1985; Metz 2008) and, as 2D objects, they are somewhat harder to estimate than scalar metrics like accuracy or even AUC itself. As usual, estimation techniques existing in literature in this sense can be classified into parametric, semi-parametric and non-parametric methods.

Non-parametric and semi-parametric methods

Non-parametric methods make no assumption about the underlying distributions of X^+ or X^-. Among them, empirical estimates are widely used for their simplicity and validity for many real world sized datasets. The method illustrated in figure 2 (Fawcett 2006) provides a straight forward empirical estimate for the ROC curve and

equation 1.1 constitutes a commonly used empirical estimate for the AUC. It is known to be an unbiased estimator for the AUC and is based on the fact that the AUC is equivalent to the probability of a randomly selected positive element to be ranked higher than a randomly selected negative element (Hanley and McNeil 1982) which also corresponds to the Mann-Whitney test (Mann and Whitney 1947) and the Wilcoxon statistic (Wilcoxon 1945). Observe that, for AUC purposes, in equation 1.1 (page 6) it is not the actual scores of each element what matters, but only their relative ranking.

Several authors (Zou, Hall et al. 1997; Lloyd 1998; Zou, Tempany et al. 1998; Lloyd and Yong 1999; Stine and Heyse 2001) discuss refining the non-parametric approach to provide a smoothed ROC curve using kernel based methods. In these approaches a function family (kernel) such as a Gaussian is chosen for $\hat{f}^+$ and $\hat{f}^-$ as estimators of the respective PDFs f^+ and f^- to derive an analytical expression for AUC and the ROC curve. Observe that in these cases, there is no assumption of the underlying distributions of positive and negative elements (f^+ and f^-) but there is a certain shape (kernel) imposed to their estimating functions $\hat{f}^+$ and $\hat{f}^-$. In certain conditions (Lloyd and Yong 1999), kernel based non parametric estimators have been found to be *better* than empirical ones in the sense of yielding asymptotically a lower mean squared error.

Parametric methods

A widely accepted parametric approach (Green and Swets 1966; Hanley 1996) is to assume that X^+ and X^- are independent normal variables $X^+ \sim N(\mu_+, \sigma_+^2)$ and $X^- \sim N(\mu_-, \sigma_-^2)$, referred to as *binormality*, and consequently the points of the ROC curve and the AUC are obtained by using the sample means and standard deviations of the positive and negative elements of the dataset in hand. This binormal model can be estimated by several methods (Hsieh and Turnbull 1996; Metz, Herman et al. 1998; Zou and Hall 2000) among others. In many cases, the normal assumption is untenable directly from the dataset data, although *ad hoc* transformations might make its application reasonable as suggested in (Goddard and Hinberg 1990; Reiser and Faraggi 1997) but those authors also provide examples where binormality fails.

Several authors proposed other parametric methods, assuming different distributions or procedures such as, among others, gamma distribution, logistic regression, etc. (Dorfman, Berbaum et al. 1997; Qin and Zhang 2003; Zou, Warfield et al. 2004)

In any case, parametric or non-parametric, efficient computation of ROC curves or AUC is essential if they are to be used in computing intensive contexts such as in machine learning. This issue is also addressed in this thesis as a necessary stage to use AUC in model fitting as mentioned previously.

2.1.2 Interpreting ROC curves

ROC curves provide a wealth of information about classifier performance and allow researchers to make decisions and compare classifiers under different scenarios (class skew, different misclassification costs, etc.), discriminating more conditions than regular threshold based metrics (such as *accuracy*). The following paragraphs provide current insight on the information that ROC plots can convey.

ROC space

The ROC space is defined in a two dimensional unit square with FPR (the false positive rate) and TPR (the true positive rate) as x and y axes respectively (Metz 1978; Fawcett 2006). A *calibrated classifier* is one that produces a rank on dataset elements where a specific threshold has been established to label them as positive or negative. This is represented as a point in the ROC space such as points A, B, C, C' and D in figure 10. Points along the horizontal diagonal represent classifiers performing equivalently to a random guess and we seek classifiers above the diagonal and approaching the $(1,0)$ point. By inverting the class label assigned by any classifier its position is mirrored around the random guess line (such as C and C').

An *uncalibrated classifier*, giving a rank of dataset elements produces a curve in the ROC space as explained in Section 1.1.1 containing all possible calibrated classifiers given that rank as the calibration threshold is moved from 0 to 1. The AUC is then taken as a single scalar to measure classifier performance without committing to a specific threshold value. However different classifiers might yield similar AUCs and one might favor one or other depending on specific problem conditions. As shown in figure 10, right, classifier containing point A seems to be a better choice on low FPRs than classifier containing point B.

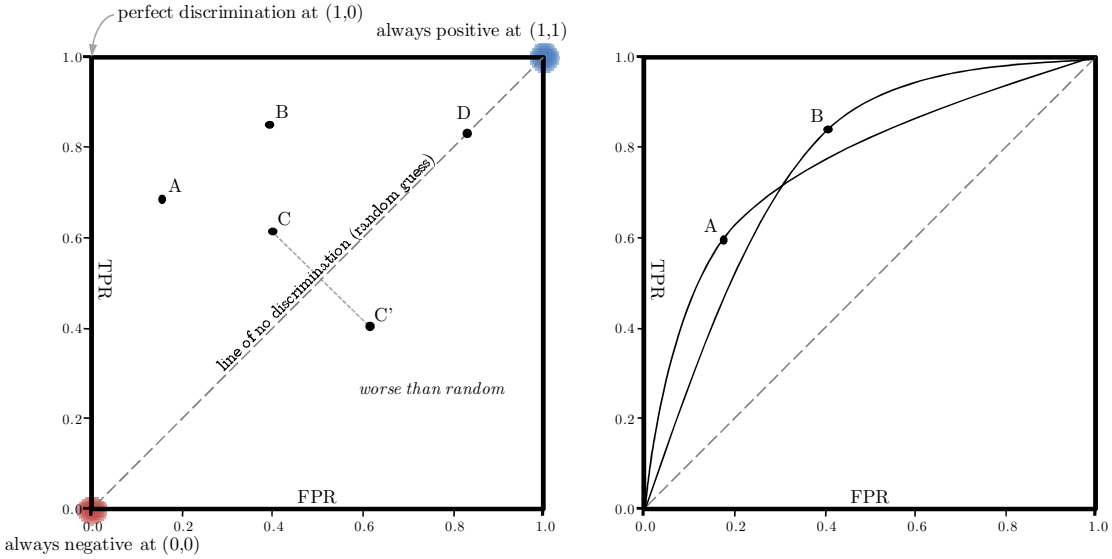


Figure 10: ROC space with example classifiers (left) and ROC curves (right)

Another view on ROC plots can be found on introductory ROC texts such as the ones mentioned above is the one depicted in figure 11, where points A and B represent different thresholds set on the distributions of positive and negative elements. The solid vertical line on the left represents point A and the figure shows how it partitions both distributions producing a specific point in the ROC space. As the threshold moves from A to B (shown as a vertical dashed line), the partition of the probability distributions changes and, therefore, the TPR/FPR point moves along the ROC curve.

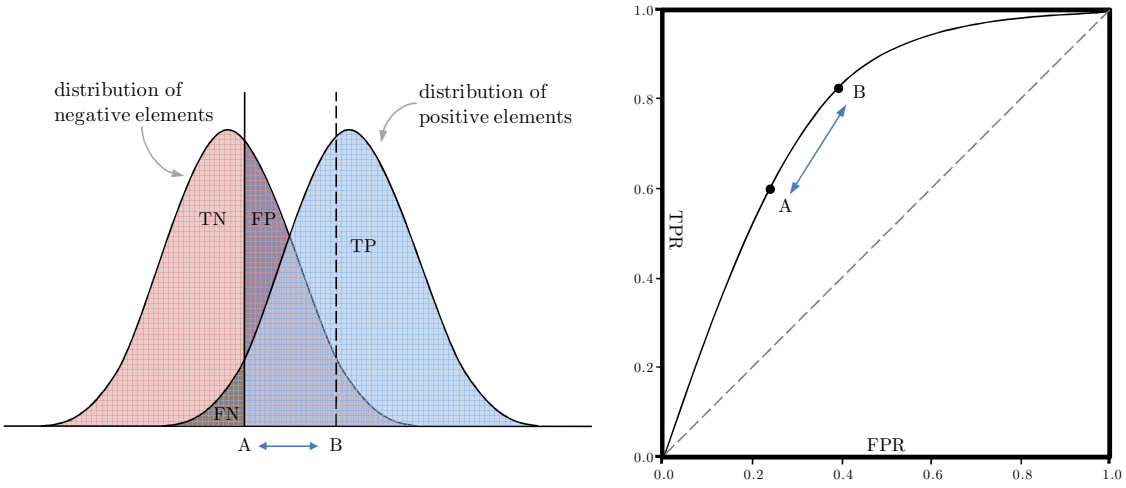


Figure 11: Example of elements distributions and classifiers in ROC space.

ROC isometrics

Provost et al. (Provost, Fawcett et al. 1998) showed that classifier comparison and selection based on accuracy has severe shortcomings with regard to class and error cost distributions. To overcome these problems (Flach 2003) and (Provost and Fawcett 2001) consider class and error cost distributions as a parameter to performance metrics. Evaluation with these metrics is named skew sensitive evaluation and a parameter *called skew* ratio expresses the relative importance of negative versus positive classes in terms of class and error cost distributions:

$$c = \frac{c(p,n)}{c(n,p)}\frac{P(n)}{P(p)}$$

where $c(p,n)$ and $c(n,p)$ denote the cost of false positive and false negative respectively, and $P(p)$ and $P(n)$ is the probabilities of a positive and negative instance. Empirically, they correspond to the proportion of positive and negative elements on the dataset. Given a certain (TPR, FPR) point on the ROC plot its expected accuracy cost is defined as: $P(p)\cdot(1-TPR)\cdot c(n,p) + P(n)\cdot FPR\cdot c(p,n)$. Then two points (TP_1, FP_1) and (TP_2, FP_2) have the same performance if their expected accuracy costs are the same, which results in the following condition

$$\frac{TP_2 - TP_1}{FP_2 - FP_1} = \frac{c(p,n)}{c(n,p)}\frac{P(n)}{P(p)} = c$$

so c is the slope of the line joining both points which is then defined as an *isometric accuracy* line. This is, all classifiers corresponding to points on the line have the same expected accuracy cost when applied to datasets characterized by c. Observe that a certain class distribution and misclassification cost correspond to a specific c value that characterizes them, so that each set of class distribution and misclassification costs defines a family of isometric accuracy lines. For instance, all datasets with the same number of positive and negative elements and same misclassification cost for both of them produce isometric accuracy lines with a slope of c=1. Lines closes to the top left corner on the ROC space correspond to classifiers with higher expected accuracy cost. Figure 12, left, shows isometric accuracy lines for c=1 and c=1/2 where it can be observed how, in the case of c=1 accuracy corresponds to the non-discrimination line (random guess).

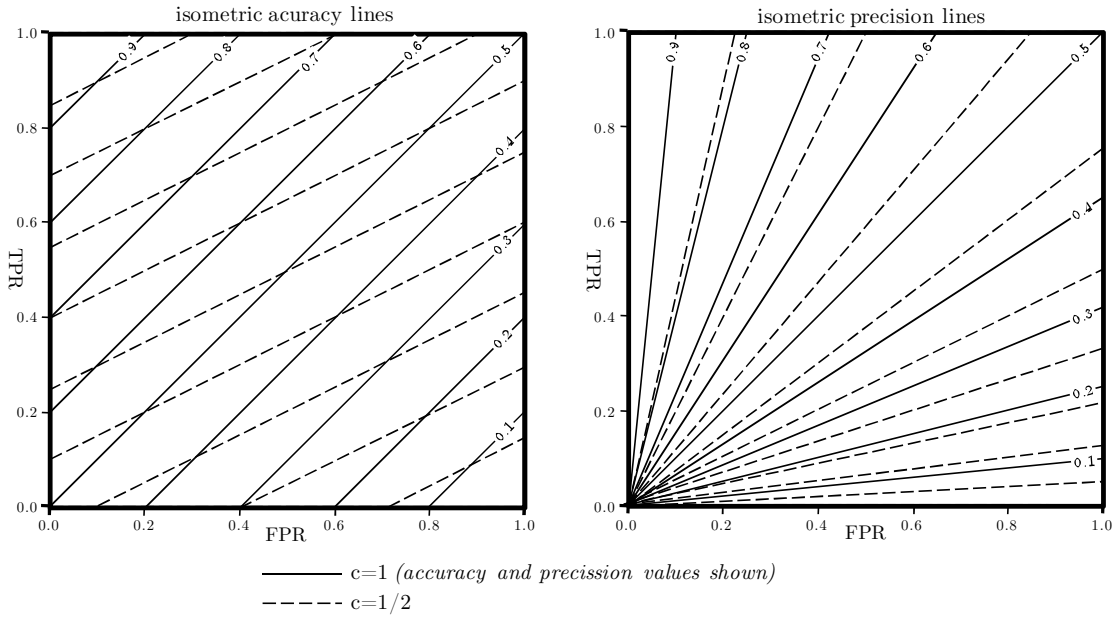


Figure 12: ROC isometrics for two class and cost distributions

Isometric accuracy is one of a family of ROC **isometrics**, where the ROC space can be navigated or partitioned by lines representing equivalent classifiers regarding a certain metric with respect to a certain class distribution and misclassification cost (c value). For illustration purposes figure 12, right, shows the *isometric precision* lines which produce a very different geometry of the ROC space. Observe how, for accuracy, we would seek calibrated classifiers or ROC curves approaching the top left corner, whereas for precision, we would be more interested on the left border of the ROC space. All these issues are further discussed in (Fawcett and Provost 1997; Flach 2003; Vanderlooy, Sprinkhuizen-Kuyper et al. 2006; Vanderlooy, Sprinkhuizen-Kuyper et al. 2009) among others.

2.1.3 ROC analysis for model evaluation

ROC analysis is primarily used in machine learning to evaluate and compare classifier performance through the AUC and few works in machine learning make a comprehensive use of ROC analysis in the sense described above. Apart from those mentioned in previous section, some machine learning tasks applied to biomedical problems use ROC analysis beyond AUC evaluation specially in medical imaging (Park, Goo et al. 2004; Metz 2008; Park, Pu et al. 2009), which is one of the motivations behind the work in this thesis.

Results in (Provost, Fawcett et al. 1998; Lachiche and Flach 2003; Ling, Huang et al. 2003; Cortes and Mohri 2004; Rosset 2004; Vanderlooy and Hullermeier 2008) suggest the usage of AUC over accuracy for measuring classifier performance mostly due to the difficulties for accuracy to deal with class skews and misclassification costs. In later years a few authors questioned the usage of AUC claiming that, in certain cases, AUC performance might be misleading since it may introduce excessive noise or greater variance in its results with respect to accuracy (Lobo, Jiménez-Valverde et al. 2008; Hanczar, Hua et al. 2010).

Application of commonly used evaluation techniques such as cross-validation and bootstrap (as mentioned in Section 1.1.1.2) is a key issue to robustly use any performance metric. Researchers have paid attention to the generation of confidence intervals or bands for ROC curves so that it could be expected that, assuming test examples are all drawn from the same fixed distribution, the model's ROC curves would fall within certain bands with probability $1-\delta$. These bands could be obtained through different techniques. *Vertical averaging* (Provost, Fawcett et al. 1998) scans successive FPR values and averages the TPR values of multiple ROC curves at that FPR value (figure 13 left). *Threshold averaging* (Fawcett 2003), on the contrary, freezes successive threshold levels and averages FPR and TPR values (figure 13 right). Other approaches to find confidence bands for ROC curves can be found at (Reiser and Faraggi 1997; Obuchowski 1998; Shapiro 1999; Jensen, Müller et al. 2000; Sorribas, March et al. 2002; Macskassy, Provost et al. 2005; Vergara, Norambuena et al. 2008).

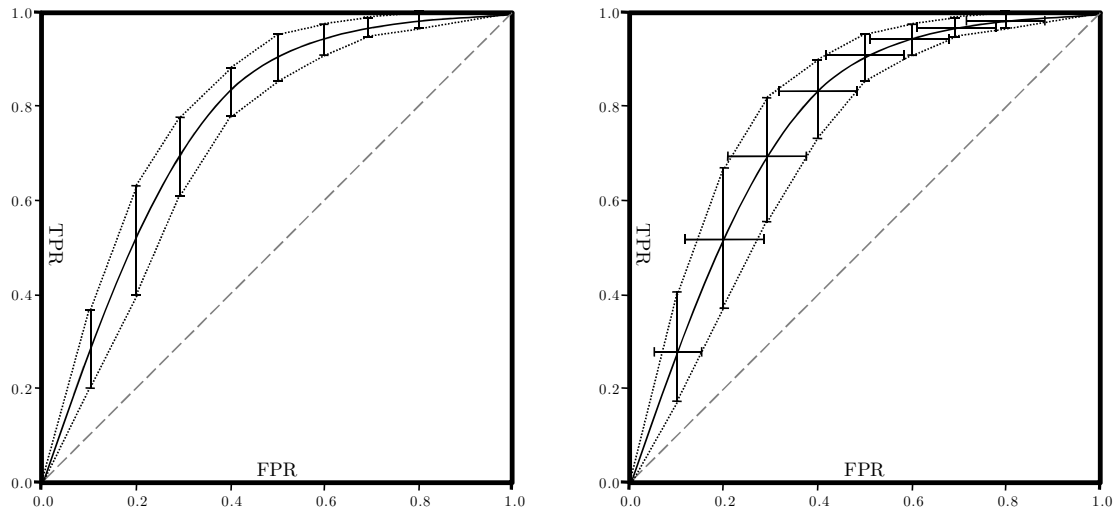


Figure 13: Confidence bands for ROC curves

These averaging techniques are then also used when applying cross-validation, bootstrapping or other resampling techniques. In another technique, *pooling,*

predictions made in each cross-validation round are pooled into one set and one common AUC or ROC curve is calculated from it. For LOOCV (leave one out cross validation) this is the only way to obtain them, although (Parker, Gunter et al. 2007) showed that when considering AUC with small datasets many commonly used cross-validation schemes suffered from substantial negative bias. To overcome this (Airola, Pahikkala et al. 2009) and (Cortes, Mohri et al. 2007) proposed LPOCV (leave pair out cross validation) discarding, at each cross-validation round a different positive-negative element pair. However, LPOCV is only computationally feasible for small datasets since requires one model fitting cycle per possible positive-negative pair. For a dataset with a $n_p + n_n$ elements (positive and negative) LOOCV requires $n_p + n_n$ cycles and LPOCV requires $n_p \cdot n_n$ cycles. Obtaining AUC estimates based on these principles can be reviewed at (Bradley 1997; Airola, Pahikkala et al. 2009; Airola, Pahikkala et al. 2011).

2.1.4 ROC analysis for model construction

Mainstream classifiers are usually designed to minimize the classification error and may not necessarily perform optimally when applied to ranking problems as shown by (Ling, Huang et al. 2003; Cortes and Mohri 2004). Furthermore, (Ling, Huang et al. 2003) suggests that AUC optimization may also result in better accuracy. This has led to a certain set of efforts to redesign existing algorithms and develop new ones aimed at AUC optimization instead of error rate minimization.

The empirical estimate of the AUC, or the Wilcoxon statistic, as defined in equation 1.1 for dataset $S = S_P \cup S_N$ with respect to the ranking produced by classifier h is recalled here:

$$AUC(S,h) = \frac{\sum_{p \in S_P} \sum_{n \in S_N} \mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P| \cdot |S_N|}$$

Several approaches for AUC optimization apply some gradient descent method. One of the major problems is that the indicator function $\mathbf{1}[\,]$ (usually modeled as the Heavyside step function) is not differentiable and a continuous approximation is often adopted such as the sigmoid function or a Chebyshev approximating polynomial. However there are two problems associated with this approach. On one side, this formulation is non linear with respect to the learning parameters and this requires an iterative search to locate the solutions. Consequently, different initializations (such as the random weights initially assigned to any ANN) may end up in different local solutions, hence incurring laborious trial and error efforts to select an appropriate

setting. The second problem is that the objective function could be ill-conditioned to the numerous local plateaus resulting from summing the flat regions of the sigmoid and step-like approximation and, therefore, a lot of search iterations could be spent in making little progress at locally flat regions (Yan, Dodier et al. 2003; Herschtal and Raskutti 2004; Rakotomamonjy 2004; Calders and Jaroszewicz 2007; Castro and Braga 2008; Toh, Kim et al. 2008). Different approaches have been proposed to circumvent these problems, mostly by using a different approximation technique such as through a quadratic approximation (Toh, Kim et al. 2008) or kernel based methods (Faraggi and Reiser 2002; Marzban 2004) which require either the assumption of an underlying distribution for the data or additional smoothing functions.

Complementing gradient descent methods, other learning techniques have been redesigned or developed to allow AUC optimization. Among them, SVMs (Rakotomamonjy 2004; Brefeld and Scheffer 2005; Joachims 2005; Steck 2007; Pahikkala, Airola et al. 2008), heuristic search and decision trees (Mozer, Dodier et al. 2001; Ferri, Flach et al. 2002; Yan, Dodier et al. 2003; Herschtal and Raskutti 2004), linear programming (Ataman, Street et al. 2006), regression learning (Waegeman, De Baets et al. 2008) and others approaches based on feature selection (Wang and Tang 2009), pair wise comparison (Marrocco, Duin et al. 2008) or by similarity with other problems (Clémençon, Vayatis et al. 2009) with relative success. Finally, in a similar aim to accuracy boosting algorithms such as AdaBoost (Freund and Schapire 1995), boosting for AUC has received particular attention recently as a method to combine underperforming classifiers (Long and Servedio 2007; Robilliard, Marion-Poty et al. 2007; Moin 2009; Komori and Eguchi 2010).

However, the mainstream techniques and accumulated methods for machine learning have been or are being designed for error rate minimization. This might be partly due to the fact that AUC (and ROC analysis in general) has not yet completely settled within the ML community, but also to the fact that redesigning the vast amount of existing ML methods is an endeavor requiring considerable effort, as pointed out by (Ling, Huang et al. 2003). It is in this context that the theoretical contributions of this thesis can be valued, providing a method to integrate AUC optimization into existing algorithms with reasonable effort.

2.1.5 ROC analysis for model selection

When ROC analysis is used in machine learning, the empirical AUC (see page 26) is the metric largely used for model evaluation and selection. Some variations over the AUC have been proposed (Wu and Flach 2005; Wu, Flach et al. 2007; Chatelain, Adam et al. 2010) exploiting the richness of ROC space as shown in Section 2.1.2. An increasingly referenced method is the ROC convex hull (ROCCH) (Provost and Fawcett 2001) based on accuracy isometrics. ROCCH builds on the observation that two different calibrated classifiers (points in the ROC space) may perform differently for different class distributions and misclassification costs. Figure 14 left illustrates this. Classifier A outperforms (better accuracy) classifier B for a dataset with, for instance, the same number of positive and negative elements and the same misclassification costs for each ($c = 1$) as shown by the isometric accuracy line passing by classifier A, but leaving B below it. However, for datasets characterized by $c = 1/2$ (such as, for instance, the ones having twice as many positive elements as negative ones at the same misclassification cost), classifier B outperforms A, as shown by the isometric accuracy line passing by B.

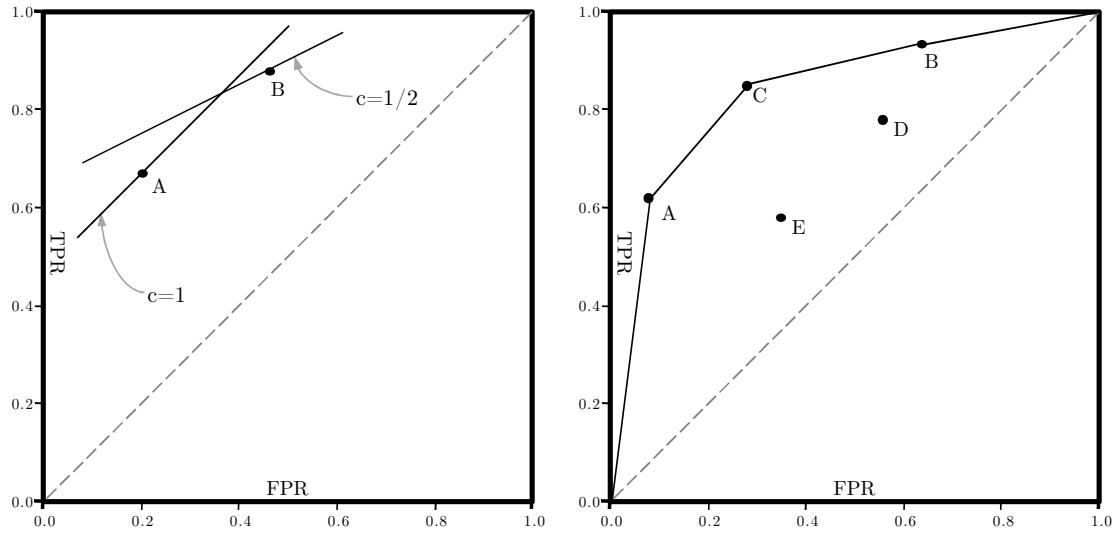


Figure 14: Isometric accuracy classifier comparison (left) and convex hull (right)

Given a set of classifiers (such as in figure 14 right) the slope of the isometric accuracy line joining them defines what kind of datasets that for which them both perform equal. Classifiers lying south east of that line perform worse. The ROCCH is then defined as the minimum set of segments of isometric accuracy lines joining classifiers so that all classifiers either lie on the convex hull or below it, as shown in

figure 14 right. As proven in (Provost and Fawcett 2001) any classifier below the convex hull is suboptimal since for any class distribution and misclassification cost (this is, for any c) there is always another classifier performing better, such as in the case of classifiers D and E in figure 14. Classifiers on the ROCCH are optimal in the sense that they are the ones performing the best for certain class distributions. The ROCCH can also be useful for sensitivity analysis. Imprecise information about class distribution or misclassification cost results in a range of slopes of isometric accuracy line passing by each optimal classifier, some of those might change its optimality condition.

2.1.6 Summary of references

Table 4: Summary of references for ROC analysis in machine learning

Introductory	(Spackman 1989) (Bradley 1997) (Metz 1978) (Fawcett 2006) (Fawcett 2003)
Non & Semi parametric estimation	(Mann and Whitney 1947) (Hajian-Tilaki and Hanley 2002) (Hanley and McNeil 1982) (Zou, Hall et al. 1997) (Lloyd 1998) (Zou, Tempany et al. 1998) (Lloyd and Yong 1999) (Stine and Heyse 2001) (Metz 2008) (Fawcett 2006)
Parametric estimation	(Green and Swets 1966) (Hanley 1996) (Hsieh and Turnbull 1996) (Metz, Herman et al. 1998) (Zou and Hall 2000) (Goddard and Hinberg 1990) (Reiser and Faraggi 1997) (Faraggi and Reiser 2002) (Dorfman, Berbaum et al. 1997) (Qin and Zhang 2003) (Zou, Warfield et al. 2004)
ROC space and isometrics	(Metz 1978) (Fawcett and Provost 1997) (Provost, Fawcett et al. 1998) (Provost and Fawcett 2001) (Flach 2003) (Fawcett 2006) (Vanderlooy, Sprinkhuizen-Kuyper et al. 2006) (Vanderlooy, Sprinkhuizen-Kuyper et al. 2009)
Model evaluation and confidence bands	(Fawcett 2003) (Reiser and Faraggi 1997) (Obuchowski 1998) (Shapiro 1999) (Jensen, Müller et al. 2000) (Sorribas, March et al. 2002) (Macskassy, Provost et al. 2005) (Parker, Gunter et al. 2007) (Airola, Pahikkala et al. 2009) (Cortes, Mohri et al. 2007) (Bradley 1997) (Airola, Pahikkala et al. 2011)
AUC vs. accuracy	(Provost, Fawcett et al. 1998) (Lachiche and Flach 2003) (Huang and Ling 2005) (Ling, Huang et al. 2003) (Ling, Huang et al. 2003) (Cortes and Mohri 2004) (Rosset 2004) (Vanderlooy and Hullermeier 2008) (Lobo, Jiménez-Valverde et al. 2008) (Hanczar, Hua et al. 2010)
AUC optimization methods	(Yan, Dodier et al. 2003) (Herschtal and Raskutti 2004) (Rakotomamonjy 2004) (Calders and Jaroszewicz 2007) (Toh, Kim et al. 2008) (Faraggi and Reiser 2002) (Marzban 2004) (Brefeld and Scheffer 2005) (Joachims 2005) (Steck 2007) (Pahikkala, Airola et al. 2008) (Mozer, Dodier et al. 2001) (Ataman, Street et al. 2006) (Ferri, Flach et al. 2002) (Waegeman, De Baets et al. 2008) (Castro and Braga 2008) (Wang and Tang 2009) (Marrocco, Duin et al. 2008) (Clémençon, Vayatis et al. 2009) (Long and Servedio 2007) (Robilliard, Marion-Poty et al. 2007) (Moin 2009) (Komori and Eguchi 2010)
Model selection and convex hull	(Provost and Fawcett 2001) (Wu and Flach 2005) (Wu, Flach et al. 2007) (Chatelain, Adam et al. 2010)

2.2 eInfrastructures

According to (FECYT 2004) eScience is understood as the set of scientific activities developed by using distributed resources (mainly computing power and storage capacity) accessible through high speed communications networks. Although supercomputers have been around already for a while, it is not until the end of the 1990s than a growing notion of a unified distributed utility computing infrastructure started to settle, encouraged by the spreading of increasingly faster communications networks and finer grained computer simulated research that was attained as computing resources became more affordable and more powerful. This notion of *eInfrastructure*, or *Cyberinfrastructure* (Atkins, Droegemeier et al. 2003; Newman, Ellisman et al. 2003; Hey and Fox 2005), was initially embraced by governments and scientific institutions who fostered their birth and funded their development. Today, as new technological developments occurred during the last decade, different kinds of eInfrastructures populate the world, in terms of their technological foundation, organizational reach, geographical distribution, targeted users, etc.

eInfrastructures can arguably be categorized into three kinds. **Supercomputers** evolved from specially designed state-of-the-art mainframe machines from decades ago, to large computer clusters of highly interconnected homogeneous CPUs. Supercomputers are mostly suited for HPC (High Performance Computing) applications, exploiting parallelism inherent within algorithms' implementations by using high speed inter CPU communication, such as through MPI (MPI-Forum 2009) or shared memory. **Grids** are federations of computer clusters, aggregating large sets of heterogeneous computing resources and providing appropriate mechanisms to ensure security and federation. Grids are suited for HTC (High Throughput Computing) applications, based on sequential algorithms parallelizable by parameter sweeps or data partitioning. Due to their academic and historic origins, both supercomputers and Grids are rooted in batch queued execution systems to regulate access to CPU cycles. Finally, **Clouds** exploit recent advancements in virtualization technology to decouple service provisioning from the physical resources. Services or application are encapsulated within virtual machines which can then be deployed on whatever physical machines are available. Clouds are not *a priori* tied to any particular kind of application or computing model, and it is the actual functionality encapsulated within a certain population of deployed virtual machines and the specifics of the infrastructure

provider what determines the final capabilities offered to the user. Virtual machines may contain packaged applications, databases, services or even Grid components.

2.2.1 Grids

The term "Grid" was coined in the mid-1990s to refer to a distributed computing infrastructure for advanced science and engineering. In (DeRoure, Baker et al. 2003) and (Foster and Kesselman 2004) three different stages in the evolution of Grid technologies are sketched. First generation Grids deploy proprietary systems in order to connect high performance supercomputers between scientific institutions. Second generation Grids (Foster 2001) focus on middleware technology to overcome the heterogeneity and scalability challenges of distributed systems, mostly built over Internet protocols. Third generation Grids (Foster 2005; Hey and Fox 2005) enlarge the standardization process of Grid technologies and their application domains. They are designed according to the principles of service oriented architectures (SOA) and make use of the Web Services technology stack.

Architecture of Grids

Second generation Grids' resources architecture is organized in the following layers (Foster 2001). The **Fabric Layer** includes protocols and interfaces that provide sharing facilities of logical resources such as CPU cycles, storage capacity, networks and sensors. The **Connectivity Layer** defines basic Grid specific network protocols, including Internet-based communications protocols to exchange messages with resources provided by the fabric layer. Furthermore, authentication protocols ensure controlled resource sharing. The **Resource Layer** consists of protocols for a secure negotiation, sharing, initiation, monitoring, control, accounting and payment of resources. Protocols of this layer are only responsible for local resources. The **Collective Layer** defines protocols and services for global resource management. It provides functions and interaction protocols required for collections of distributed resources. The **Application Layer** comprises a variety of Grid enabled user applications that benefit from the resources of the underlying Grid infrastructure.

At its core, a Grid is made of a federation of many Resource Centers (**RC**) and a set of central coordinating services hosted by one or more Resource Operation Centers (known as ROCs in middleware terminology, but denoted as **RO Center**, in this thesis

to avoid confusion with the acronym for *Receiver Operating Characteristic*). RO Centers take care of, for instance, maintaining a central catalogue of files stores throughout the federation, or to determine which RC is the most appropriate one to run a computing job required by a user (according to its hardware, free resources, applications available, etc.) A Grid federation is managed through middleware, a set of software components through which a federation offers its services. For instance, a RO Center hosts the middleware services allowing forwarding jobs to the most appropriate RC, gathering its results, storing them and making them available to users, etc. It also may guarantee file replicas across different RCs so that they are physically close to job execution, authenticate users through digital X.509 certificates, coordinate the execution of jobs requiring resources from several RCs, etc. Figure 15 shows a sample Grid federation with typical middleware components such as the Computing Element (CE, regulating Grid access to local computing power), the Storage Element (SE, regulating access to Grid local storage resources), the User Interface (UI, through which user interact with all Grid resources, local or remote), the Virtual Organization Membership Service (VOMS, determining users' membership to different VOs), the Workload Management System (WMS, distributing jobs to appropriate CEs), the File Catalog (LFC, accounting for the SEs where files of the federation are stored and replicated), etc.

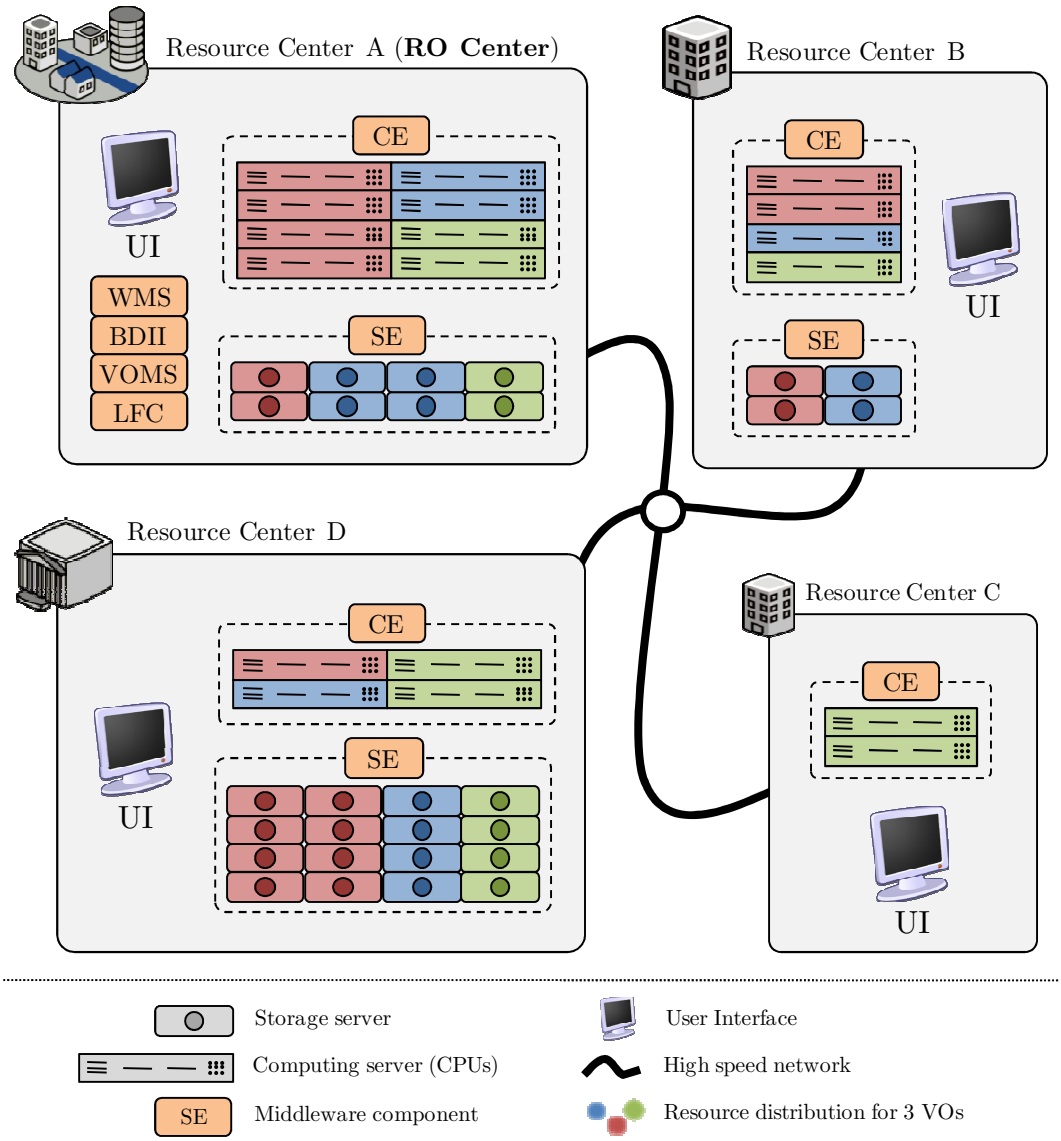


Figure 15: Example Grid federation

Grid resources are made available to users and applications through the notion of Virtual Organizations (**VO**) (Foster 2001). A VO represents users of a certain community, scientists participating in the same experiment, project or disciple. Within a Grid federation, a certain VO negotiates with RCs the resources the community behind needs to satisfy its computing and storage requirements, aggregating resources offered by each RC. Each VO determines and manages, following their own criteria, which users belong to the VO, and may negotiate resources across many RCs and Grid federations. This is, each VO has its own organization and rules according to the culture, procedures, tradition, etc. of the user community it represents. In general, a user is granted access to Grid resources of a federation according to the VO he belongs to.

Beyond resources, architecture for third generation Grids is standardized mostly by OGSA, the Open Grid Services Architecture (Global-Grid-Forum 2005) that provides a framework to expose Grids' resources through Web services. For instance, it includes execution management services, monitoring and discovering services, data transport, replication services, etc. By offering a Web services exposure (beyond artisanal command line utilities), OGSA provides the means to build client applications that consume Grid provided resources.

World federations, standardization and middleware

Today, different transnational Grid federations are being built across the globe, supported by middleware stacks increasingly integrated and standardized that enable the architectures just described. In Europe, the **European Grid Infrastructure** (EGI 2011) integrates the National Grid Infrastructures (NGIs 2011) across the continent to create a federated pan-European Grid resource, gathering today more than 300 resource centers and 70'000 CPU cores. Created in February 2010, it culminates the series of EU funded projects (DataGrid, EGEE I, II and III) starting in 2001 to guarantee the long-term availability of a generic eInfrastructure for all European research communities and their international collaborators. In the US, the **TeraGrid** integrates high-performance computers, data resources and tools, and high-end experimental facilities around the country. Currently, TeraGrid resources include more than 2.5 petaflops of computing capability and more than 50 petabytes of online and archival data storage, with rapid access and retrieval over high-performance networks. Other Grid federations can be found in Latin America (IGALC 2011), India (EUIndiaGrid 2011), Japan (NAREGI 2011), etc.

GT4, the Globus Toolkit 4 (Foster 2005), is the reference middleware implementation of the OGSA model and constitutes the foundation of many middleware distributions. In Europe, the gLite middleware (gLite 2011), based on GT4, was produced by the EGEE projects and is deployed on most of EGI sites. Other middleware distributions were also produced through publicly funded projects such as UNICORE (UNICORE 2011) or ARCO (NordUGrid 2011) among others, and a comprehensive effort to produce a unified middleware distribution is being done in the context of the European Middleware Initiative project (EMI 2011)

However, in spite of having been around for the last 10 years, Grids are still far from achieving the initial expectations of penetration and dissemination throughout scientific

disciplines and business domains, remaining mostly within the academic area. Among others, one of the reasons behind this is the difficulty of usage of the middleware, constituting a steep learning curve and cost barrier for new communities not having the tradition nor the resources to work with Grids (Kacsuk 2007; Schwiegelshohn, Badia et al. 2009; InsightCorp 2011). In fact, the lifecycle through which new applications are adapted to use existing middleware is long and slow and this has caused even the more privileged scientific communities (such as High Energy Physics) to develop their own particular tools and methods to reduce usage costs and ease up their users' lives, such as, among others, DIRAC (DIRAC 2010) and AliEn (Bagnasco and et al. 2008) used respectively by the CERN LHCb and ALICE experiments.

2.2.2 Clouds

Since the mid 2000's, with the increasing maturity of virtualization technology, the possibility to decouple the physical infrastructure from the computing service provided became a feasible reality. Services, applications, databases, etc. are then encapsulated within virtual machines which could now be deployed on a "cloud" of physical resources (figure 16). When physical machines are added or removed from the cloud, or even when they fail, cloud middleware relocates the virtual machines transparently and automatically without disturbing the service. Although many definitions have been proposed for the notion of a *Cloud* both in academia and industry, the one provided by (Mell and Grance 2009) seems to include the key elements commonly used:

> *"Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*

This definition then refers to a set of characteristics, service and deployment modes for Clouds. The characteristics are the following: *on-demand self-service* (computing resources are provisioned automatically according to user demand), *broad network access* (including usage by different client hardware platforms such as servers, laptops, PDAs, etc.), *resource pooling* (resources from providers are pooled to/from different physical and virtual locations in a transparent manner to end user), *rapid elasticity* (capabilities are quickly provisioned, scaled out and scaled in, appearing to be unlimited to the end user) and *measured service* (providers optimize their resources monitoring and leverage them according to demand).

Resources capabilities can then be serviced through different models according to the end object consumed by the user: *Software as a Service* (SaaS, the object provisioned are software applications or components), *Platform as a Service* (PaaS, provisioning specific platforms, such as for development, etc.) and *Infrastructure as a Service* (IaaS, provisioning specific computing resources, such as virtual machines, storage capability or network bandwidth). Applications, platforms or, sometimes operating systems, encapsulated within a virtual machine are often denoted as *virtual appliances*, emphasizing the fact that the can be used and disposed off the shelve.

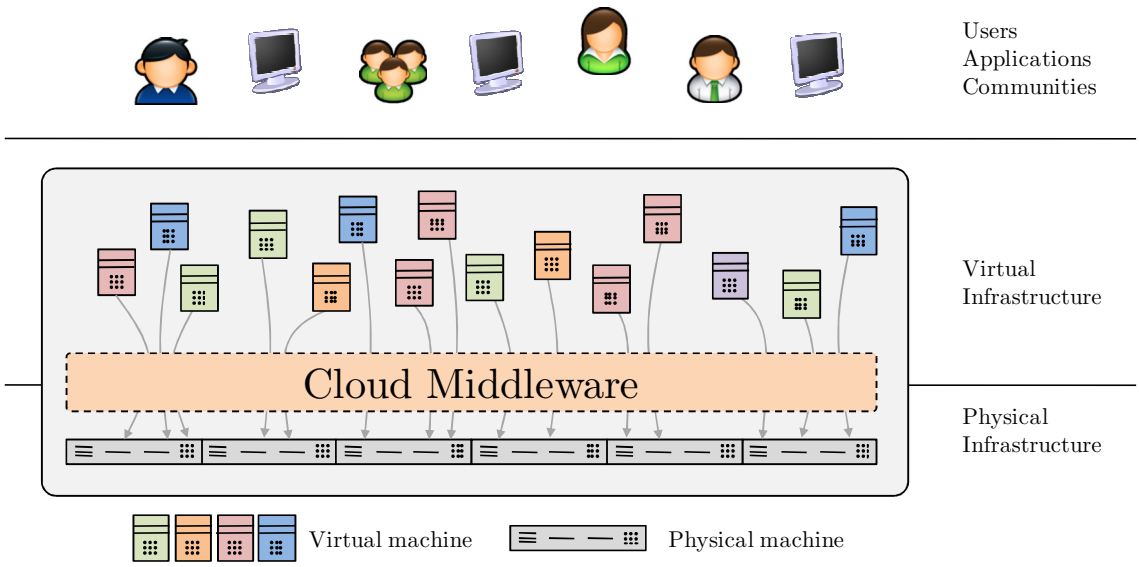


Figure 16: Cloud resource provision through virtualization

Different Cloud platforms are currently available being Amazon EC2 (Amazon 2011) arguably the pioneer, but also Google App Engine (Google 2011), Microsoft Azure (Microsoft 2011), IBMs Blue Cloud (IBM 2011), Nimbus (NIMBUS 2011), Open Nebula (OpenNebula 2011), etc. See (Rimal, Eunmi et al. 2009) for a comprehensive review. Across the different cloud providers, virtualization is the key technology being used, decoupling the final service provisioning (whether SaaS, PaaS or IaaS) from the physical infrastructure and enables achieving many of the previously mensioned characteristics by managing the load of virtual machines over the physical infrastructure. This way, the physical resources do not need to *be prepared* in advance to host a user demand as long as they can host the virtual machines servicing it.

2.2.3 Using eInfrastructures

Grids have mostly remained in the academic area, whereas Clouds seem to gain terrain in industry. However a few issues still remain unsolved in both worlds. One such issue is interoperability, which refers to the capability of different eInfrastructures to seamlessly interchange resource provisions to comply with a user demand (such as user jobs being transferred between different Grid federations or virtual machines between different Clouds) and remains still unsolved both in Grid and Cloud systems (Parameswaran and Chaddha 2009). This is mostly an organizational issue since, technically, interoperability is a standardization issue. Several architectures, standardization bodies and working task forces have been established to seeking solutions to this problem such as (CCIF 2011; Cordys 2011; UnifiedCloud 2011) just to mention a few. However, their success depends mostly on the degree on consensus achieved between the different stakeholders in the public and private sectors (resource providers, users, service providers, developers, etc.)

Another issue is cost of usage, in terms of effort required to efficiently exploit eInfrastructures for a problem in hand. In Grids, this refers to the percentage of jobs failing due to middleware and non-application specific causes (job reliability), and the overhead in job execution due to middleware handling of jobs (job latency). Depending on middleware and data center installations Grids typically show job latencies in the order of several minutes, see (Tristan, Diane et al. 2007; Glatard, Zhou et al. 2009). It is also known that job failure rates are significantly high ranging between 5% of 25% of submitted job failures due to middleware reasons (Hui, Groep et al. 2006), although this is improving in latest releases. All this requires users and administrators to develop their own management tools (for resubmitting failed jobs, data integrity checking, job checkpointing, etc.) to ensure their scientific production gets correctly executed and the infrastructure can be managed (Ramos-Pollan, Callejon et al. 2007), raising considerably the cost (effort) of using and maintaining Grids and therefore slowing its adoption in a wide sense.

2.3 Machine learning classifiers for breast cancer CAD

There are two types of examinations performed using mammography: screening mammography and diagnostic mammography. Screening allows detecting breast cancer in an asymptomatic population and diagnostic aims to examine a patient who has

already demonstrated abnormal clinical findings (Ng KH 2003). Double reading of mammograms (two radiologists read the same mammograms) (Brown, Bryan et al. 1996) has been advocated to reduce the proportion of missed cancers, but the workload and cost associated are high. With the support of computer-aided detection and/or diagnosis (CAD) systems only one radiologist is needed to read each mammogram rather than two.

2.3.1 Mammography classification standards

Mammography is a radiological diagnostic method which relies on an X-ray examination of breasts and is a process involving the use of low-dose amplitude-X-rays (usually around 0.7 mSv). The aim of mammography is to detect very small abnormalities in the breast tissue before they develop into breast cancer, typically through detection of characteristic masses and/or microcalcifications. Mammography is a very sensitive diagnostic method that requires very precise equipment and qualified medical personnel to perform the examination (Kowalik and Konstanty 2010)

The BIRADS Atlas (D'Orsi, Bassett et al. 2003), for Breast Imaging-Reporting and Data System, developed by the American College of Radiology, is a quality assurance tool originally designed for use with mammography widely used in the community. In day-to-day usage, the term BIRADS refers to standardized mammography assessment categories and lesion descriptions. BIRADS defines nine categories typically assigned by a radiologist when interpreting a mammogram. These categories are listed in table 5.

Table 5: BIRADS categories for standardized diagnostic assessment

BIRADS-0	Incomplete, additional evaluation required
BIRADS-1	No findings. 0% possibility of cancer
BIRADS-2	Benign findings. 0% possibility of cancer
BIRADS-3	Probably benign findings, further control after months needed. 0% to 3% chance of cancer
BIRADS-4	Doubtfully malign finding
BIRADS-4.a	Low suspicious of malignancy. 3 to 49% chance of cancer
BIRADS-4.b	Medium suspicious of malignancy. 50% to 89% chance of cancer
BIRADS-4.c	Medium-high suspicious of malignancy. 90% to 94% hance of cancer
BIRADS-5	High suspicious of malignancy. >95% chance of cancer
BIRADS-6	Malignancy confirmed

Transversally to this, mostly four kinds of lesions are of interest in the breast cancer CAD field as shown in examples in figure 17.

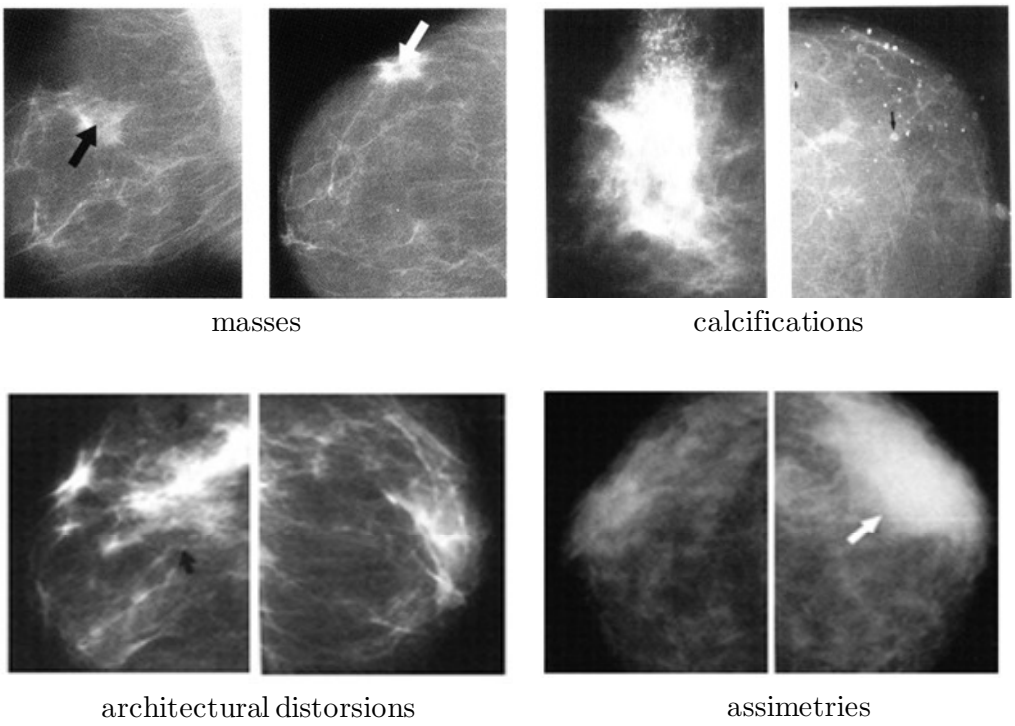


Figure 17: Common kinds of mammography lesions

These lesions are defined by the BIRADS Atlas as follows:

Masses: A Mass is a space occupying lesion seen in two different projections. If a potential mass is seen only in one projection it should be called an "Asymmetry" until it is three-dimensionally confirmed.

Calcifications: Calcifications that can be identified as benign on mammography are typically larger coarser and more easily seen than malignant calcifications. Calcifications associated with malignancy (and many benign ones as well) are usually very small and require often the use of a magnifying class to see them.

Architectural Distortion: The normal architecture is distorted with no definite mass visible. This includes thin lines or spiculations radiating from a point and focal retraction or distortion of the edge of the parenchyma. Architectural distortions can also be associated with a mass, asymmetry or calcifications. In the absence of appropriate history of trauma surgery,

architecture distortion is suspicious for malignancy or radial scar and biopsy is appropriate.

Asymmetries: Asymmetric breast tissue is judged relative to the corresponding area in the contralateral breast and represents a greater volume of breast tissue over a significant portion of the breast. There is no mass, distorted architecture or associated suspicious calcifications. Global asymmetric breast tissue usually represents a normal variation, but may be significant when it corresponds to a palpable abnormality.

BIRADs further characterizes these lesions by providing more specific criteria (shape, texture, margins, etc.) and standardizes notation for location, description, etc. In fact, usually calcifications are easier to detect (either manually or automatically) for their small determined size and high contrast.

2.3.2 Computer Aided Detection/Diagnosis

Developing CAD schemes has been attracting rapidly growing interest in biomedical imaging research field in the past two decades, aiming to assist clinicians (radiologists, pathologists, etc.) more accurately, consistently and efficiently to read and understand biomedical images. With some ambiguity, the literature uses the CAD term to refer both to "Computer-Aided Detection" (CADe) and the "Computer Aided Diagnosis" (CADx). While CADe is concerned with locating suspicious regions within a certain medical image (such a mammogram), CADx is concerned with offering a diagnosis to a previously located (identified/segmented) region. The applied work presented in this thesis is focused on the CADx area.

Breast cancer CAD methods/systems have been applied to detect suspicious lesions depicted on mammography images. After identifying initial candidates for the targeted suspicious lesions, most schemes use a pre-trained multi-image-feature based machine learning classifier to classify these candidates into two groups of positive (malignant) and negative (benign) tissue detections. In these systems a radiologist uses the output from computerized analysis of medical images as a "second opinion" in detecting and classifying lesions with subsequent diagnostic and patient management decisions (Cheng, Cai et al. 2003; Cheng, Shi et al. 2006; Yoon 2007; Dehghan and Abrishami-Moghaddam 2008; Giger and Suzuki 2008; López, Novoa et al. 2008; Paquerault, Hardy et al. 2010).

At present, developed CAD systems have been mainly focused onto the detection of lesions associated to both microcalcifications and masses, but methods with better performance are related to microcalcification detection. Clusters of microcalcifications are an important indicator of breast cancer appearing in 30%–50% of diagnosed cases (Kopans 2006). Microcalcification detection methods could be divided into the following categories: image enhancement, stochastic modeling, multiscale decomposition and machine learning (ML). The basic idea behind *image enhancement* is to improve the contrast of microcalcifications, and then apply a threshold to separate them from their surroundings. Successful approaches include filtering (Nishikawa, Giger et al. 1995), noise and histogram equalization (McLoughlin, Bones et al. 2004; López, Novoa et al. 2008), and region grouping (Wei, Fei et al. 2002). Its main advantages are its simplicity, ease implementation and efficiency. *Stochastic modeling* exploits statistical differences (skewness, kurtosis, etc) between microcalcifications and their surroundings (Gurcan, Yardimci et al. 1997). Stochastic modeling based on Markov random fields (Casaseca-de-la-Higuera, Arribas et al. 2005) demonstrated advantages to characterize the spatial intensity distribution of an image, but estimating a proper prior distribution remains a challenging task. *Multiscale decomposition* makes use of differences in frequency content among microcalcification spots and their surrounding background and it is generally used as feature extraction technique. Wavelet transforms decomposition, with higher regularity, yield improved performance for multiscale decomposition techniques in detection and segmentation of clustered microcalcifications (Lemaur, Drouiche et al. 2003). Combined methods such as filter banks associated to a Bayes classifier (Nakayama, Uchiyama et al. 2006) and wavelet transform associated to hidden Markov random fields (Regentova, Zhang et al. 2007) demonstrated to be satisfactory.

2.3.3 Machine learning for breast cancer CAD

Machine learning based detection/diagnosis methods for supporting semi-automated or automated breast cancer CAD systems have been developed with minor or major degree of success in the last two decades, seeking to modify the habitually qualitative diagnostic criteria into a more objective quantitative feature classification task. ML classifiers based on ANN (Songyang and Ling 2000), evolutionary genetic algorithms (Jiang, Yao et al. 2007) and support vector machines (SVM) (Singh, Kumar et al. 2006) have been demonstrating high accuracy, specially in detecting microcalcifications. Other approaches include naïve Bayes classification methods, such as in (Butler, Webb

et al. 2003), distinguishing between the diffraction patterns of normal (20 instances) and cancerous tissue (22 instances) using as input a dataset of computed features vectors from X-ray mammography scatter images (above 90% accuracy); a supervised fuzzy clustering classification technique (Abonyi and Szeifert 2003) validated with the UCI Wisconsin breast cancer dataset (*bcw*, see table 24 on page 154) to distinguish between benign and malignant cancers (95.57% accuracy); a method for rule extraction from ANN (Setiono 2000) validated on the *bcw* (98.1% accuracy); an hybrid model integrating a case-based data clustering method and a fuzzy decision tree (Fan, Chang et al. 2011) validated on the *bcw* (98.4% accuracy); a comparative study of different SVM training methods (Sweilam, Tharwat et al. 2010) that integrated: particle swarm optimization, quantum particle swarm optimization, quadratic programming, and the least square SVM method tested on the *bcw* (93.52% accuracy).

Masses are more difficult to detect than microcalcifications because the features of a mass may be obscured by or be similar to those of normal breast parenchyma. Masses are space-occupying lesion seen in more than one projection, usually characterized by its shape and margin. Regular shape masses have a higher probability of being benign, whereas irregular shape masses have a higher probability of being malignant. Some significant reported mass detection ML methods are: SVM-based featureless approach (Campanini, D Dongiovanni et al. 2004) that used two SVM classifiers. A first SVM classifier retrieves the masses candidates, and the second SVM classifier reduces the number of false positives (80% true positive detection was reported); a comparative study of logistic regression and ANN-based classifiers (Song, Venkatesh et al. 2005) using as input a dataset of features vectors extracted from ultrasound images of 24 malignant and 30 benign masses (ANN-based classifier with 0.856 ROC Az, 95% sensitivity and 76.5% specificity); an automated CAD (Bellotti, De Carlo et al. 2006) including three steps: edge-based segmentation to select the suspicious regions, eight gray-tone independent texture ROI features and a supervised two-layered trained feedforward neural network classifier was employed to classify masses in a database of 3369 mammographic images (including 2307 negative and 1062 positive cases), reporting an area under the ROC curve = 0.783±0.008 for the ROI based classification and 80% of sensitivity in mass detection. An experimental CAD system (López, Novoa et al. 2008) including five steps: ROI selection, contrast-limited adaptive histogram equalization, segmentation, selected features extraction and classification using ANN-based classifier was used to diagnosis six mammography pathological lesions classes as benign or malignant tissues, achieving 94.0% of true positives detection rate. A recent

review of existing approaches of automatic detection and segmentation of masses methods (including a single or multiples mammography views) can be found in (Oliver, Freixenet et al. 2010), and an overview of digital image processing and pattern analysis techniques addressing several areas in breast cancer CADe, including: contrast enhancement, detection and analysis of calcifications, detection and analysis of masses and tumors, analysis of bilateral asymmetry, and detection of architectural distortion can be revised in (Rangayyan, Ayres et al. 2007).

Finally, it can be concluded that ML-based methods for detecting breast cancer are achieving now a successful development degree, and these are beginning to be accepted and introduced by the medical community. However, a major effort is needed for developing more precise and robust ML-based methods, as support, to improve the performance of breast cancer CADx systems focused on diagnosis (classification) of suspicious breast cancer pathological lesions.

2.3.4 Mammography data availability and clinical acceptance

Although a large number of image features and machine learning classifiers have been developed and tested using different image databases, selecting the optimal image features and a machine learning classifier remains a challenged issue in CAD development and the performance of current commercial CAD systems still needs to be improved so that they can meet the requirements of clinics and screening centers (Park, Goo et al. 2004; Pisano, Gatsonis et al. 2005; Ciatto, Houssami et al. 2007; Metz 2008; Park, Pu et al. 2009; Oliver, Freixenet et al. 2010).

Two main publicly available annotated mammograms databases are used in the vast majority of the CAD systems in the literature: the Mammographic Image Analysis Society Digital Mammogram Database (MIAS) (J. Suckling, S. Astley et al. 1994) consisting on about 300 mammograms classified into seven different classes and the University of South Florida Digital Database for Screening Mammography (DDSM) (Heath, Bowyer et al. 2001) containing about 3000 annotated mammograms. Additionally, the Nijmegen database (Karssemeijer 1993) is also used in some studies but left often recently.

Most CAD systems presented in literature are evaluated with a few hundred cases from those databases or custom ones developed usually in collaboration with some medical institution providing anonimized patient cases. In general, together with the preprocessing required to handle the images, this makes results difficult to compare.

Apart from this, data is scarce and collecting it is a lengthy and tedious process, as requires deep involvement of specialists to segment and classify mammograms and access to potentially private data is mostly unwelcome by health sector technical staff.

2.3.5 Summary of references

Table 6 gathers selected literature sources for different aspects on breast cancer CAD and computer methods for medical imaging in general. Additionally, tables 7 and 8 show a selection of results reported in the literature using different methods for the classification of **microcalcifications** and **masses** respectively. Note how results are reported using a variety of metrics using different datasets. For instance, the notion of specificity sometimes is reported as *false positives per image* (FPI) and sometimes as a true negative rate.

Table 6: Selected general references for breast cancer CAD methods

CAD reviews	(Eadie, Taylor et al. 2011) (Sadaf, Crystal et al. 2011) (Oliver, Freixenet et al. 2010) (Sweilam, Tharwat et al. 2010) (Jiang, Trundle et al. 2010) (Boyer, Balleyguier et al. 2009) (The, Schilling et al. 2009) (Rangayyan, Ayres et al. 2007) (Nishikawa 2007) (Wei, Yang et al. 2005)
Image enhancement reviews	(Martí, Oliver et al. 2010) (Gurcan, Boucheron et al. 2009) (Krupinski 2008) (Rangayyan 2005).
Clinical acceptance	(Onega, Aiello Bowles et al. 2010) (Paquerault, Hardy et al. 2010) (Sanchez Gómez, Torres Tabanera et al.) (Wang, Jiang et al. 2010) (Sohns, Angic et al.) (Park, Pu et al. 2009) (Metz 2008) (Ciatto, Houssami et al. 2007) (Pisano, Gatsonis et al. 2005) (Park, Goo et al. 2004)

Table 7: Selected references for microcalcifications classification methods

Reference	Classification method and dataset used	Results
(Ren, Wang et al. 2011)	ANN with DDSM, 648 cases, 633 benign, 115 malign	AUC=0.975
(Yu and Huang 2010)	ANN with 20 mammograms from MIAS	94% sensitivity @ 1 FPI
(Balakumaran, Vennila et al. 2010)	Filter banks, 100 patients from DDSM	94% sensitivity @ 0.8 FPI
(Kang, Ro et al. 2009)	Foveal based method	93% sensitivity 87.5% specificity
(Papadopoulos, Fotiadis et al. 2008)	Image enhancement techniques on MIAS and Nijmegen datasets	AUC=0.932 (MIAS) AUC=0.915 (Nijmegen)
(Regentova, Zhang et al. 2007)	Wavelet transforms and Markov random fields. 40 mammograms from MIAS, 150 from DDSM	95% sensitivity @ 2.5 FPI 98% sensititivy @ 3.3% FP
(Jiang, Yao et al. 2007)	Genetic Algorithms with DDSM , 188 mammograms, 300 ROI MC present, 300 ROI no MC	AUC=0.9684

(Yongyi, Liyang et al. 2007)	Content retrieval based. Custom dataset with 104 cases of which 46 malignant and 58 benign	AUC=0.82
(Singh, Kumar et al. 2006)	SVM. Custom dataset with 435 mammograms	AUC=0.9803
(Yu, Li et al. 2006)	Wavelet filters and Markov random fields. 20 mammograms from MIAS database.	92% sensitivity @ 0.75 FPI
(Wei, Yongyi et al. 2005)	Relevance Vector Machine. Custom, dataset with 141 mammograms	90% sensitivity @ 1 FPI
(Soltanian-Zadeh, Rafiee-Rad et al. 2004)	Feature selection and wavelets. Custom dataset with 103 mammograms	90% sensitivity @ 0.5 FPI
(Butler, Webb et al. 2003)	Naïve Bayes. Custom dataset with 20 normal cases and 22 tumorous	>90% accuracy
(El-Naqa, Yongyi et al. 2002)	SVM. Custom dataset with 76 mammograms	94% sensitivity @1 FP
(Songyang and Ling 2000)	ANN with Nijmegen dataset, 40 mammograms 105 clusters of MCs	90% sensitivity @ 0.5 FPI

Table 8: Selected references for mammographic masses classification methods

Reference	**Classification method and dataset used**	**Results**
(Moon, Shen et al. 2011)	Logistic regression custom dataset with 147 cases, 76 benign, 71 malignant	AUC=0.9466 84.5% sensitivity 85.5% specificity
(Tzikopoulos, Mavroforakis et al. 2011)	SVM with 322 mammograms from MIAS dataset	85.7% accuracy
(Shi, Cheng et al. 2010)	Fuzzy SVM with 87 ultrasound images, 36 malignant, 51 benign	AUC=0.964 91.67% sensitivity 96.08% specificity
(Wang, Lederman et al. 2010)	ANN trained with genetic algorithms, custom dataset with 200 mammograms (100 positive, 100 negative)	AUC=0.754 42% sensitivity 90% specificity
(Oliver, Freixenet et al. 2010)	Review (by reimplementing) different methods (region, contour, model, cluster based) on 261 mammograms from MIAS and 89 from custom dataset	AUC=0.787 (MIAS) AUC=0.757 (custom)
(Velikova, Samulski et al. 2009)	Bayesian network on multiview custom dataset with 1063 cases (385 cancerous)	AUC=0.868
(Eltonsy, Tourassi et al. 2007)	Multiple concentric layers contours on 270 mammograms from DDSM	81% sensitivity @ 2.4 FPI
(Kom, Tiedeu et al. 2007)	Local thresholding on custom dataset with 61 mammograms	AUC=0.946 95.91% sensitivity
(Bellotti, De Carlo et al. 2006)	ANN with custom dataset 3360 images (2307 negative, 1062 positive)	AUC=0.783 80% sensitivity
(Song, Venkatesh et al. 2005)	ANN with custom dataset 24 malignant, 30 benign	AUC=0.856 95% sensitivity 76.5% specificity
(Campanini, D Dongiovanni et al. 2004)	Ensembled SVM	80% sensitivity

2.4 Conclusion

The above sections aimed at describing current developments and status in the three different areas in whose intersection this thesis contributes, namely (1) ROC analysis usage in machine learning, (2) exploiting eInfrastructures and (3) machine learning applied to breast cancer computer aided diagnosis. In summary, this thesis aims to contribute at dealing with the following issues which have not yet been addressed to its full extent in related work:

1. ROC analysis is not used systematically in machine learning. Mostly, AUC measures are used for classifier evaluation and primarily when ML is applied in specific areas such as medical imaging.

2. There is substantial evidence that error rate minimization does not necessarily yield to AUC optimization but the vast majority of ML methods are designed for error rate minimization. This makes them suboptimal when AUC is a more appropriate measure for the problem in hand (such as in medical imaging). This is partly due to the fact that AUC usage in ML started relatively recently but also because of the magnitude of the effort to required to redesign the existing methods. In addition, there is moderate evidence that AUC might be a more comprehensive metric than error rate for evaluating classifier performance.

3. eInfrastructures are in general costly to use, in terms of the effort required to adapt existing software and the available tools to interact with existing middleware and resource providers, specially in Grids. Furthermore, interoperability issues limit the practical possibility to exploit eInfrastructures different from the ones in which an application is initially deployed, hindering the reach of the invested efforts.

4. In automated breast cancer detection there have been significant results in the CADe for microcalcifications but moderate progress have occurred in CADe for masses, architectural distortions and asymmetries. CADx is still underdeveloped with respect to CADe and more work is needed in this area.

5. Machine learning applied to breast cancer CAD has obtained reasonably good results but, as shown before, still needs improvement. Systematic inclusion of ROC analysis in machine learning could contribute to results in

this area which might, in turn, become a rich application ground for ROC oriented machine learning.

Therefore, the work leading to this thesis was devised to contribute in these issues and, as such, the goals and contributions described in Sections 1.2 and 1.3. The following sections describe this work and its results. First, Chapter 3 describes a method by which existing ML methods can be adapted for AUC optimization with reasonable effort. It also provides the means for the proposed method to be practical by developing a computationally efficient AUC calculation algorithm. Then, Chapter 4 describes the software tools developed to enable effective exploitation for machine learning of computational resources provided by eInfrastructres. These tools enable both the validation of ML methods (such as what is described in Chapter 3) and its application in practice. This is precisely what is shown in Chapter 5, which uses the techniques and tools developed in a real medical environment for breast cancer CADx. This last issue constitutes the contribution of this thesis to the project within which it was initially conceived.

Chapter 3

ROC analysis for Machine Learning based classifiers

3.1 Introduction

Evidence reported by literature (see Section 2.1) shows that AUC optimization is not necessarily achieved by minimizing the error rate, which is what most machine learning methods are designed for. Although a few efforts have been devoted into developing new machine learning algorithms for AUC optimization it would be desirable to be equipped with a method to adapt existing algorithms with reasonable effort, so that the existing body of knowledge and techniques can be reused for AUC optimization. However, AUC calculation is computationally costly and this constitutes a major drawback since machine learning algorithms using AUC intensively may become too expensive to compute rendering them impractical. To sort this out firstly, Section 3.2 describes a method to efficiently approximate the AUC of any ranked dataset with arbitrary user defined precision. Then, Section 3.3 provides a general formulation for an AUC based loss function aimed at substituting the error measures used in machine learning (such as the squared error) enabling existing machine learning algorithms for AUC optimization. Then, it is incorporated and experimentally validated into different kinds of existing training algorithms for multilayer perceptrons (Ramos-Pollan, Guevara Lopez et al. 2010).

3.2 Efficient AUC error bounded approximation

Efficient AUC computation is essential when using AUC based metrics in iterative ML algorithms, since it may render good theoretical results impractical to use. In the initial experimental runs of this thesis work, it was observed that commonly used AUC calculation techniques, such as the Wilcoxon-Mann-Whitney statistic provided by Weka (Mark Hall, Eibe Frank et al. 2009), slow down 5 to 10 times the ML algorithms modified for AUC optimization described in Section 3.3. Still, it was also observed that Weka already provides a fast method with respect to other implementations of the statistic. To overcome this, and herewith presented, an efficient AUC calculation method was developed to approximate the actual AUC value to an arbitrary maximum error established by the user. It is based on discretizing the score space for each dataset element and, therefore, removing the need to full sorting which is what hinders computing performance. Additionally, it produces all necessary error metrics and partial AUC components required by the method described later in Section 3.3.

3.2.1 Definition

Table 23 in page 153 shows the notation used throughout this thesis to denote datasets and classifiers. The Wilcoxon-Mann-Whitney statistic providing an empirical measure of the AUC of dataset $S = S_P \cup S_N$ whose elements have been ranked by classifier h was given in equation 1.1 and it is recalled here:

$$AUC(S,h) = \frac{\sum_{p\in S_P}\sum_{n\in S_N}\mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P|\cdot|S_N|}$$

Now, we define the contribution of element $x \in S$ to the AUC in as

$$AUC(x,h) = \begin{cases} \dfrac{\sum_{n\in S_N}\mathbf{1}[h_{sc}(n) < h_{sc}(x)]}{|S_P|\cdot|S_N|} \; if\; x \in S_P \\ \dfrac{\sum_{p\in S_P}\mathbf{1}[h_{sc}(x) < h_{sc}(p)]}{|S_P|\cdot|S_N|} \; if\; x \in S_N \end{cases} \tag{3.1}$$

Note that in the Wilcoxon-Mann-Whitney statistic the order of the summation does not matter, therefore:

$$\sum_{x\in S} AUC(x,h) = \sum_{p\in S_P} AUC(p,h) + \sum_{n\in S_N} AUC(n,h) = 2\,AUC(S,h)$$

We also define the maximum contribution of element $x \in S$ by observing that, if x is positive, the maximum possible value for $AUC(x,h)$ is reached when the score of x is greater than all negative elements of S, and inversely when x is negative

$$AUC_{max}(x) = \begin{cases} \dfrac{|S_N|}{|S_P| \cdot |S_N|} = \dfrac{1}{|S_P|} & if\ x\ \in\ S_P \\ \dfrac{|S_P|}{|S_P| \cdot |S_N|} = \dfrac{1}{|S_N|} & if\ x\ \in\ S_N \end{cases} \tag{3.2}$$

Approximation through discretization

The notation in table 9 is used to define a discretization of the score space into contiguous, non-overlapping subintervals of equal length. Then, each element of the dataset is assigned to the subinterval containing the score given to it by a classifier (denoted by I_x^h), and positive and negative elements in subintervals below and above are counted in the corresponding definition for $below.I_x^{h,S_P}$, $below.I_x^{h,S_N}$, $above.I_x^{h,S_P}$ and $above.I_x^{h,S_N}$

Table 9: Discretization of the score space

$I = \{I_1, \dots, I_u, \dots, I_m\}$ $I_u \subset [0,1], m \in \mathbb{N}$	A set of m subintervals in the [0,1] interval. I_u denotes a generic interval of the set
$\forall\ I_u, I_v \in I, I_u \cap I_v = \emptyset$	All intervals are non overlapping
$\forall\ I_u, I_v \in I, \lvert I_u\rvert = \lvert I_v\rvert$	All intervals have the same length
$\bigcup_{I_u \in I} I_u = [0,1]$	The set of intervals fully covers the [0,1] interval
$I_u < I_v \Leftrightarrow\ \forall\ i \in I_u, \forall\ j \in I_v\ \ i < j$	Order relation between any two intervals
$I_x^h = I_u \mid h_{sc}(x)\ \in\ I_u$	Interval to which $x \in S$ belongs according to the score given by classifier h
$I_u^{h,S} = \{\, x \in S \mid I_x^h = I_u\}$	Set of elements belonging to interval I_u
$I_u^{h,S_P}\ \ I_u^{h,S_N}$	Sets of positive and negative elements belonging to interval I_u
$I_x^{h,S} = \{x' \in S \mid h_{sc}(x') \in I_x^h\}$	Set of elements belonging to the interval to which $x \in S$ belongs
$I_x^{h,S_P}\ \ I_x^{h,S_N}$	Set of positive/negative elements belonging to the same interval to which $x \in S$ belongs
$below.I_x^{h,S} = \bigcup_{I_u < I_x^h} I_u^{h,S}$	Set of elements belonging to intervals below the interval to which $x \in S$ belongs
$below.I_x^{h,S_P}\ \ below.I_x^{h,S_N}$	Set of positive/negative elements belonging to intervals below the interval to which $x \in S$ belongs

$above.I_x^{h,S} = \bigcup_{I_u > I_x^h} I_u^{h,S}$	Set of elements belonging to intervals above the interval to which $x \in S$ belongs
$above.I_x^{h,S_P}$ $above.I_x^{h,S_N}$	Set of positive/negative elements belonging to intervals above the interval to which $x \in S$ belongs

With this, the contribution of element $x \in S$ as defined in equation (3.1) is approximated by

$$AUC_{app}(x,h) = \begin{cases} \dfrac{|below.I_x^{h,S_N}|}{|S_P| \cdot |S_N|} \; if \; x \in S_P \\ \dfrac{|above.I_x^{h,S_P}|}{|S_P| \cdot |S_N|} \; if \; x \in S_N \end{cases} \tag{3.3}$$

This is, for positive elements, by counting the number of negative elements in subintervals below the one to which the positive element belongs; and for negative elements, by counting the number of positive elements in subintervals above the one to which the negative element belongs. Figure 18 below illustrates this discretization process with positive elements (in blue) and negative elements (in red) binned into the corresponding intervals. The count of positive and negative elements is also shown for each interval.

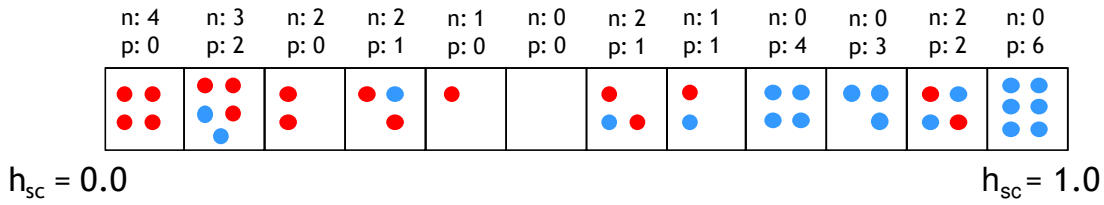


Figure 18: Discretization of AUC score space

Finally, $AUC(S,h)$, the actual dataset AUC, is approximated either by looping over the positive elements:

$$AUC_{low}(S,h) = \frac{\sum_{p \in S_P} |below.I_p^{h,S_N}|}{|S_P| \cdot |S_N|} \tag{3.4}$$

or by looping over the negative elements:

$$AUC_{low}(S,h) = \frac{\sum_{n \in S_N} |above.I_n^{h,S_P}|}{|S_P| \cdot |S_N|} \tag{3.5}$$

The function is named $AUC_{low}(S,)$ since it actually constitutes a lower bound of the actual AUC, as will become evident below. In fact, we observe that the expression $|below.I_p^{h,S_N}|$ renders the same value for all positive elements of each subinterval so we do not need to loop over all positive elements, but only over all subintervals knowing that all positive elements of such subinterval will contribute the same value through this approximation. This way equation 3.4 can be rewritten as:

$$AUC_{low}(S,h) = \frac{\sum_{I_u \in I} |I_u^{h,S_P}| \cdot |below.I_u^{h,S_N}|}{|S_P| \cdot |S_N|} \tag{3.6}$$

As all subintervals have the same length $\left|I_u^{h,S_P}\right|$ and $\left|I_u^{h,S_N}\right|$ can be calculated in one scan of the dataset S by dividing the score of each element with the chosen subinterval length, and $|below.I_u^{h,S_N}|$ can be calculated in one further scan over the set of subintervals accumulating $\left|I_u^{h,S_N}\right|$. This way, $AUC_{low}(S,h)$ computing time is proportional to $|S| + |I|$ (the number of elements in the dataset plus the number of subintervals defined). Since $|I|$ is a fixed constant established a priori, not depending on dataset size, the time complexity of this method remains as $\mathcal{O}(n)$, where $n = |S|$, as opposed to $\mathcal{O}(n \log(n))$ as required by the sorting operations usually made to calculate the Wilcoxon-Mann-Whitney statistic in equation 1.1. In fact, experiments in Section 3.2.2 below show that values for $|I|$ between 50 and 100 give produce in general the greatest speedups.

Computing the approximation error

We choose equation 3.4 to understand the error we are incurring when using it as approximation for the AUC (an analogous reasoning could be made for equation 3.5). For any positive element $p \in S_P$ the Wilcoxon-Mann-Whitney statistic uses the expression:

$$\sum_{n \in S_N} \mathbf{1}[h_{sc}(n) < h_{sc}(p)] \tag{3.7}$$

which counts the number of negative elements whose score is less than the score of p, whereas equation 3.4 uses the expression:

$$|below.I_p^{h,S_N}| \tag{3.8}$$

which denotes the number of negative elements in subintervals below the subinterval to which p belongs, without including the elements of the subinterval itself (this is why 3.4 constitutes a minimum). Therefore, the only negative elements that equation 3.8 leaves out with respect to equation 3.7, are the negative elements belonging to the same subinterval of p, denoted by I_p^h. Those elements are accounted for by equation 3.7 only if their score is less than the score of p. In the worst case (perfect relative score of positive and negative elements within I_p^h), all negative elements of I_p^h have a lower score than p itself and will be accounted for in equation 3.7 but not in equation 3.8, so the maximum number of negative elements left out by using 3.8 instead of 3.7 will be $|I_p^{h,S_N}|$ which denotes the number of negative elements of the subinterval to which p belongs. Thus, the maximum error of using equation 3.3 to approximate the contribution of a positive element $p \in S_P$ to the AUC is then:

$$AUC_{max.error}(p,h) = \frac{|I_p^{h,S_N}|}{|S_P| \cdot |S_N|}$$

Note that the maximum error is the same for all positive elements within any subinterval. So the accumulated maximum error of all positive elements within the same subinterval is:

$$AUC_{max.error}(I_u,h) = \frac{|I_u^{h,S_P}| \cdot |I_u^{h,S_N}|}{|S_P| \cdot |S_N|} \tag{3.9}$$

where, recalling the definitions made before, $|I_u^{h,S_P}|$ and $|I_u^{h,S_N}|$ denote respectively the number of positive and negative elements of subinterval I_u. And finally, we add up the maximum error of all subintervals to obtain the total maximum error incurred when using this approximation for a given dataset:

$$AUC_{max.error}(S,h) = \frac{\sum_{I_u \in I} |I_u^{h,S_P}| \cdot |I_u^{h,S_N}|}{|S_P| \cdot |S_N|} \tag{3.10}$$

Therefore, using equation 3.6, the maximum value for Az is given by:

$$AUC_{high}(S,h) = AUC_{low}(S,h) + AUC_{max.error}(S,h)$$

A naive approximation to the total AUC would be to take the midpoint between $AUC_{high}(S,h)$ and $AUC_{low}(S,h)$

$$AUC_{app}(S,h) = AUC_{low}(S,h) + \frac{AUC_{max.error}(S,h)}{2}$$

$$= \frac{\sum_{I_u \in I} |I_u^{h,S_P}| \cdot |below.I_u^{h,S_N}|}{|S_P| \cdot |S_N|} + \frac{1}{2}\frac{\sum_{I_u \in I} |I_u^{h,S_P}| \cdot |I_u^{h,S_N}|}{|S_P| \cdot |S_N|} \Rightarrow$$

$$AUC_{app}(S,h) = \frac{\sum_{I_u \in I} |I_u^{h,S_P}| \cdot \left(2|below.I_u^{h,S_N}| + |I_u^{h,S_N}|\right)}{2|S_P| \cdot |S_N|} \quad (3.11)$$

and it is this expression that we use as approximation to $AUC(S,h)$. This approximation can be calculated in the same interval scan together with $|below.I_u^{h,S_N}|$ adding no extra time complexity to the method. A pseudo code to compute this approximation is shown in algorithm 1, page 161.

Note that, in equations 3.9 and 3.10, subintervals having no negative or no positive elements produce no error and thus AUC_{app} is exact for those intervals. This means that datasets whose positive and negative elements are well distributed in different intervals tend to get better approximations.

Approximation example

An example illustrates how this approximation works with a synthetically generated dataset of 30 elements, running the proposed method by using 15 subintervals of equal length. Table 10 shows the dataset, composed of 17 positive and 13 negative elements and whose Wilcoxon-Mann-Whitney statistic for its Az value is 0.905.

Table 10: Elements of example dataset

element	1	2	3	4	5	6	7	8	9	10
class	P	P	P	P	P	P	P	P	P	P
score	0,071	0,108	0,158	0,222	0,299	0,388	0,484	0,579	0,666	0,737
element	11	12	13	14	15	16	17	18	19	20
class	P	P	P	P	P	P	P	N	N	N
score	0,782	0,798	0,782	0,737	0,666	0,579	0,484	0,388	0,299	0,222
element	21	22	23	24	25	26	27	28	29	30
class	N	N	N	N	N	N	N	N	N	N
score	0,158	0,108	0,071	0,045	0,027	0,016	0,009	0,005	0,002	0,001

Table 11 shows the components of equation 3.11 for each one of the 15 intervals. The actual AUC=0.905 lies well within the [0.900, 0.936] range determined by the AUC_{low} and $AUC_{max.error}$. The approximated AUC is therefore 0.919 and the actual error of this approximation is 0.014. Note that only intervals containing at least one positive element contribute to the AUC, and only the ones containing both positive and

negative elements contribute to the error, as expressed in equation 3.9. This shows how datasets having positive and negative elements mixed up within the same intervals get higher approximation errors.

Table 11: AUC approximation components for the example dataset

I_u	I_u low limit	I_u high limit	$\lvert I_u^{h,S_N}\rvert$	$\lvert I_u^{h,S_P}\rvert$	$\lvert below.I_u^{h,S_N}\rvert$	AUC_{low}	$AUC_{max.error}$	AUC_{app}
1	0,000	0,067	7	0	0	0,000	0,000	0,000
2	0,067	0,133	2	2	7	0,063	0,018	0,072
3	0,133	0,200	1	1	9	0,041	0,005	0,043
4	0,200	0,267	1	1	10	0,045	0,005	0,048
5	0,267	0,333	1	1	11	0,050	0,005	0,052
6	0,333	0,400	1	1	12	0,054	0,005	0,057
7	0,400	0,467	0	0	13	0,000	0,000	0,000
8	0,467	0,533	0	2	13	0,118	0,000	0,118
9	0,533	0,600	0	2	13	0,118	0,000	0,118
10	0,600	0,667	0	2	13	0,118	0,000	0,118
11	0,667	0,733	0	0	13	0,000	0,000	0,000
12	0,733	0,800	0	5	13	0,294	0,000	0,294
13	0,800	0,867	0	0	13	0,000	0,000	0,000
14	0,867	0,933	0	0	13	0,000	0,000	0,000
15	0,933	1,000	0	0	13	0,000	0,000	0,000
					TOTALS	0,900	0,036	0,919

Error bounded AUC approximation

The same process we just did by splitting the [0,1] interval and assigning all the elements of the dataset to its subintervals, can be repeated within selected subintervals to lower their $AUC_{max.error}(I_u, h)$ and then reach a priori established desired error bounds. To calculate $AUC_{app}(S, h)$ with $AUC_{max.error}(S, h) \leq \varepsilon$, having ε a user defined maximum desired error, we proceed as follows, starting with [0,1] as our first interval:

1. split current interval and distribute its elements into its subintervals
2. calculate $AUC_{app}(S, h)$ and $AUC_{max.error}(S, h)$
3. while ($AUC_{max.error}(S, h) > \varepsilon$)

   choose subinterval I_u with greatest $AUC_{max.error}(I_u, h)$ and repeat steps 1 and 2 but applied only to elements within I_u
4. end while

Figure 19 illustrates this approach, where the interval having the greatest error is chosen for further granularity in its discretization. Whenever steps 1 and 2 are

performed to further split a selected subinterval, I_u, an extra time complexity $\mathcal{O}(|I_u^{h,S}|)$ is added to our method where, recall, $|I_u^{h,S}|$ denotes the number of elements of interval I_u. In the worst, and rare, case I_u gathers all dataset elements (and all other subintervals remain empty) adding an extra $\mathcal{O}(n)$ time complexity, with $n = |S|$, the dataset size. In our validation below, test runs needed an average of five iterations (splitting the initial [0,1] interval and other three subintervals) to reach a maximum guaranteed error $\varepsilon = 0.01$, meaning that the time complexity of this method stayed in practice well below $5\mathcal{O}(n)$, since each iteration deals with a decreasing number of dataset elements. Appendix II shows the algorithms implementing the full method, which is denoted in the rest of this text as AUC_{app}.

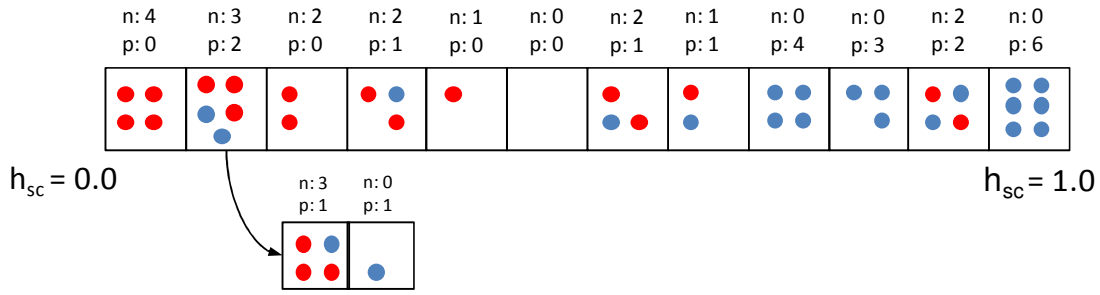


Figure 19: Iteratively reducing the AUC approximation error

3.2.2 Experimentation and Validation

AUC_{app} is to be used in situations when AUC needs to be computed intensively such as for AUC optimization in ML methods as in Section 3.3. Therefore, building evidence that it actually speeds up AUC computation is essential to use it with confidence further on. Its major theoretical weakness is that, in order to achieve a user defined maximum error in the approximation, the number of recursions needed to be performed is left unbounded, depending on the degree of "mixture" of the scores assigned by a classifier to negative and positive elements of a dataset. It is therefore necessary to understand how AUC_{app} behaves with scored datasets with different degrees of overlapping of their distributions for positive and negative elements.

3.2.2.1 Goals and metrics

The goal of the experiments carried out is two-fold:

1. Measuring whether AUC_{app} actually provides a faster AUC computation than a selected baseline implementation.
2. Gaining some insight on the conditions (scored datasets) under which AUC_{app} provides greater or lower speedups.

A straight forward implementation of the Wilcoxon-Mann-Whitney statistic for a scored dataset would require sorting and then counting how many negative elements fall below each positive element. This approach has been taken by virtually all software toolkits providing AUC metrics and, in particular, the Weka toolkit (Mark Hall, Eibe Frank et al. 2009) which is a widely used machine learning toolkit implemented in Java. Weka uses its own optimized data structures for dataset manipulation (fast vectors) so it is in general faster than other algorithms using standard Java data structures (*Lists*, *Collections*, etc). This was tested by developing a straight forward implementation of the Wilcoxon-Mann-Whitney statistic sorting regular Java collections and comparing its computation time with Weka's. This resulted in Weka being always faster in computing AUC and, in the case of large datasets (10000 elements or more), by almost one order or magnitude. In addition, Weka's source code is publicly available which allows for deeper understanding and comparison of its implementation.

Therefore, Weka's AUC is considered to be the baseline used herewith against which a Java implementation of AUC_{app} would be compared. In particular, the following class/method included in Weka 3.6.4 (released on Dec 2010) is used in this thesis to compute the AUC:

```
weka.classifiers.evalaution.ThresholdCurve.getROCArea()
```

AUC computation times of Weka and AUC_{app} were measured strictly, this is, discarding all data preparation and processing stages required before and after making the call to the actual computation method, and the speedup obtained by AUC_{app} for a given dataset S with respect to Weka is defined as the ratio of the measured times:

$$speedup\ (S) = \frac{Time(Weka(S))}{Time(AUC_{app}(S))}$$

Thus, a speedup=2 means that AUC_{app} runs twice as faster as Weka (in half the time), and a speedup=1/2 means the opposite.

3.2.2.2 Experimental setup

Two sets of experiments were designed to cover the goals just explained. First using synthetically generated datasets and, second, with real datasets scored by ML classifiers. Using synthetic datasets allows control of the generated distributions of the scores for positive and negative elements and, therefore, experiments can be designed to cover different degrees of overlapping of the distributions, aiming at gaining insight on the behavior of the speedups provided by AUC_{app}. Real datasets, which need to be scored by a certain ML classifier to produce a rank, allow understanding its applicability in practice. In this sense, experiments were designed to understand speedups with respect to (1) datasets with different number of elements, class skew and distributions of positive and negative scores and (2) the number of intervals used when running AUC_{app}

Synthetic Datasets

Synthetic scored datasets were generated by drawing the score of the positive and negative elements from two different normal distributions, denoted in their generic form by $\mathcal{N}^P(\mu_P, \sigma_P)$ and $\mathcal{N}^N(\mu_N, \sigma_N)$. A normal distribution is determined uniquely by its mean (μ) and standard deviation (σ) and if both are similar for $\mathcal{N}^P$ and $\mathcal{N}^N$ their distributions are more mixed up. This might hinder AUC_{app} performance, since recursion only happens if positive and negative scores lie within the same intervals. Therefore, to observe AUC_{app} behavior under this perspective, scored datasets were generated from different combinations of $\mathcal{N}^P$ and $\mathcal{N}^N$ having, each combination, different means and standard deviations. Table 12 below shows the values used to generate the synthetic datasets.

Table 12: Values of means and standard deviations for synthetic datasets

μ_P	0,0	0,2	0,4	0,6	0,8	1,0
μ_N	0,0	0,2	0,4	0,6	0,8	1,0
σ_P	0,01	0,05	0,1	0,2	0,3	0,5
σ_N	0,01	0,05	0,1	0,2	0,3	0,5

There are 6^4 different combinations of the values in table 12 and for each combination datasets were generated of different sizes (with 100, 1000, 10000 and 100000 elements) and different class skews (with 10%, 30%, 50%, 70% and 90% of the

elements labeled as positive). Note that AUC_{app} requires scores to lie within the [0,1] interval so, when sampling, values outside [0,1] were discarded. Due to this, the final sampled distributions of positive and negative scores are not exactly normal, but nevertheless, valid for our purposes. Finally, for each dataset AUC computation with AUC_{app} and Weka was run 10 times to allow for statistical smoothing.

The AUC of each dataset is therefore determined by four parameters $(\mu_P, \sigma_P, \mu_N, \sigma_N)$. To simplify data visualization and analysis, datasets are grouped by the difference of means $(\mu_P - \mu_N)$ and standard deviations $(\sigma_P - \sigma_N)$ of their positive and negative distributions, so that experimental analysis can be focused in the degree of mixture of positive and negative scores that, as just mentioned, might hinder AUC_{app} performance. Figure 20 shows, for each $\mu_P - \mu_N$ and $\sigma_P - \sigma_N$, the PDFs of the positive scores (blue) and negative scores (red) of a representative generated dataset of that group, together with the averaged AUC of all datasets generated in that group. It can clearly be seen how AUC is directly influenced by $\mu_P - \mu_N$ and, at a lesser degree, by $\sigma_P - \sigma_N$.

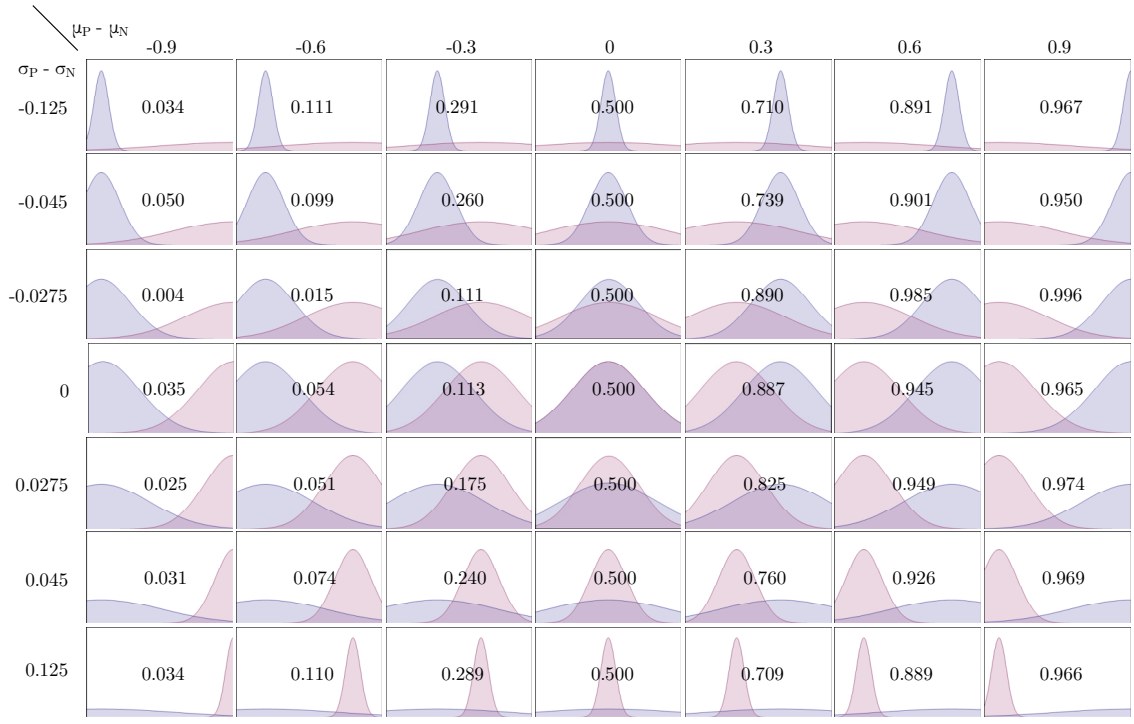


Figure 20: Representative positive and negative distributions of synthetic datasets

UCI datasets

Table 24 in page 154 shows the datasets used in this experimentation which were retrieved from the UCI Machine Learning repository (Frank and Asuncion 2010). Several datasets were chosen with a variety of class skews and sizes. Most datasets are originally binary while others distributed data into more than two classes. In these cases different classes were merged together as indicated to finally produce a single binary dataset.

For each selected UCI dataset 50 configurations of multilayer perceptron and support vector machine classifiers were trained and later used to score the dataset. The resulting scored datasets were then given to AUC_{app} and Weka for AUC computation and times were measured. AUC_{app} was run five times, each one with a different number of preset intervals (50, 75, 100, 200 and 500). This process was repeated 10 times and measured times were averaged. The goal was not to obtain well performing classifiers and, in fact, we were interested in obtaining a variety of AUC scores for each dataset since the aim is to understand the speedups obtained by using AUC_{app}. For this reason, for each classifier configuration values of the parameters (such as *learning rate* or *number of intermediate neurons* for multilayer perceptrons, or *kernel type*, *gamma* or *cost* for support vector machines) were randomly selected before each of the 50 training and scoring cycles for each dataset.

3.2.2.3 Results and discussion

For synthetic datasets, table 13 shows the average speedup obtained by AUC_{app} with respect to Weka for the different $\mu_P - \mu_N$ and $\sigma_P - \sigma_N$ ranges shown in figure 20. Then, figure 21 shows the speedup further averaged per $\mu_P - \mu_N$ (left) and $\sigma_P - \sigma_N$ (right). Each point represents the average of speedups obtained by all datasets with the same $\mu_P - \mu_N$ and $\sigma_P - \sigma_N$ values and vertical bars represent with their length one standard deviation (for the speedup).

Table 13: AUC speedup per differential mean and standard deviation range

		μP-μN range						
		-1.0 -0.75	**-0.75 -0.45**	**-0.45 -0.15**	**-0.15 0.15**	**0.15 0.45**	**0.45 0.75**	**0.75 1.0**
σP-σN range	**-0.55 -0.35**	5.69	5.46	3.68	3.80	4.10	4.95	6.41
	-0.35 -0.2	6.01	5.84	4.06	3.58	4.22	4.96	5.61
	-0.2 -0.05	6.30	5.54	4.90	3.37	5.02	6.28	7.13
	-0.05 0.05	5.54	5.96	5.44	3.11	5.48	5.53	5.96
	0.05 0.2	5.91	5.83	4.55	3.65	4.88	6.04	5.93
	0.2 0.35	6.27	5.88	4.05	4.27	4.48	6.31	6.24
	0.35 0.55	6.46	5.04	4.37	4.14	4.85	5.40	6.38

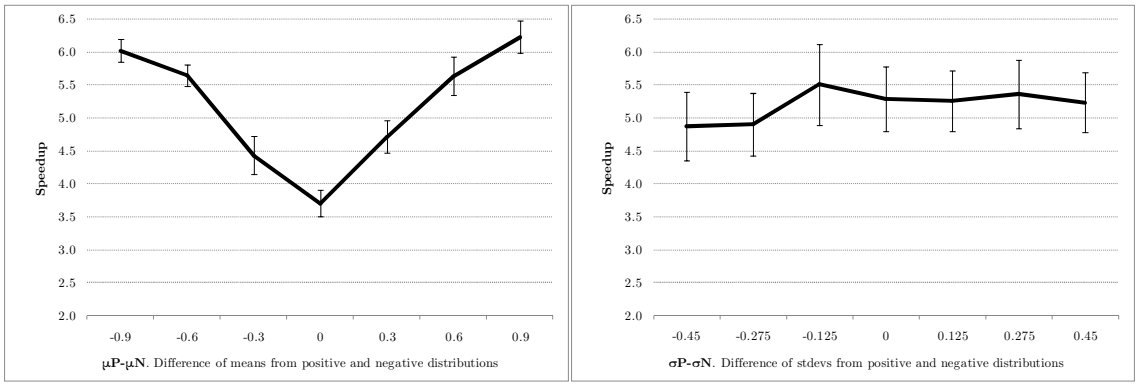


Figure 21: AUC speedup per differential mean and standard deviation range

It can be clearly seen that greatest speedups are obtained when the means for positive and negative scores are more separated which, in general, corresponds to less mixed up distributions (see figure 20). This confirms the intuition that AUC_{app} works better in these cases. Nevertheless, the average speedup is always above 3.5. Differences in standard deviation seem to have a must lesser influence.

Figures 22 and 23 show the speedup obtained with respect to dataset size and class skew. Whereas for synthetic datasets (left sides) the plot shows the averages and standard deviation for the speedup obtained for all generated datasets, in the dispersion plot for the selected UCI datasets the speedup is shown for each dataset represented as a dot when running AUC_{app} with 75 intervals.

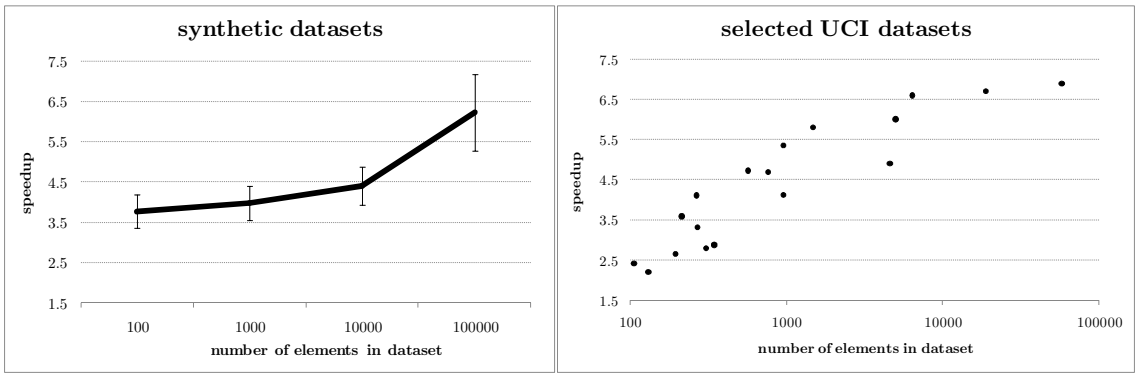


Figure 22: AUC speedup per dataset size for synthetic and UCI datasets

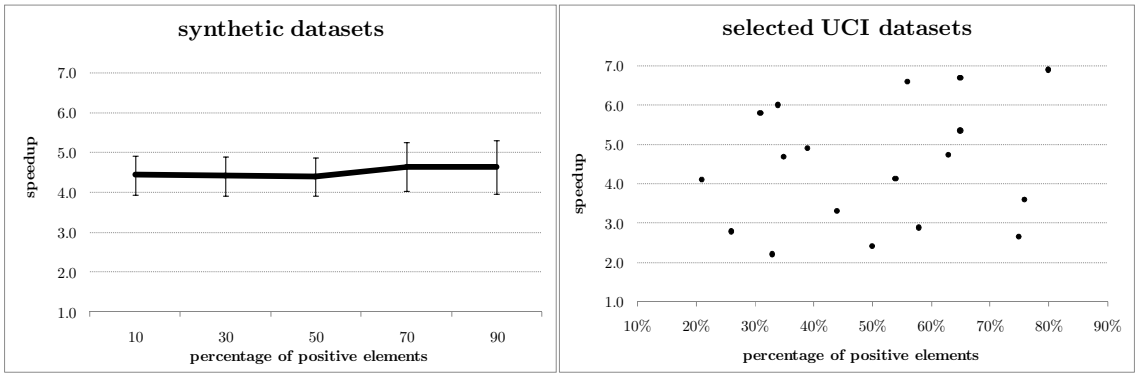


Figure 23: AUC speedup per class skew for synthetic and UCI datasets

Finally, figure 24 shows how the choice of the number of intervals with which AUC_{app} is run also influences speedup. Observe that the number of intervals is a user definable parameter when running AUC_{app} and not a dataset characteristic such as the dataset size or class skew. Therefore, the right plot shows the averaged speedup for all UCI dataset for each number of intervals and the standard deviations obtained.

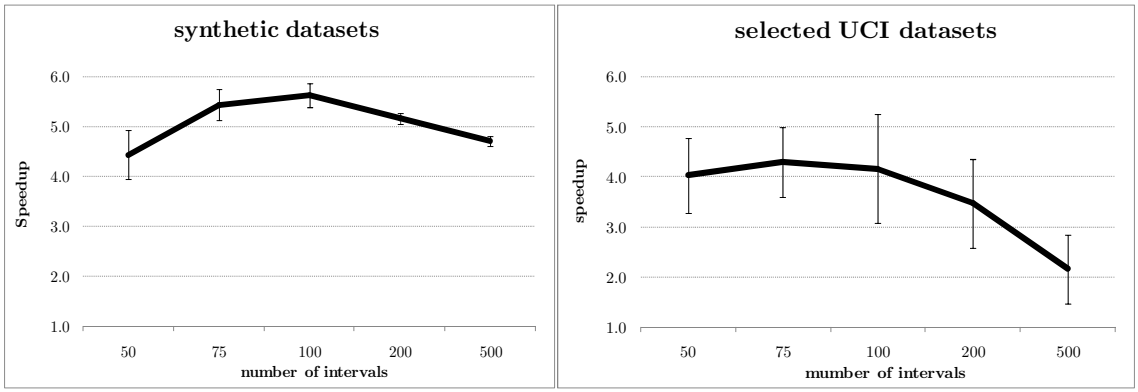


Figure 24: AUC speedup per intervals for synthetic and UCI datasets

Results show that speedups obtained by AUC_{app} are mostly influenced by dataset size and the number of intervals used to run AUC_{app}. Whereas dataset size is determined by the nature of the dataset in hand, the number of intervals is a user definable parameter and, based on this experimentation, values between 75 and 100 seem to be a reasonable choice. Class skew seems not to influence speedup.

In addition, the behavior observed in the selected UCI datasets seems to follow the one modeled by the synthetic datasets with somewhat lower speedups and greater variability, which might be due to the great difference in the sampling population in each case (several thousands of synthetic datasets and 18 selected UCI datasets). In any case, this is a result that provides the longed for confidence to effectively use AUC_{app} in the following developments of this thesis. Furthermore, these experiments were also useful to tune the number of intervals with which AUC_{app} was to be later used.

3.3 Generalizing AUC optimization in multilayer perceptron classifiers

This section provides a general formulation for an AUC error measure to be used as loss function in machine learning, where typically a squared error measure is used. Doing so, existing machine learning algorithms can be adapted for AUC optimization with reasonable effort by interchanging the error calculation routines and leaving their core logic untouched. Experimental validation was performed on different training algorithms for multilayer perceptrons showing, in the way, how to sort out their specific particularities.

3.3.1 Theoretical definition

The general definitions made in table 23 in page 153 and those in Section 3.2 for measuring $AUC(x,h)$ and $AUC_{max}(x)$ are now used to define the following error measures for individual dataset elements and for the whole dataset:

$$e^{AUC}(x,h) = 1 - \frac{AUC(x,h)}{AUC_{max}(x)}$$
$$\mathcal{E}^{AUC}(S,h) = 1 - AUC(S,h) \tag{3.12}$$

Recall that $AUC(x,h)$ represents the contribution of element x to the AUC of dataset S with respect to the scores assigned by classifier h, and $AUC_{max}(x)$, is the maximum possible contribution of element x. It would have been tempting to define $e^{AUC}(x,h) = AUC_{max}(x) - AUC(h,x)$, however, $AUC_{max}(x)$ is usually a very small value (specially for large datasets), which would make it unpractical for machine learning training purposes. Therefore, $e^{AUC}(x,h)$ and $\mathcal{E}^{AUC}(S,h)$ become, respectively, the loss function (for a single dataset element) and empirical error over the whole training set to substitute the commonly used ones in machine learning (see equations 1.3 and 1.4 on page 11):

$$\begin{aligned} L\big(y,h(x)\big) &= e^{AUC}(x,h) \\ \overline{err} &= \mathcal{E}^{AUC}(S,h) \end{aligned} \tag{3.13}$$

Observe, first, that this definition preserves the fact that $\overline{err}$ is the mean of $L\big(y,h(x)\big)$ as demonstrated in Lemma 1 (page 164) and, second, it is only the loss function or the global empirical error that needs to be substituted whenever using this approach in existing machine learning algorithms, leaving their core logic intact. Equally important, the contribution to the dataset AUC of each element, $AUC(x,h)$, can provided by the efficient AUC approximation method described in previous section through $AUC_{app}(x,h)$ as defined in equation 3.3.

AUC optimization in multilayer perceptrons

We now set forth to use this definition with different kinds of training algorithms for multilayer perceptrons (MLP) and experimentally validate it (Ramos-Pollan, Guevara Lopez et al. 2010). Without loss of generality, definitions in table 23 include those for binary MLP based classifiers having two output neurons which, for a given input vector x, produce two values, $h_P(x)$ and $h_N(x)$, within the $[nn_{\min}, nn_{\max}]$ interval. The output neuron producing $h_P(x)$ conveys a notion of the *positiveness* ascribed by h to input vector x and $h_N(x)$ its *negativeness*. Thus, their ideal values are defined as:

$$\begin{aligned} d_P(x) &= nn_{\max} \\ d_N(x) &= nn_{\min} \end{aligned} \quad if \quad x \in S_P$$

$$\begin{aligned} d_P(x) &= nn_{\min} \\ d_N(x) &= nn_{\max} \end{aligned} \quad if \quad x \in S_N \tag{3.14}$$

and fix a score metric which linearly transforms the output of the two neurons to the [0,1] interval according to equation 3.14, so that a score of 0.0 corresponds to an

ideal negative element ($h_P(x) = nn_{\min}$ and $h_N(x) = nn_{\max}$) and a score of 1.0 corresponds to an ideal positive element ($h_P(x) = nn_{\max}$ and $h_N(x) = nn_{\min}$)

$$h_{sc}(x) = \frac{h_P(x) - h_N(x)}{2(nn_{max} - nn_{\min})} + \frac{1}{2}$$

It can be easily proven that this definition ensures that $h_{sc}(x)$ stays within the [0,1] interval. In fact, for AUC purposes this restriction is not strictly needed as what matters is the relative ordering between positive and negative dataset elements induced by the score h_{sc} assigned to each one. Commonly, a distance metric measures the error at the neuron's output with respect to the ideal output:

$$\begin{aligned} \Delta_P(x,h) &= d_P(x) - h_P(x) \\ \Delta_N(x,h) &= d_N(x) - h_N(x) \end{aligned} \tag{3.15}$$

and a root mean square error defines the loss function and empirical error:

$$\begin{aligned} e^{RMS}(x,h) &= \sqrt{\frac{\Delta_P(x,h)^2 + \Delta_N(x,h)^2}{2}} \\ \mathcal{E}^{RMS}(S,h) &= \frac{\sum_{x \in S} e^{RMS}(x,h)}{|S|} \end{aligned} \tag{3.16}$$

so that $\overline{err} = \mathcal{E}^{RMS}(S,h)$ and $L\big(y,h(x)\big) = e^{RMS}(x,h)$ preserving the fact that $\overline{err}$ is the mean of $L\big(y,h(x)\big)$. Then, $e^{RMS}(x,h)$ is mapped to each output neuron by simply using the distance metric of equation 3.15 as follows

$$e_P(x,h) = e_P^{RMS}(x,h) = \Delta_P(x,h)$$

$$e_N(x,h) = e_N^{RMS}(x,h) = \Delta_N(x,h)$$

At this point it is relevant to distinguish two kinds of MLP training algorithms: (1) those using the loss function iteratively at each dataset element x through $e_P(x,h)$ and $e_N(x,h)$, the error measures at each output neuron, and (2) those using $\overline{err} = \mathcal{E}(S,h)$, the global empirical error measure for the whole dataset. We denote the first kind of algorithms by *element error training algorithms* and the second kind by *dataset error training algorithms*. Notice that *dataset error training algorithms* only use $\mathcal{E}(S,h)$ regardless how it is calculated. In the case above (equation 3.16), it happens that $\mathcal{E}^{RMS}$ directly uses e^{RMS}, but this might not be necessarily the case.

Now, to enable for AUC optimization existing MLP training algorithms using $\mathcal{E}^{RMS}$ or e^{RMS}, we use instead $\mathcal{E}^{AUC}$ and e^{AUC} as defined in equation 3.12. Out of the four MLP training algorithms mentioned in table 2 (page 14) both feedforward backpropagation (*ffbp*) and resilient propagation (*ffrp*) are *element error training algorithms*, whereas feedforward simulated annealing (*ffsa*) and genetic algorithms (*ffga*) are *dataset error training algorithms*, which provides a rich test ground for validating the proposed substitution.

However, one final twist is required, since $e^{AUC}(x,h)$ still needs to be mapped to each output neuron of a binary MLP classifier so that the error at each neuron can be backpropagated in the case of element error training algorithms (*ffbp* and *ffrp*). This is done by defining $e_P(x,h) = e_P^{AUC}(x,h)$ and $e_N(x,h) = e_N^{AUC}(x,h)$ in the following way:

$$e_P^{AUC}(x,h) = e^{AUC}(x,h) \cdot \frac{\Delta_P(x,h)}{|\Delta_P(x,h)| + |\Delta_N(x,h)|}$$

$$e_N^{AUC}(x,h) = e^{AUC}(x,h) \cdot \frac{\Delta_N(x,h)}{|\Delta_P(x,h)| + |\Delta_N(x,h)|}$$

Therefore, the loss function, $L\big(y,h(x)\big) = e^{AUC}(x,h)$ is distributed between the positive and negative neurons according to how far each one is from their ideal value, maintaining the direction of the distance metric (Δ_P and Δ_N). This way, dataset elements having a perfect AUC score, where $AUC(x,h) = AUC_{max}(x)$, do not produce any error even if the output values of the output neurons are not the ideal ones. This argument is at the root of the analysis in (Cortes and Mohri 2004) claiming that error rate minimization does not necessarily yield to AUC optimization. In the limit case, where $|\Delta_P(x,h)| + |\Delta_N(x,h)| = 0$, we establish both $e_P^{AUC}(x,h) = 0$ and $e_N^{AUC}(x,h) = 0$.

With this, the proposed $\mathcal{E}^{AUC}(S,h)$ error measure is injected for dataset error training algorithms (*ffsa* and *ffga*) replacing $\mathcal{E}^{RMS}(S,h)$, whereas $e_P^{AUC}(x,h)$ and $e_N^{AUC}(x,h)$ replace $e_P^{RMS}(x,h)$ and $e_N^{RMS}(x,h)$ respectively for element error training algorithms (*ffbp* and *ffrp*). This is achieved by simply substituting the error calculation routines without altering the rest of the algorithm logic.

3.3.2 Experimentation and Validation

The error calculation routines of each of the four MLP training algorithms mentioned (*ffbp*, *ffrp*, *ffsa* and *ffga*, as implemented in the Encog toolkit) were modified to inject

$\mathcal{E}^{AUC}$ and e^{AUC} as just defined. The corresponding modified algorithms are named *ffbproc*, *ffrproc*, *ffsaroc* and *ffgaroc* and, since only the error calculation routine is modified, they accept the exact same training parameters as their unmodified counterparts. The term *original algorithms* will be used to refer generically to the algorithms as originally delivered by Encog (using RMS based error measures), whereas the term *modified algorithms* refers to their counterparts modified as described in previous section to target for AUC optimization. A *training configuration* is a set of particular values of the training parameters each MLP training algorithm accepts. Therefore, the same training configuration can be used by the original and the modified algorithms facilitating comparisons. For instance, *learning rate* and *momentum* are the training parameters accepted by both *ffbp* and *ffbproc*.

3.3.2.1 Goals and metrics

Experimentation seeks to provide evidence that the modified algorithms actually obtain better AUC measures that their original counterparts. A variety of datasets and training configurations was devised so that experimentation was rich enough to provide evidence statistically meaningful. For each dataset, the same configurations were trained for each algorithm (such as *ffbp*) and its modified counterpart (*ffbproc*), and the AUC obtained by all configurations for each one were averaged. Therefore, each dataset and training algorithm produced two AUC measures, one representing the average of all the configurations trained with the original algorithms and another one representing the average of the same configurations trained with its modified version. Direct comparison of both measures across all datasets is to give, for each algorithm, a notion of the difference in AUC obtained by the method proposed above.

3.3.2.2 Experimental setup

The following datasets from the UCI repository described in table 24 in page 154 were used: *bcwd*, *echocard*, *glass*, *haber*, *heartsl*, *liver*, *mmass*, *park*, *pgene*, *pimadiab* and *spectf*. The criteria to select those datasets was: (1) they are binary datasets, (2) they provide a diversity of skews in their class distribution, (3) they represent classification tasks of different nature and (4) they contain less than 1000 elements, which makes MLP training affordable from a computational point of view. Class skew was considered important since AUC is known to be insensitive to class distribution and this might affect optimization obtained.

For each dataset, a set of MLP configurations was defined for each original algorithm and its modified version (*ffbp/ffbproc, ffrp/ffrproc, ffsa/ffsaroc, ffga/ffgaroc*). The BiomedTK/C3 software frameworks (see Section 4) were used to manage MLP configurations and send them to a Grid computer cluster for training. MLP configurations are defined in text files with a specific format. Figure 25 shows the configuration file used for *ffsa/ffsaroc* MLPs with the SPECTF dataset, which results in 24 configurations for *ffsa* and another 24 configurations for*ffsaroc*.

```
explore.neurons.input                = 44
explore.neurons.output               = 2
explore.neurons.layer.01             = 89:178
explore.neurons.layer.02             = 44:132

explore.activation.function          = sigm

explore.trainingsets                 = spectf
explore.trainengines                 = encog.ffsa:encog.ffsaroc
explore.validation                   = cvfolds 10

explore.encog.ffsa.starttemp         = 30:100
explore.encog.ffsa.endtemp           = 2:10
explore.encog.ffsa.cycles            = 50
explore.encog.ffsa.stop.epochs       = 300
explore.encog.ffsa.stop.error        = 0.0001

explore.encog.ffsaroc.starttemp      = 30:100
explore.encog.ffsaroc.endtemp        = 2:10
explore.encog.ffsaroc.cycles         = 50
explore.encog.ffsaroc.stop.epochs    = 300
explore.encog.ffsaroc.stop.error     = 0.0001

explore.numberofjobs                 = 40
```

Figure 25: Configurations evaluated to compare FFSA vs. FFSAROC.

Configurations include MLPs with one or two hidden layers, with 89 or 178 neurons in the first hidden layer, with the parameter *start-temperature* set to 30 or 100, etc. Similar configurations sets were defined for each algorithm and dataset, fixing the particular parameters of each training algorithm to be the same for all datasets and varying only the number of input neurons according to the dataset input features while keeping the same proportions in the number of neurons of the hidden layers with respect to the input layer as in the example in figure 25. Each MLC configuration was trained with 10-fold cross-validation. All together, 180 MLC configurations were trained for each of the 12 datasets, 90 MLC configuration corresponding to the original algorithms and 90 to their corresponding modified versions. In total, 2160 MLC

configurations were trained on a Grid cluster with 50 computers which, dedicated, took about 4 full physical days.

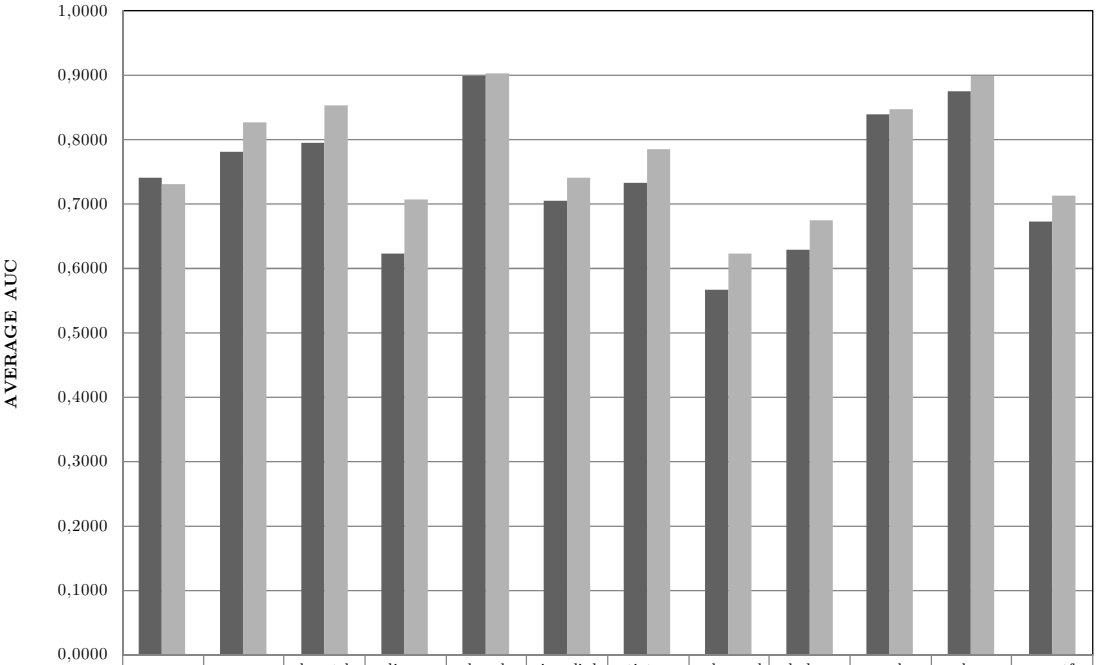
3.3.2.3 Results and discussion

Tables 25 and 26 (pages 155 and 156) summarize experimental results. For each dataset and training algorithms, results are processed and averaged in following way:

1. Each MLP configuration is trained both with the original algorithm from Encog and its modified version.
2. AUC results from all configurations trained with the original algorithms are averaged and its standard deviation is calculated. This is shown in column 'ORIGINAL' for each algorithm.
3. AUC results from all configurations trained with the modified algorithms are averaged and its standard deviation is calculated. This is shown in column 'MODIFIED' for each algorithm.
4. The percentage of improvement of the averages (positive or negative) obtained by the modified version (column 'MODIFIED') is calculated with respect to the original version (column 'ORIGINAL). This is shown in column 'IMPROV' through the following formula:

$$improv = 100 \times \frac{MODIFIED_{avg} - ORIGINAL_{avg}}{ORIGINAL_{avg}}$$

5. Table 25 in page 155 summarizes the results per training algorithm and dataset, whereas table 26 in page 156 aggregates them in total (column 'OVERALL', shown graphically in figure 26) and per category of algorithm (dataset error based or element error based). Finally, table 27 in page 156 provides some correlation measures between different obtained variables.



	pgene	mmass	heartsl	liver	bcwd	pimadiab	tictac	echocard	haber	park	glass	spectf
ORIGINAL	0,7425	0,7818	0,7950	0,6226	0,9005	0,7048	0,7337	0,5668	0,6293	0,8404	0,8761	0,6735
MODIFIED	0,7316	0,8273	0,8541	0,7075	0,9035	0,7418	0,7863	0,6239	0,6747	0,8469	0,8997	0,7127

Figure 26: Averaged AUC for MLP algorithms modified for AUC optimization

As it can be observed, there is a generalized AUC improvement by our proposed method, having occasional degradations in particular datasets (mostly in *pgene*, *bcwd* and *park*). The global averaged improvement is 5.86% (table 26 page 156) with a great variability on each dataset and algorithm. It can also be observed that improvement is greater in *element error training algorithms* (*fffbp* and *ffrp*) than in *dataset error training algorithms*, although this might be due to the fact that these later ones tend to give better results as shown by the correlation between *improve* and *ORIGINAL AUC avg* in table 27 line 2 (improvement is greater when ORIGINAL AUC is lower). However, it is important to acknowledge that this last observation might be biased by the way *improv* is defined since greater AUC leave less room for improvement.

Other interesting observations are the following:

- Both AUC averages and standard deviations are strongly correlated between the original algorithms and their modified versions (table 27 lines 4 and 5). Notice that small standard deviations result from MLC configurations producing similar AUC scores (all configurations classify the dataset as good or as bad), whereas larger standard deviations result from some MLC configurations producing significantly better AUC scores than others. The strong correlations observed in averages and standard deviations leads to

think that modified algorithms behave similarly to the original ones in the sense that they respond in the same way to dataset particularities (difficulty or easiness to classify).

- Class skew seems to have little influence on improvement (table 27 line 1), or at least in a non-homogeneous way across the different training algorithms.
- Except in the case of *ffbp*, there seems to be a significant correlation (table 27 line 3) between the standard deviation of the ORIGINAL AUC and the improvement obtained by our method in the positive direction (increasing standard deviation with increasing improvement). Large standard deviations may occur in many scenarios, such as when a dataset is hard to separate and well performing MLP configurations are scarce. The observed correlation might suggest that our method could be more appropriate in these situations to increase overall MLP AUC performance.

All these issues might be subject of further research, seeking still stronger statistical evidence to support the hypothesis outlined.

3.4 Conclusion

This Chapter presented the major theoretical contribution of this thesis (see Section 1.4): a new method to insert AUC based error metrics in existing machine learning algorithms for AUC optimization and validated it with different kinds of multilayer perceptrons. In practical terms, the proposed approach only requires the substitution of the error computing functions of the underlying training algorithms, respecting their core logic. Experimental evidence demonstrated a consistent improvement in AUC in multilayer perceptrons through a variety of datasets and training algorithms requiring little coding effort. In addition, and equally important, an efficient AUC calculation method has been developed ensuring training remains computationally affordable when using the proposed AUC based error metrics. Finally, it can be concluded that the newly developed AUC error metrics show a consistent behavior in both its theoretical definition and experimental results.

As it could be observed, experiments carried out specially in Section 3.3 required extensive access to computer power such as that provided by eInfrastructures (see Sections 1.1.3 and 2.2) to train different sets of parameter configurations throughout a

variety of datasets and training algorithms. Without such, providing evidence to the claim that the proposed method actually improves AUC would simply be impossible. However, as mentioned in Chapter 2, access to eInfrastructures is not straight forward and a set of software tools had to be developed to undertake the experimental endeavor. Given the project context within which this thesis was developed (see page 20), rather that developing something *ad-hoc* for this purpose the goal was to develop a general purpose tool to exploit eInfrastructures for machine learning, allowing an agile management of datasets, configurations and algorithms over the computer resources available. This would allow an efficient classifier and CAD construction cycle as data is being produced by the project. These tools are referred to as the *BiomedTK/C3 software frameworks* and they are the subject of next chapter.

Chapter 4

Exploiting eInfrastructures for Machine Learning Classifiers

4.1 Introduction

As shown in previous section, seamless access to computing resources is key to validate new machine learning methods. This is even more the case when machine learning is to be applied to specific problems, such as for breast cancer CAD as happens in the case of the project within which this thesis was motivated and described in Chapter 5. Confronted with the task of finding well performing machine learning classifiers for specific data, researchers need to evaluate many parameters configurations for different kinds of classifiers acting on a variety of datasets generated from the original data (such as the ones produced by using different data preprocessing techniques, etc.). This results in a *classifier development lifecycle* through which researchers build knowledge on what classifier configurations and data preprocessing options suit better their problem in hand. Access to computing resources is required to materially be able to train classifiers and process data, but it is the agility with which those computing resources can be harnessed that determines how efficiently that knowledge can be built and its reach. This agility refers to the capability of efficiently setting up datasets and classifier configurations, evaluating them on computing resources, analyze their results, refine or reprocess datasets and configurations, evaluate and analyze them again and so on.

Aware of the common difficulties in accessing computing resources available through existing eInfrastructures (see Section 2.2), two software frameworks were developed to reduce their utilization effort cost and simplify classifier development lifecycles in the sense just described, providing the tools to exploit computing resources in a systematic manner for discovering and evaluating machine learning classifiers for data mining and, specifically, for biomedical data (Ramos-Pollan, Guevara-López et al. 2011). The *Biomedical Data Analysis Toolkit* (**BiomedTK**) enables the definition, management and execution of explorations of parameter configurations for third party classifier implementations for data mining, whereas the *Cloud Computing Colonies Framework* (**C3**) enables the maintenance of colonies of Job Agents over distributed and heterogeneous e-Infrastructures, providing a fast job submission service and a transparent application deployment mechanism.

In this sense, a wide interpretation of the notions behind Cloud and Grid computing is herewith adopted and, through BiomedTK/C3, we show (1) how existing industry standards, mostly Java based, can be used to leverage resources in an agile, affordable and efficient manner, both for the end user, and the resource provider (decoupling to a great extent user application specifics from resource providers and performing, at a much simpler scale, what virtualization obtains in existing Cloud Computing implementations) and (2) how Grid resources can be used to provision, in a basic manner, Cloud characteristics such as *On-demand Self-service*, *Resource Pooling* and *Rapid Elasticity* (see page 43).

4.2 The Biomedical data analysis ToolKit (BiomedTK)

The Biomedical Data Analysis Toolkit (**BiomedTK**) is a Java software tool that exploits third party libraries for data analysis augmenting them with methods and metrics commonly used in the biomedical field. In addition, it provides the means to massively search, explore and combine different configurations of machine learning classifiers provided by the underlying libraries to build robust data analysis tools. It is possible to manipulate datasets, train Multilayer Perceptrons (MLP), and Support Vector Machines (SVM) based binary and multiclass classifiers with many different configurations, search for best ensemble classifiers, generate different types of ROC curve analysis, etc. BiomedTK uses the Cloud Computing Colonies Framework (**C3**) described in Section 4.3 to harness seamlessly the resources scattered throughout distributed computing infrastructures of different nature. In addition, it is possible to

manipulate datasets, including export/import to/from data formats of commonly used applications, allowing users to feed BiomedTK with datasets preprocessed by other tools to, for instance, filter, or transform the data, normalize it, reduce its dimensionality, etc. For researchers, it offers a command line interface to access its functionality (manage datasets, launch classifier explorations, analyze results, etc. see table 28 in page 157 for a list of BiomedTK commands). For programmers, it offers a simple API (*Application Programming Interface*) so that new data analysis engines can be integrated in a modular manner with reasonable effort (Ramos-Pollan, Guevara-López et al. 2011).

4.2.1 BiomedTK engines and basic elements

BiomedTK integrates engines from the Encog toolkit (Heaton 2010) implementing multilayer perceptrons, the libsvm toolkit (Chang and Lin 2001) implementing support vector machines and the multilayer perceptron Encog engines modified for AUC optimization as explained in Chapter 3. Table 14 lists the engines currently integrated in BiomedTK along with the parameters each one accepts. Particular values of those parameters for a specific engine constitute a *classifier configuration* for that engine. For a given data analysis task in hand, classifier design amounts to finding configurations yielding acceptable performance results and, therefore, the researcher is often confronted with the need to explore and evaluate several classifier configurations. For any engine, BiomedTK allows the evaluation of binary classifiers through plotting ROC curves and computing their AUC (Fawcett 2006; Yoon 2007) offering the bi-normal distribution method as provided by JLABROC4 (Eng 2011), the Wilcoxon-Mann-Whitney statistic provided by WEKA (Mark Hall, Eibe Frank et al. 2009) and the approximation method proposed in Section 3.2.

Table 14: Machine learning engines integrated in BiomedTK

engine	description	accepted parameters	source
ffbp	multilayer perceptron trained with backpropagation	ANN structure, learn rate, momentum	encog
ffbproc	ffbp modified for AUC optimization	ANN structure, learn rate, momentum	modified encog (see Section 3.3)
ffrp	multilayer perceptron trained with resilient propagation	ANN structure (no more params)	encog

ffrproc	ffrp modified for AUC optimization	ANN structure (no more params)	modified encog (see Section 3.3)
ffsa	multilayer percptron trained with simulated annealing	ANN structure, start temp, end temp, cycles	encog
ffsaroc	ffsa modified for AUC optimization	ANN structure, start temp, end temp, cycles	modified encog (see Section 3.3)
ffga	feed forward ANN trained with genetic algorithms	ANN structure, population size, mating size, survival rate	encog
ffgaroc	ffga modified for ROC optimization	ANN structure, population size, mating size, survival rate	modified encog (see Section 3.3)
csvc	cost based Support Vector Machine	kernel, cost, degree, gamma, coef0, weight, shrink, estimates	libsvm
nusvc	ν Support Vector Machine	kernel, nu, degree, gamma, coef0, shrink, estimates	libsvm

The following summarizes the basic elements of BiomedTK:

Dataset (train/test datasets): Contains the data to analyze. A dataset is made of a set of elements (or instances), each one containing a set of numerical features used as input values for classifiers and, optionally, an ideal class to which it belongs for supervised training (expected classifier output). Depending on validation strategies, elements of a dataset are usually split into two subsets, one for training and one for testing. The training subset is used in training classifiers built to distinguish automatically the classes to which each element belongs. The test subset is used to measure the generalization capabilities of classifiers when applied to unseen data.

Binary Dataset: A dataset whose elements belong to only two classes. As opposed to a **Multiclass Dataset**, whose elements may belong to several (more than two) classes. As part of its functionality, BiomedTK provides the tools to create multiple binary training sets for a given multi-class training set, each one to build a classifier specific for each class.

Engine: Engines encapsulate third party classifiers (such as MLPs from Encog or SVMs from libsvm). Each engine accepts a different set of parameters for training (MLP structure specification, learning parameters, etc.)

Engine or **Classifier configuration:** An engine configuration specifies the parameters with which a particular engine is used to train given a dataset. For instance, a configuration of a MLP might specify an Artificial Neural Network (ANN)

with 3 layers having 10 neurons each, using the sigmoid activation function, with 0.1 as learning rate and 0.5 as momentum over 10000 iterations.

Exploration: An exploration over a dataset defines a *set of classifier configurations* to train in batch mode in order to find the one(s) that best classify the dataset, or to later use them to build ensemble classifiers.

Jobs: Each exploration is split into a number of user defined jobs. Each job will train a subset of the engine configurations defined in a given exploration. Jobs can be then executed in sequentially over the same computer or in parallel over a distributed computing infrastructure.

BiomedTK interfaces seamlessly with distributed computing resources made available through C3 (see Section 4.3) providing commands to launch and manage explorations to C3 job agents, enabling researchers to harness computing power in an agile manner and allowing them to gradually gain understanding on what engine configurations better classify their data. This constitutes the basis of the exploration method described below.

4.2.2 BiomedtTK explorations

The classifier exploration method supported by BiomedTK and herewith described, is aimed at addressing two issues: 1) the need to manage, in a simple and unified way, many classifier configurations integrating third party engine implementations of different nature (implementing different machine learning models such as ANNs, SVMs, etc. on different programming languages such as C, Java, etc.) and 2) the need to efficiently exploit distributed resources required to evaluate such a diversity of classifier configurations. In this sense, *efficiency* denotes both an economy in the researcher's effort to setup and run experiments and a rational usage of available computing resources. BiomedTK supports this method through a series of configuration artifacts and tools that together with C3 constitute the material means through which researchers can efficiently use eInfrastructures to perform complex explorations of classifier configurations.

Explorations of classifier configurations are the key element of this method. Through them, researchers can gradually build understanding on the search space of possible classifiers (engine configurations) for a given dataset. With BiomedTK explorations are defined in text files with a specific format. Figure 27 shows an example exploration file

producing configurations to train classifiers for two datasets derived from the original BREAST-TISSUE dataset from the UCI repository with the engine using *ffrp*, *ffsaroc*, *ffga* and *nusvc* (see Table 14, page 83).

```
explore.neurons.input       = 9
explore.neurons.output      = 2
explore.neurons.layer.01    = 18:36
explore.neurons.layer.02    = 9:18
explore.neurons.layer.03    = 5:9


explore.activation.input    = tanh
explore.activation.output   = tanh:sigm
explore.activation.layer.01 = tanh:sigm
explore.activation.layer.02 = tanh
explore.activation.layer.03 = tanh


explore.nblayers.fixed      = yes


explore.datasets = BREAST-TISSUE.CAR
                   BREAST-TISSUE.NORM


explore.stop.error          = 0.1
explore.stop.epochs         = 2000
```

```
explore.engines    = ffrp:ffga:
                     ffsaroc:nusvc

explore.validation          = testpct 40

explore.numberofjobs        = 50

explore.encog.ffga.matepercent  = 0.5
explore.encog.ffga.percentmate  = 0.2
explore.encog.ffga.population   = 100:200


explore.encog.ffsaroc.starttemp = 10:15
explore.encog.ffsaroc.endtemp   = 2:4
explore.encog.ffsaroc.cycles    = 100

explore.libsvm.nusvc.kernel        = rbf:pol
explore.libsvm.nusvc.nu            = 0.4:0.35
explore.libsvm.nusvc.degree        = 2:3:4
explore.libsvm.nusvc.gamma         = 0.03:0.035
explore.libsvm.nusvc.coef0         = 0.0
explore.libsvm.nusvc.shrink        = yes
explore.libsvm.nusvc.probestimates = yes
```

Figure 27: Example BiomedTK exploration file

For MLP based engines, this exploration will generate configurations with ANNs having three hidden layers with 18 or 36 neurons in the first hidden layer, 9 or 18 in the second one, different neuron activation functions, etc. It also contains classifier specific parameter sweeps. For instance this exploration generates *ffsaroc* configurations with the *starttemp* parameter set to 10 and 15, the *endtemp* parameter set to 2 and 4, *nusvc* configurations with *radial basis* and *polynomial* kernel functions, etc. This exploration generates, for each dataset, 32 ANN based configurations for the *ffrp* engine (it is an engine with no parameters, 32 is the number of combinations of MLPs generated with the specified layers, neurons and activation functions per layer), 64 configurations for the *ffga* engine, 128 for *ffsaroc* and 24 for *nusvm* (a SVM engine and, therefore, not using ANN structure specifications). In total this exploration generates 496 engine configurations (248 for each dataset) split into 50 jobs as specified in the same exploration file. This way, 49 jobs will train 10 engine configurations, and one job that will train 6 configurations.

Exploration jobs are sent to computing resources for training the classifier configurations they contain. Explorations can be made as large or small as desired, depending their feasibility on the capacity of the available computing resources. Training times for different classifier engines vary greatly depending on dataset size,

parameters, MLP complexities, etc. When confronted to a new dataset for which one wants to explore classifier configurations, the following method is proposed:

1. Start with small explorations launched locally (on the desktop computer) to get a sense on the computing times needed by each different classifier and their initial performance.
2. Perform increasingly larger explorations, first locally and then over a larger computing infrastructure, to understand what classifier parameters might work better with each classifier.
3. Devise very large explorations (either by number of configurations and/or by computing time each configuration takes) and send them to a large computing infrastructure.

For this method to be effective, an agile and fast interaction between the researcher and the computing infrastructure becomes essential, so that, if computing resources are available, multiple explorations can be tested in reasonable time. It is fundamentally this agile interaction that allows researchers to efficiently build exploration strategies and gain understanding on what classifiers suit better a certain problem in hand.

For performing explorations BiomedTK offers the researcher two options: (1) launch the jobs sequentially over his local machine (desktop), or (2) submit them to C3 Job Agents running on available computing resources. Results (accuracy and AUC measures for test and train data) of each trained engine configuration are stored in a database that can be later inspected through standard SQL sentences. A command line tool exposes BiomedTK functionality to researchers. There are commands to import and manipulate datasets, to create a job set from an exploration file, to launch locally and exploration, to launch and monitor explorations to an eInfrastructure made available through C3, to inspect and test exploration results, to build ensemble classifiers, etc. See table 28 in page 157 for a list of BiomedTK commands. In practice, when faced with the task of exploring MLC configurations for a given dataset, BiomedTK enables researchers to cycle through the following steps

1. make initial exploration issuing BiomedTK commands *jobset + launch* (local launch)
2. inspect database for results
3. refine and enlarge initial exploration by issuing commands *jobset + c3.prepare + c3.submit*

4. inspect database for results
5. repeat from step 1 or step 3

Figure 41 in page 158 shows a typical BiomedTK session, importing a dataset and launching an exploration locally. BiomedTK and the method it supports are focused on providing the means to exploit eInfrastructures for exploring classifier configurations, regardless the sources and previous manipulations of data. For this purpose, BiomedTK provides basic functionality for data normalization and tools for importing and exporting data to commonly used formats (CSV, WEKA arff, etc.), so that researchers can use their usual tools for data preprocessing before feeding datasets to BiomedTK.

Validation

When performing explorations BiomedTK implements several validation strategies for the researcher to use. Validation is a key issue to understand the applicability and significance of classification results and was addressed in Chapters 1 and 2. Table 15 lists the validation procedures available in BiomedTK through the value of the `explore.validation` field in any exploration configuration file (see example in figure 27).

Table 15: Validation procedures available in BiomedTK

field value	description
asints	Uses the test/train split as specified in the dataset stored in the database, where each dataset element is tagged as either *test* or *train*. This tag can be set manually by the researcher by using standard SQL tools on the database, or also through the BiomedTK command `split.dataset`, researchers can establish a certain test/train split percentage for the dataset. In this case, the split is stratified (the class composition percentage is preserved in the generated train and test datasets)
cvfolds N	Uses cross validation with N folds each fold using one N^{th} of the dataset elements for testing and the rest for training. This split is also stratified
one2all	Uses leave one out cross validation (LOOCV). It is equivalent to using cvfolds with N equal to the number of elements in the dataset minus one.
testpct N	Before training each classifier configuration a stratified split is made with N% of the elements for testing. This implies that each classifier configuration is trained with a different set of elements (although always representing the same stratified percentage of the dataset). This enables bootstrapping validation through repeating this process several times and then averaging (see `repeats` exploration parameter in table 16)

Other exploration options

Additionally, a few other parameters are accepted by BiomedTK within exploration files to fine tune its behavior as shown in Table 16.

Table 16: Additional BiomedTK exploration parameters

name	description
probeperiod	number of training iterations between which BiomedTK reports training progress on stdout
store.trainelements	specifies if the list of dataset elements used for training are stored in BiomedTK DB together with the classification results of each trained classifier configuration
store.engine	whether to store the trained classifier (fitter model) of each configuration. Storing it allows to apply it to other datasets later on.
store.scores	whether to store in the DB the scores assigned by a classifier to dataset elements.
store.cvfolds	whether to store in the DB the classification results of each cross validation fold or only the final averaged cross validation result
plot.testroc	whether to plot a ROC curve for test element for each trained classifier configuration. ROC curves are stored as PNG files in the current directory
repeats	number of times each classifier configuration is trained. Used in conjunction with *testpct* validation enables bootstrapping validation
min.errortrend	as training progresses for any classifier engine, BiomedTK measures how much the training error has been reduced in the last 15% of the training process. If the error trend falls below the value specified in this parameter then BiomedTK stops training. This usually enables to detect stalled training processes.
skip.trained	before training a classifier configuration BiomedTK checks whether a result already exists in the DB for this configuration. If this is the case and this parameter is set to "yes" then BiomedTK does not train this configuration. This allows completing unfinished explorations.

4.2.3 BiomedTK architecture

BiomedTK is a software framework developed as a stand-alone Java application which fully supports the cycle explained in previous section. Figure 28 shows the architecture of BiomedTK.

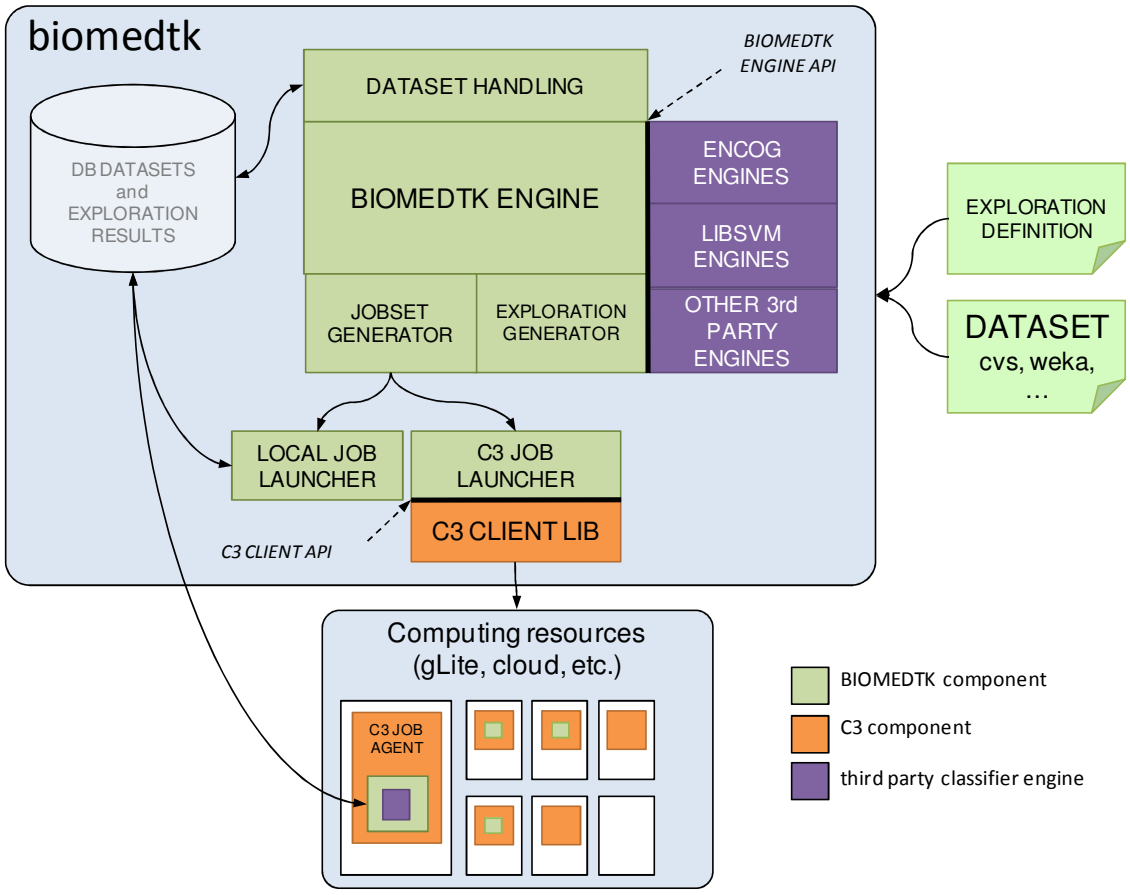


Figure 28: BiomedTK component archtecture

The *BiomedTK Engine* orchestrates all functionality invoking the *Exploration Generator* to process explorations and the *JobSet Generator* to effectively execute the explorations. Itself, the *JobSet Generator* handles jobs to a *Local Job Launcher* or to a *C3 Job Launcher* according to the user command. Jobs are sent to C3 by the *C3 Job Launcher* using the *C3 Client Lib* which is delivered by C3 simply as a jar file that is included within the BiomedTK distribution. For completeness and clarity, figure 28 includes a simplified representation of the C3 architecture which is further explained in Section 4.3. Datasets and exploration results are stored in a common database. Jobs, whether launched locally or over C3, feed the BiomedTK database with results. Any JDBC compliant database can be used and, for convenience, BiomedTK embeds the H2 database engine (H2 2011) which uses the local filesystem for storage, so there is no need to install another third party database if it is not desired. Initially, BiomedTK was initially conceived to send jobs directly to a gLite infrastructure and, hence, included a *gLite Job Launcher* and a set of command line functionalities to manage

jobs submitted to gLite. But the impact of accumulated job latencies in the classifier exploration cycle made us develop the C3 framework as mentioned before. Note how, through the *C3 Job Launcher*, BiomedTK jobs arrive directly at the Job Agent already living on a computing resource (such as gLite working node) without interacting with any middleware.

Finally, BiomedTK defines a *Classifier API* through which new classifier engines are integrated. Java classifier engines must deliver their implementation as a jar file containing the interface classes implementing the BiomedTK *Classifier API* and their own implementation classes (the actual engine). The following shows an excerpt of the BiomedTK *Classifier API* main interface that each classifier engine must implement:

```
public interface TrainedClassifier {
        public void initialize (ClassifierParameters p, Dataset d);
        public String getEncodedEngine();
        public void setEncodedEngine(String encodedEngine);
        public ClassifierParameters getParameters();
        public Double getTrainError();
        public void train(TrainListener l);
        public void ClassifierParameters[]
                            generateParameterSweep(ExploreProperties p);
        public void classify (List<DatasetElement> e);
}
```

It is a simple Java interface requiring mainly implementations of methods to train and classify datasets, to generate parameters sweeps from an exploration definition and create and restore a trained engine to/from a `String` based representation, which allows BiomedTK to store and reconstruct any third party trained engine. BiomedTK also supports native implementations of classifier engines. In this case, a classifier must deliver a jar file containing the interface classes, and a set of precompiled binaries for different hardware platforms. This way, whenever invoking a third party classifier BiomedTK searches for the platform specific binary and it will be able to use engines for as many hardware platform as precompiled binaries are provided. This is the case of the already integrated *libsvm* C library, for which binaries for MacOS, Windows XP/7, and different Ubuntu and Scientific Linux kernels were generated. This allows users, for instance, to start using BiomedTK to classify a given dataset on their local MacOS machine and then send larger explorations to a C3 Job Agent Colony living on top of a

Scientific Linux based gLite Grid infrastructure in a completely transparent manner. In total, BiomedTK is approximately made of 18000 lines of code and 160 Java classes.

4.2.4 Building ensemble classifiers

Ensemble methods in machine learning (Dietterich 2000) combine existing classifiers as an attempt to improve the performance of single classifiers either by compensating with each other in cases where a single classifier outperforms other, or by combining redundancy classification offered by several single classifiers.

For their simplicity and generality we are interested in Error Correcting Output Codes, a technique developed in the context of digital signal processing theory (Hocquenghen 1959; Bose and Ray-Chaudhuri 1960) providing redundancy in communications to compensate for noise and applied in machine learning since (Dietterich and Bakiri 1991). In short, for a given multiclass classification problem with n classes, an Error Correcting Output Code (ECOC) is defined by a matrix of size $m \times n$ describing, for each of the n possible final classes, how the m available binary classifiers are combined. This matrix is referred to as the *codematrix*, and figure 4 shows an example where 7 binary classifiers are used to produce a 6 class ensemble classifier. Each class is assigned a codeword that represents the participation of each binary classifier in that class. When an input vector is fed into each one of the 7 classifiers, a binary codeword is produced combining the output of all the binary classifiers. Then, a distance measure between codewords is defined and the class with the closest codeword to the one produced by the binary classifiers is selected to be the one assigned to the initial input vector. ECOCs are being used frequently in machine learning (Passerini, Pontil et al. 2004; Escalera, Tax et al. 2008; Huiqun, Stathopoulos et al. 2009) and also in biomedical contexts (Escalera, Pujol et al. 2008).

BiomedTK supports the ECOC based method for building ensemble classifiers. This includes (1) the definition of code matrices for a certain multiclass dataset, (2) the generation of binary datasets for each column in the code matrix and (3) assembling previously trained binary classifiers for each column into ensemble classifiers. Figure 29 represents an actual codematrix definition file as accepted by BiomedTK, where BREAST-TISSUE is a six-class dataset (from the UCI repository, *dataset with electrical impedance measurements of freshly excised tissue samples from the breast*) and the codematrix specifies seven binary classifiers (columns). The first six columns would be used to train classifiers distinguishing each class from the rest. The *norm*

column will be used to train classifiers distinguishing elements tagged as *car* (for carcinoma), *fad* (fibro-adenoma) or *mas* (mastopathy) from the rest (representing normal tissue tagged as *glandular*, *connective* or *adipose*).

```
ensemble.trainingset           = BREAST-TISSUE

ensemble.classifiers.validation = cvfolds 10
ensemble.classifiers.select     = bestPct any bestAUC 4035  bestPct bestPct any
ensemble.classifiers.names      =  car    fad    gla   mas    con    adi   norm
ensemble.codematrix.car         =   1      0      0     0      0      0      0
ensemble.codematrix.fad         =   0      1      0     0      0      0      0
ensemble.codematrix.gla         =   0      0      1     0      0      0      1
ensemble.codematrix.mas         =   0      0      0     1      0      0      0
ensemble.codematrix.con         =   0      0      0     0      1      0      1
ensemble.codematrix.adi         =   0      0      0     0      0      1      1
```

Figure 29: Example codematrix defintion file

The following summarizes the method supported by BiomedTK for building ensemble MLC:

1. Start with a multiclass dataset with n classes and a code matrix specification with n rows and m columns, such as the one in figure 4. Then, create for each column in the code matrix a new binary dataset according to the column specification (for column c, each element is labeled as "c" or "non-C" according to whether its ideal class in the original dataset is marked as 1 or 0 in column c). This generates m binary datasets.

2. Explore classifier configurations for each newly created binary dataset. In fact, figure 2 shows an exploration file to search binary classifier configurations for the *car* and *norm* derived binary datasets as created in the previous step starting from the code matrix shown in figure 29. This generates, for each column, a set of trained classifiers. BiomedTK supports this step through the exploration method explained in section 2.1.

3. Choose, one classifier for each column and ensemble them through the ECOC method. This is done through the *ensembles* command, which takes classifiers for each column, ensembles them and measures the performance on the ensemble classifier.

Observe that, for each column in a code matrix, one might have generated several binary classifiers in step 2 above; hence, the researcher needs to decide which specific classifier to use. BiomedTK supports this by interpreting the *ensemble.classifiers.select* line in the codematrix in which, for each column, the researcher specifies what criteria to follow to select binary classifiers. With *bestPct*, the binary classifier for that column

with best classification rate for test data is used. With *bestAUC* the one with best AUC measure for test data is chosen. A number (such as in the *mas* column) refers to the ID of a specific binary classifier present in the database that the researcher wants to use. With *any*, we instruct BiomedTK to try all binary classifiers available for that column.

4.3 The Cloud Computing Colonies framework (C3)

The Cloud Computing Colonies Framework (**C3**) is a Java light weight, open standards based set of utilities providing a fast job submission mechanism to distributed computing resources and a transparent, non-intrusive application deployment facility. It decouples user interaction from particularities of access methods of computing resources of different nature (different middleware, technologies, etc.). It is based on a *Job Agents* architecture such as in the DIRAC (DIRAC 2010) or AliEn (Bagnasco and et al. 2008) frameworks of different CERN experiments where, once job agents are deployed to the actual computing resources, all communications (job submission, status, results transfer, etc.) happen between the job agent and the C3 system, using specific resource provisioning mechanisms (such as gLite in case of certain Grid infrastructures) merely as a means to deploy Job Agents into the computing resources (worker node, host machine, etc.)

Once deployed, a *C3 Job Agent* receives incoming jobs directly from the C3 system, looks up in an application repository the application specified in the job description, executes it locally, sends regularly job status information to the C3 system (including process *stdout* and *stderr* as they are produced by the application) and, finally, sends the results back to the C3 system for publishing. Users and other applications interact with a C3 system to send jobs and retrieve their status, *stdout*, *stderr* and final results. A command line utility is offered to users exposing this functionality, and a simple client API, distributed as a jar file, is available for applications to programmatically manage this interaction. This is the case of BiomedTK as described in previous section.

C3 introduces the notion of a *Job Agent colony*, representing a set of job agents deployed on a specific computing infrastructure, such as on the worker nodes of a specific gLite (gLite 2011) computing element within the EGI federation (EGI 2011). A *colony maintainer* is a C3 module that tries to ensure a certain job agent population over a specific computing infrastructure. C3 includes colony maintainers for computing resources available through gLite middleware, Amazon EC2 and regular SSH access. New colony types to harness computing resources available through other technologies

and models can be easily integrated by developing new *colony maintainers* to deploy job agents colonies into their computing resources. Colony maintainers must follow the *C3 colony API* and encapsulate all specifics of accessing a particular computing resource. All job agents, regardless the colony to which they belong or the colony maintainer that ensures their existence, are readily available in a homogeneous manner to all users of the C3 system that manages them.

4.3.1 C3 architecture

C3 is mainly based on the Java Message Service (JMS 2011) an industry standard general purpose message delivery mechanism for distributed applications. As in other standards, JMS establishes the concepts and implementations that any JMS provider must offer, and clients following JMS are not tied to a specific provider. A JMS Provider is therefore, a software package implementing the JMS specification and a JMS Client is any application or process producing and/or receiving messages through a certain JMS Provider. Messages can be interchanged through Queues or Topics, which are managed by the JMS Provider. Messages sent to a Queue are guaranteed to arrive exactly to one JMS Client, and will be stored in a staging area until some JMS Client asks to receive a message from the Queue. If several clients decide so, the JMS provider will leverage the queue of messages among the existing JMS Clients as they become available, delivering each message once to only one client. Topics follow a publish/subscribe model where messages sent to a topic are delivered to all clients registered to that topic and, if no client is subscribed when it is published, the message usually goes ashtray.

JMS Clients may choose to receive only certain messages from queues or topics according to custom message properties or other criteria. JMS Providers implement all logic necessary to guarantee message delivery, persistence, security, etc. as required by the JMS specification. C3 is therefore engineered as a set of JMS Clients interchanging messages through Queues and Topics configured in a JMS Provider. C3 currently uses the Open Message Queue (MQ 2011), an open source implementation of JMS. Figure 30 shows the C3 architecture, organized around a set of Queues and Topics.

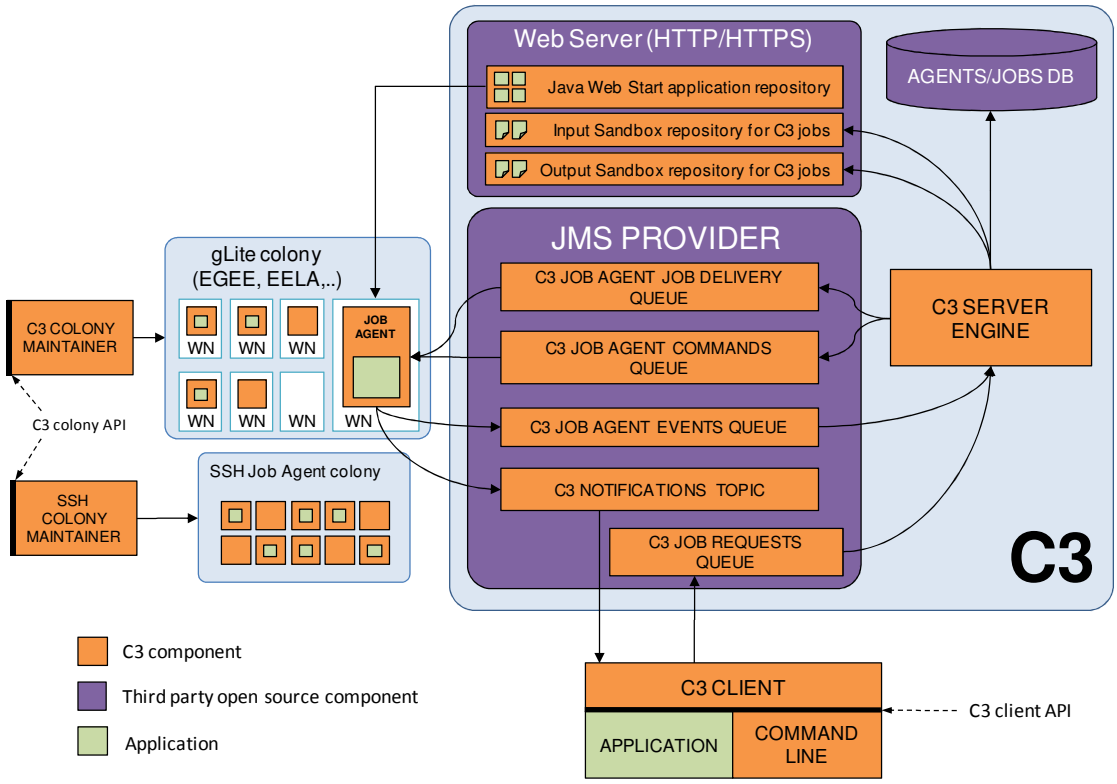


Figure 30: C3 component architecture

Three C3 components act as JMS Clients: the *C3 Client*, which submits jobs and receives jobs notifications (status, *stdout*, *stderr*), the *C3 Job Agent*, which receives jobs to be executed in the computing resource where they live and commands (such as 'shutdown', or 'cancel job') and the *C3 Server Engine*, which is the core of the C3 system. The *C3 Server Engine* regulates traffic of jobs, status messages and commands among the different actors. It is a Java process that must be kept running along with the JMS provider (Open Message Queue in our case). These components interact through the following queues and topics:

C3 Job Requests Queue: Clients use this queue to send three kinds of messages: *jobs submission*, *job status query* and *job cancelation*. Only C3 clients send messages to this queue. The only consumer for this queue is the *C3 Server Engine*.

C3 Job Agent Job Delivery Queue: When the *C3 Server Engine* receives a *job submission message* through the previous queue, checks its content, updates the jobs database, publishes its input sandbox in the web server and dispatches it to the *C3 Job Agent Job Delivery Queue*. All job agents are potential consumers for all messages to this Queue and they register to receive new jobs (encapsulated within a JMS message) only when they are finished running a previous job. A job agent may only execute one job at a time and if several job agents are free, the JMS Provider will distribute evenly

job submission messages across them. The JMS specification guarantees that each job submission message is either held in the queue if no job agents are available, or delivered to exactly one job agent. This logic is implemented by the JMS Provider. C3 components just limit their activity to receive and send messages to queues.

C3 Job Agent Commands Queue: This queue is used by the *C3 Server Engine* to deliver commands to specific job agents. Although all active job agents are consumers to this queue, messages are addressed to a single job agent using standard message filtering as defined in JMS. This way, the *C3 Server Engine* can instruct a specific job agent to cancel the job it's running after receiving a job cancellation request sent by a *C3* client to the *C3 Job Requests Queue*. It works analogously for job status requests, or agent shutdown commands.

C3 Job Agent Events Queue: This queue is used by job agents to send regularly information about their status (start up, shutdown, job processing progress, etc.) to the *C3 Server Engine* who, in turn, usually updates the database with the received information. The database is later used to know what agent is processing what job, how many agents are available, etc.

C3 Notifications Topic: This is a topic, so JMS simply delivers messages to the active clients that have subscribed to this topic if there is any, otherwise the message goes ashtray. Typically, a C3 client subscribes to this topic to receive regular messages about job or agent status. These messages include job status and *stdout* and *stderr* chunks as they are produced by the job process running, information about the machine in which the job agent is running, etc. This allows any C3 client to easily monitor job status and progress.

In addition, C3 uses an input and output sandbox model similarly to Grid middleware, where users have the possibility to determine a set of files to travel with the job submission (input sandbox) and a set of files which persist after job execution (output sandbox). C3 provides a simple implementation of this model, where files in the input and output sandboxes are published through a web server by the *C3 Server Engine* when jobs are submitted and finished. Afterwards, users can employ standard HTTP/HTTPS tools to access them (browsers, command line fetching, etc.)

C3 makes intensive use of Java Web Start (JWS 2011) as a transparent application deployment mechanism avoiding custom application installation on computing resources and not requiring users to include their applications as part of their input sandbox. With JWS, an application and its dependant libraries are made available at a

certain web server where an application description file contains information on application versioning, libraries and dependencies. Application descriptions are given in JNLP files (for *Java Network Launching Protocol*). When using JWS, an application is launched by providing a URL to its application descriptor and the underlying JWS implementation reads it, fetches its library files, checks versions, etc. and finally launches it. *C3 Job Agents* use JWS to launch applications and C3 clients, when submitting jobs, specify what application to launch by simply providing the URL to the application descriptor. Applications can then be published into an application repository through any web server, including the one already bundled with C3. Two JWS features are of key benefit for C3. First, JWS is included as part of any regular Java installation (JDK, the Java Development Kit, or JRE, the Java Runtime Environment) and a JWS client library is part of the C3 Job Agent package, so it imposes no additional installation requirements on computing resources hosting job agents. Second, it provides versioning and caching of applications so that they are only fetched from the repository if they do not exist in the local job agent cache (managed by JWS) or if a new version has been published.

As in the case of BiomedTK, C3 works with any JDBC database and also with any Web server as long as there is a shared file system through which input and output sandboxes can be published. For convenience, the H2 database server (H2 2011) and the Jetty web server (Jetty 2011) have been embedded within C3 so that they can be started together with the *C3 Server Engine* process, avoiding, if desired, the need to install additional database or web server software.

As mentioned, C3 client functionality is exposed through an API so that other applications can programmatically interact with the C3 system. C3 also provides a command line tool (which just uses C3 Client API, as BiomedTK or any other application may do) for users to directly submit jobs and query the status of the jobs and job agents. The C3 Client API is delivered as a set of utility classes through a regular jar file with a very small footprint (128KB) which together with the required JMS client libraries makes a total footprint of about 900KB. The following shows the main utility class that any C3 client application can use:

```
public class JobActions {
    public static Long submitJob (C3JobProperties props);
    public static String cancelJob (Long jobid);
    public static JobBean getJobInfo (Long jobid);
    static String getLastStdouterr (Long jobid);
}
```

It contains straight forward methods for job submission and management. Note the **getLastStdouterr** method allowing client applications to retrieve the last reported *stdout/stderr* chunk from the jobs. It is actually the job agents the ones in charge of capturing *stdout/stderr* and other job status conditions and sends them to the *C3 Job Agent Events Queue* and the *C3 Notifications Topic.*

4.3.2 C3 jobs

From a client application perspective (such as BiomedTK), a C3 job is a simple object containing the following properties (*C3JobProperties* class of the C3 Client API):

```
public class C3JobProperties extends BasicC3Properties {
    public String   jnlpurl;
    public String[] inputfiles;
    public String[] outputfiles;
    public String   command;
    public String[] arguments;
}
```

which basically amounts to defining two lists of files that make the input and output sandboxes and then either (1) a command and a set of arguments or (2) the address of an application in a Java Web Start repository (the *jnlp* property). When a job arrives to a job agent, it first checks if the *jnlp* property is set and uses the standard Java runtime utilities to fetch and launch locally the application through Java Web Start. Otherwise it executes whatever command and arguments are present in the job description. Applications using the *C3 Client API*, create *C3JobProperty* instances to submit jobs.

From a user perspective, the C3 Command Line interface (invoked through the **c3c** command line shell), being itself just another C3 client, retrieves the job properties from a job definition file so that users can later submit the job and query their status. Figure 31 shows a sample job file and user session, with the commands issued by the user to submit the job and query its status. Note how the information on what job agent the job was run is kept and shown. Of course, this data is also available programmatically to applications invoking the *C3 Client API.* In this example, the job was executed in job agent number 21, belonging to a gLite colony of the EELA-CETA

Grid federation, and the *jnlpurl* property of the job file was used to specify the address of the application to fetch and run by JWS when the job arrives to a job agent.

file **sample.job**

```
c3job.jnlpurl:      http://ui-eela.ceta-ciemat.es/apps/HelloWorld.jnlp
c3job.inputfiles:   dummy-input-1.txt:dummy-input-2.txt
c3job.outputfiles:  dummy-output.txt
c3job.command:      java
c3job.arguments:    "hello from a sample job"
```

User session

```
rlx@rlx-desktop:~/c3/as-client$ c3c submit sample.job
[INFO] [2010.08.05 19:00.04 ]  submitting job from file sample.job
[INFO] [2010.08.05 19:00.05 ]  job submitted successfully. jobid =14


rlx@rlx-desktop:~/c3/as-client$ c3c info 14
--------------- JOB INFORMATION ----------------------------------
job id              14
job status          FINISHED
submit date         2010.08.05 19:00.04
scheduled date      2010.08.05 19:00.04
running date        2010.08.05 19:00.05
finished date       2010.08.05 19:00.37
exit code           0
--------------- AGENT INFORMATION --------------------------------
agent id            21
agent name          c1b345
agent hostname      ba-01-14.ceta-ciemat.es
agent ip address    192.168.30.21
agent colony        EELA-CETA
agent colony type   glite
agent os name       Linux
agent os arch       Linux
agent os version    2.6.9-67.EL.cernsmp
agent java version  1.6.0_16
agent max wait      50000
---------------------- LAST STDOUT CHUNK --------------------------------
[14.out.00001] ba-01-14 hello from a sample job
-------------------------------------------------------------------------
```

Figure 31: Sample C3 job file and client session

C3 Jobs go through a state workflow which typically takes a job from *submitted* (timestamp of the user localhost when submitted), to *scheduled* (when it enters the *C3 Job Agent Job Delivery Queue*), to *running* (when the JMS job message is delivered by the JMS provider to a job agent and the job is started), to *finished* (when its output sandbox is finally published in the web server). Alternatively, jobs may fall into *canceled* (if the user requested its cancellation), *aborted* (if the job agent running it stopped or was instructed to shutdown before it finished) or *failed* (if the job process finished with a non-zero exit code). The sample application in figure 31 waits for 30 seconds and then outputs a "hello" message (see *Last Stdout Chunk* section in the job info output). Note the little latency between *submit* and *running* (1 second), that includes (1) zipping the input sandbox (just a few kilobytes in this example), (2) sending a JMS Message to the *C3 Job Requests Queue*, (3) having the *C3 Server*

Engine receiving the message, verifying its contents, publishing its input sandbox in the web server, updating the C3 database and relying it to the *C3 Job Agent Job Delivery Queue*, (4) relying on the JMS Provider delivers the job message to some job agent and (5) having the job agent effectively forking and running a new process. Note as well the little latency to process its output in a similar fashion (32 seconds between *running* and *finished* states, which includes the 30 seconds of application effective running).

4.3.3 C3 job agents and colony maintenance

The mission of colony maintainer modules is to ensure that populations of job agents are available on specific computing resources. Usually, colony maintainers stay running monitoring the population of job agents available, redeploying agents if they die to ensure, if resources are available, the population reaches a certain number of job agents. C3 includes three colony maintainers, one to deploy job agent populations to gLite infrastructure, encapsulated within virtual machines for Amazon EC2 and, finally, another one to deploy job agents to remote machines through remote SSH commands execution. The *gLite C3 Colony Maintainer* uses the standard gLite job submission mechanisms (through issuing *glite-wms-job-submit* and *glite-wms-job-status* commands and using JDL files generated on the fly) to deploy C3 job agents on gLite worker nodes. A job agent is encapsulated within a JDL file, including the C3 libraries within the input sandbox and then submitted to gLite. The *gLite Colony Maintainer* is started on a gLite user interface and uses the credentials (proxy) of the user launching the maintainer process.

The SSH Colony Maintainer issues ssh commands to remote machines sharing some file system to deploy the job agents. In both cases, the only pre-requisite on the final resources (gLite worker nodes or remote SSH available machines) is that the Java Runtime Environment v1.5 or above is installed on the remote machines.

In the case of Amazon EC2 it was only required to encapsulate the C3 job agents within a standard Linux virtual machine on their Cloud. This was straight forward, starting off from a basic virtual machine template (in which java is already installed) requiring simply the installation of the C3 toolkit and setting it up so that it is the C3 job agent is launched when the machine starts, and that the machine shuts down when the C3 agents finishes or dies (so that the instance is removed and billing by Amazon stopped). No specific colony maintainer was required since the AWS Management

Console (see Figure 34 on page 108) provides all functionality to launch, manage and monitor virtual machines.

If a certain eInfrastructure does not provide mechanisms guaranteeing the survival of the colony, C3 includes a utility for colony monitoring and simple management through which it can easily integrate new colony maintainers which must implement the C3 Colony API. The following shows the main interface of this API:

```
public interface C3Colony {
    public void init        (C3ColonyProperties colonyProperties);
    public void addAgents   (int numberOfAgents);
    public void removeAgent (C3ColonyAgent agent);
    public List<C3ColonyAgent> getColonyAgents();
}
```

which requires new colony maintainers to implement straight forward methods to add, remove and find out what agents are available. For instance, in the case of the gLite Colony Maintainer, its implementation of the *addAgents* method that creates the JDL file on the fly, includes the C3 libraries and submits the actual gLite job.

C3 job agents auto shutdown if they have not received jobs for a certain amount of time configurable by the administration that launches the colony. This is so to avoid C3 take over resources that may not be used if there are no users submitting C3 jobs. When a colony maintainer is instructed to ensure a colony population of, say, 20 job agents, it will constantly monitor the population and, as job agents die, it will redeploy them. Job agents may die for many reasons, among them, because they auto shutdown due to lack of activity. This gives a chance for other users or applications to access the same computing resource. Anyhow, if a job agent is kept constantly busy (because of long jobs, or because of many jobs) it will not auto shutdown.

Figure 32 shows a sample deployment scenario of using C3 over a single gLite federation where (1) the *C3 Server Engine* runs together with the Open Message Queue JMS provider in the same server machine, (2) a *C3 gLite Colony Maintainer* runs within a gLite User Interface, which is required so that it can effectively submit and check the gLite jobs containing the job agents and (3) the user sends jobs to C3 from his desktop.

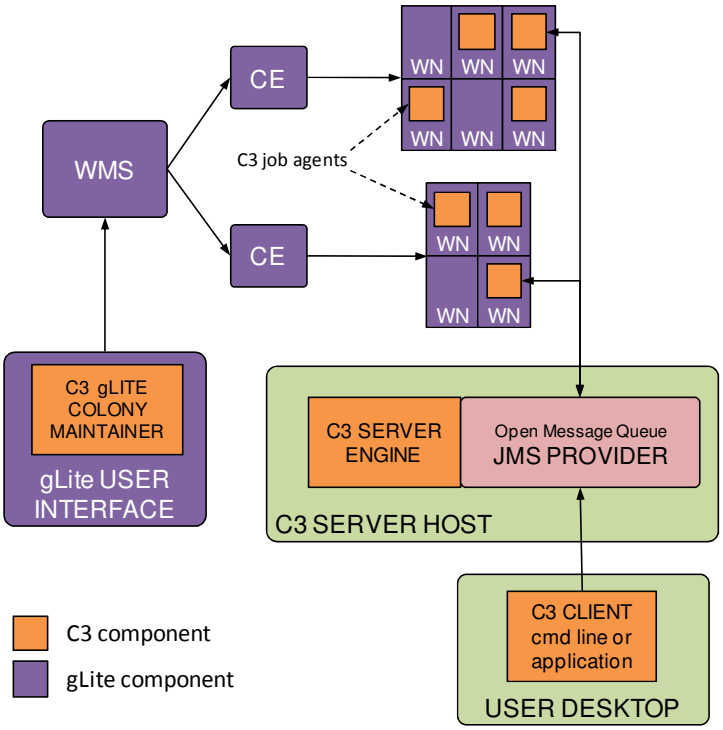


Figure 32: C3 sample deployment scenario

Note that gLite jobs (containing C3 job agents) are sent under a certain user credentials and all C3 jobs will be run on the worker nodes under those user credentials. Also, the *C3 Server Engine* and JMS providers do not need to be on the same private network as the worker nodes, as long as the right connectivity between them is provided (as explained below). Even the *C3 Server Engine* and the JMS providers may be in different physical machines, since the *C3 Server Engine* is simply one additional JMS Client. This property allows many deployment and scalability options.

4.3.4 Other issues

Communication: C3 components (C3 Server Engine, C3 Clients and C3 Job Agents) are JMS Clients so they need connectivity to the JMS Provider ports for sending and receiving messages. In JMS, it is always the JMS Client who initiates any connection to any JMS Provider, therefore all C3 components require outgoing connectivity to reach the JMS Provider, but not incoming connectivity. This is quite convenient as in most occasions, job agents are deployed in computing resources not visible from the internet (with local IP addresses) but with outgoing connectivity (through NAT or other mechanism). The Open Message Queue JMS Provider currently used by C3 listens for

JMS clients on port 7676 by default and this is configurable. In case ports cannot be reached by clients, it also offers a servlet based HTTP bridge that can be deployed into any web container and thus made available through port 80 which is regularly allowed for outgoing connections. This HTTP bridge is included and configured within the Jetty web server embedded in C3 and therefore ready to use. The HTTP bridge however introduces latencies in message delivery that could reach a few seconds.

Scalability: High utilization systems may generate many messages (for job submission, job status events, commands, etc.). Two main bottlenecks may arise within a C3 System when too many messages are produced: (1) the JMS Provider, Open Message Queue in our case, cannot timely deliver the messages and (2) the *C3 Server Engine* cannot timely process the messages as explained in Section 4.3.1. The first case must be solved at the JMS Provider level, and most JMS Providers today, including Open Message Queue, are inherently scalable, offering some mechanism through which new hardware and processes can be added to share the message processing and delivery load. For the second case, it must be observed that the *C3 Server Engine* is a stateless service, processing independently each message from others. New instances of the *C3 Server Engine* can be added with no reconfiguration required, consuming messages from the same queues and standard JMS Provider message delivery mechanisms will distribute the message processing load among the available *C3 Server Engine* instances present, which might also be distributed into different machines. This includes the possibility to hot-start new instances as the load increases, not because a C3 design quality, but profiting from the application model devised in the JMS specification.

Security: C3 adheres to the JAAS (JAAS 2011) security standard (Java Authentication and Authorization Service) and while, in a first stage, C3 provides a simplistic security model, more elaborate authentication and authorization mechanisms can be plugged in following JAAS. This would allow to control, for instance, that jobs are only sent to colonies to which the user has permission, to delegate job acceptance to each colony maintainer, or retain a centralized authorization control, etc.

Other job delivery mechanisms: Currently the *C3 Server Engine*, takes all job submission requests and sends them to any agent available through the *C3 Job Agent Job Delivery queue*. As needs arise, more complicated delivery mechanisms can easily be incorporated within the *C3 Server Engine* logic by using JMS filtering facilities, just like C3 uses them now to send commands (such a job cancel command) to a specific job agent. For instance, one might want to specify, as part of the C3 job properties,

certain criteria to limit the range of job agents that might accept a specific job, such as memory or operating system requirements.

4.4 Experimentation and Validation

A set of experiments was setup in order to show the contribution of BiomedTK/C3 and the exploration method it supports to researchers' efficiency and experimental reach (Ramos-Pollan, Guevara-López et al. 2011). The aim was to demonstrate how (1) complex experiments consisting of several datasets, engines and classifier configuration can be managed with relative simplicity and (2) the computing resources required to execute the proposed experiments (about 16000 CPU hours) could be utilized in an efficient manner. In these experiments, BiomedTK/C3 is to be used to find classifiers for selected datasets from the UCI machine learning repository (Frank and Asuncion 2010) by exploiting distributed computing resources, and aiming at reaching, in reasonable time, accuracy levels comparable to the ones reported in different literature sources for the given datasets. By being able to obtain efficiently these results, researchers are then positioned to pursue research in a timely manner using the methods herewith described.

4.4.1 Goals and metrics

The main design objective of the experiments described below was to set up a complex exploration task involving different datasets, engines and classifier configurations through several exploration cycles, as described in page 87, requiring a large number of computing resources. Showing the utility of BiomedTK/C3 amounts to (1) demonstrating the agility with which computing resources can be used to perform the experiments and (2) reporting acceptable classification results on the selected datasets with the classifier configurations evaluated by BiomedTK/C3.

A notion on the degree of the *agility* achieved is conveyed by describing in next section the experimentation process through the number of configurations managed, the number of local and C3 exploration cycles performed and the CPU hours consumed. In particular, in order to gain understanding on what classifier configurations to evaluate, it is important to observe how for each dataset different exploration cycles had to be performed both locally and over the computer resources made available by C3.

Then, *classification results* obtained for each dataset were compared with those reported on different literature sources that included the following references: (Wilson and Martinez 1997; Estrela da Silva, Marques de Sá et al. 2000; Vlachos, Domeniconi et al. 2002; Li and Wong 2003; Sebban, Nock et al. 2003; Soares 2003; Domeniconi and Yan 2004; Esmeir and Markovitch 2004; Fung, Dundar et al. 2004; Jiang and Zhou 2004; Kotsiantis, Zaharakis et al. 2006; Elter, Schulz-Wendtland et al. 2007; Lorena, de Carvalho et al. 2008; Urbanowicz and Moore 2009) The aim was not to make an exhaustive literature review, but to sample referenced works and reach comparable results in reasonable time.

4.4.2 Experimental setup

The UCI datasets shown in table 24 on page 154 were selected for experimentation. For each dataset, increasingly larger explorations were devised including all engines supported by BiomedTK (see table 14 on page 83) according to the following exploration cycle:

1. Reformat original UCI data file, import and normalize it
2. Create exploration file and perform a few local classifier explorations.
3. Analyze results and refine exploration file for large explorations
4. Launch large explorations to C3.
5. Analyze and gather results.

Datasets were imported from a CSV formatted file, after some reformatting from the datasets delivered by the UCI repository, and basic data normalization that was performed by BiomedTK before the explorations. This normalization consisted in mapping each input feature to the [0,1] interval so that for each element of a dataset the value v_i of feature i was normalized to $v_i' = \frac{v_i}{Max_i - Min_i}$ where Min_i and Max_i are the minimum and maximum values of feature i in all elements of the dataset

Explorations for each dataset included many classifier configurations (see table 17 below and table 14 on page 83), each configuration using different classifier parameters, (ANN layers, neurons per layer, SVM kernel type, etc.) Each exploration was tuned for each dataset to account for their different characteristics. For instance, the input layer of any ANN for a multilayer perceptron must have as many neurons as dataset

features, datasets harder to classify might require more exploration cycles, larger datasets require more training time so explorations cannot be as large, etc.

In addition, experimental conditions change between authors in validation methods used, how results are summarized, etc. so comparisons must be carefully interpreted. To be as coherent as possible with the different setups, 10-fold cross validation was used. Furthermore, in some works it is not clear what specific classifier configuration or validation method was used, which somehow also constitutes a challenge in the exploration endeavor undertaken.

A total of 200 C3 job agents were deployed on different computing resources as shown in figure 33. Although most job agents were deployed on a gLite Grid infrastructure a small C3 Amazon EC2 colony and C3 remote SSH colony were setup to demonstrate how, regardless where they were physically deployed, all jobs agents become available to BiomedTK/C3 in a homogeneous manner and jobs, executing classifier explorations as they were handled to free job agents indistinctly.

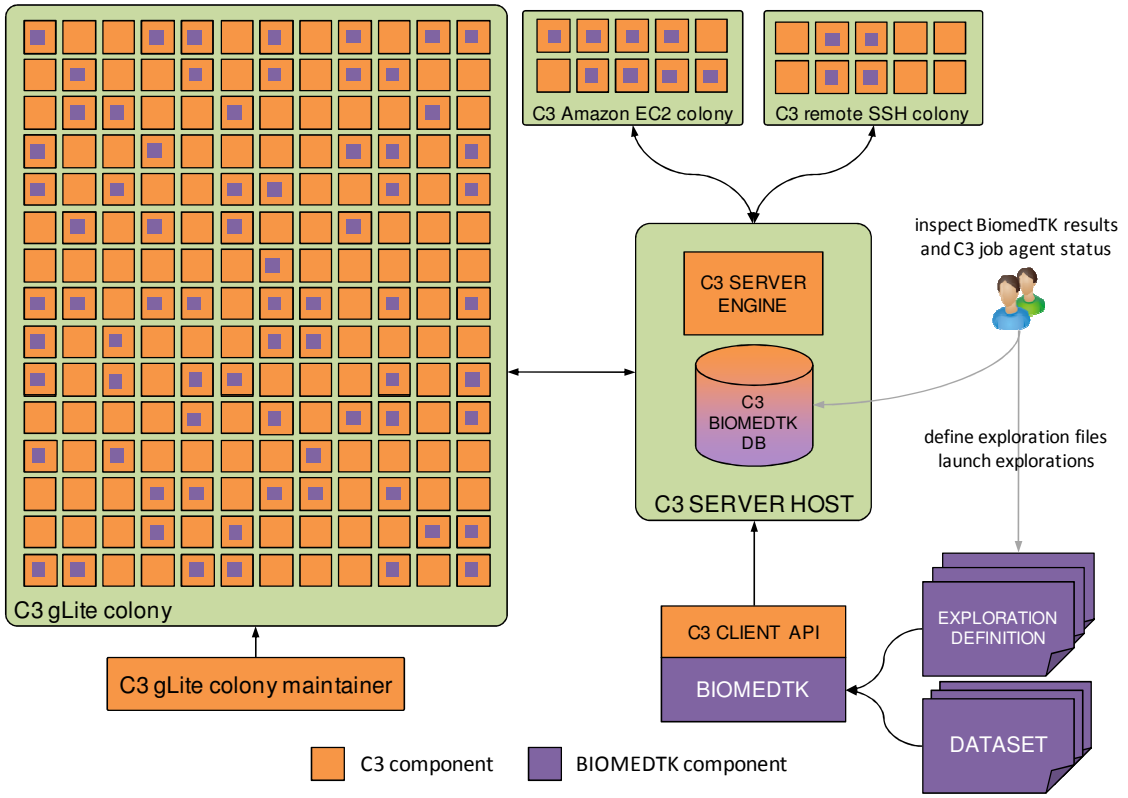


Figure 33: C3 job agents deployment for experimentation

C3 job agents were distributed in the following way: 180 C3 job agents on a gLite Grid site and managed by the the *C3 gLite Colony Maintainer*; 10 C3 job agents were deployed on office computers through the *SSH C3 Colony Maintainer* and 10 job

agents were also deployed for 10 hours on the Amazon EC2 service (as shown on the screenshot of the Amazon EC2 management console in figure 34).

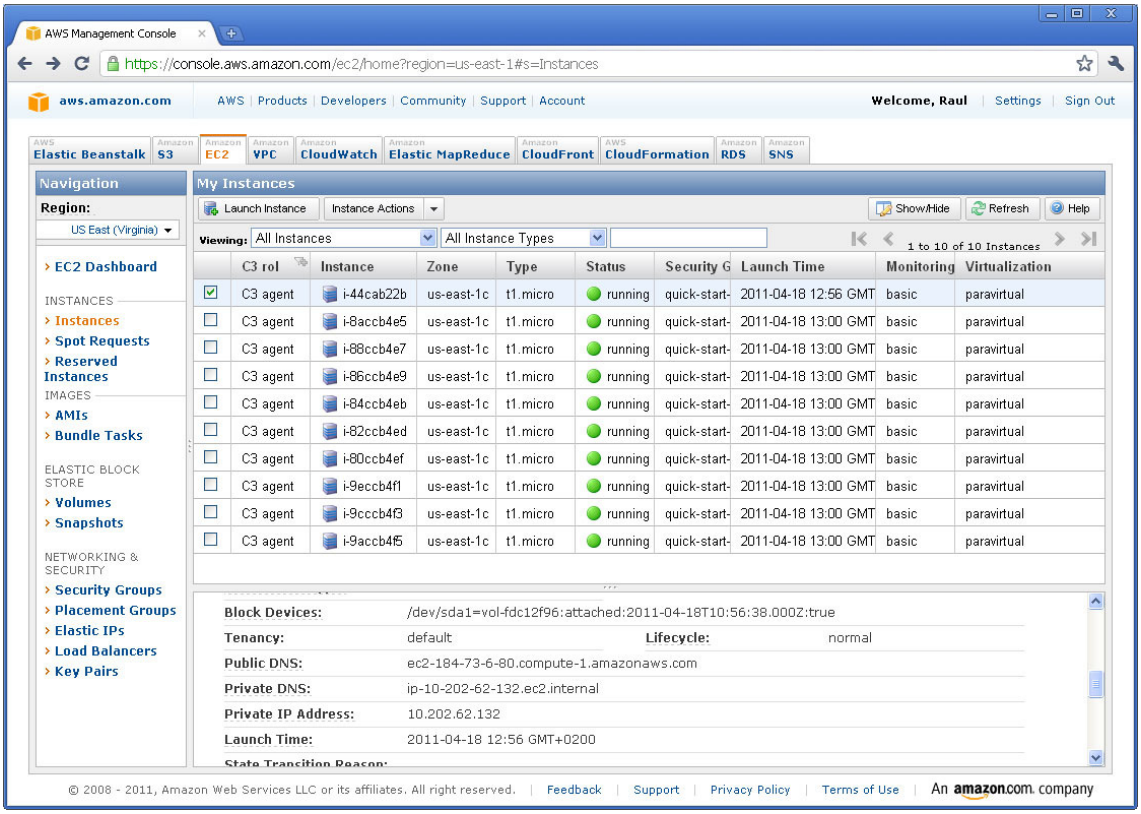


Figure 34: Amazon EC2 Management Console with 10 C3 job agents running

Datasets were explored in the order described in the results tables (table 17 and table 29) and, as large explorations for a given dataset were sent to C3, work started on the following dataset (importing it, performing local explorations, enlarging them and sending them also to C3.). This produced a great overlap on what specific datasets and classifier configurations were being evaluated on the C3 Job Agents at any given moment, as it is desirable to use efficiently the computing resources.

4.4.3 Results and discussion

A total of 16519 CPU hours were consumed to train 8842 different classifier configurations over 15 datasets. For each dataset, experiments were set up by creating a single exploration configuration file containing classifier configurations for all engines. Then, this file was refined at each the exploration cycle adding and removing configuration parameters. Table 17 shows, for each dataset, the number of configurations trained, the number of local and C3 explorations, the total CPU hours

consumed by the C3 explorations, and the actual physical time taken by them over the available job agents. The whole experiment took less than a week to be carried out, including all configuration and data analysis tasks.

Table 17: Summary of configurations and computer resources in explorations

	bcw	**bcwd**	**btcat**	**echocard**	**haber**	**heartsl**	**hepat**	**liver**
configs	280	228	856	872	942	568	376	736
local explorations	2	2	1	3	3	1	3	3
C3 explorations	1	2	1	2	1	1	1	2
C3 CPU hours	695.38	1215.67	714.87	423.65	198.89	1072.72	644.16	321.06
C3 run time (hrs)	5.07	13.46	6.34	3.44	2.16	12.14	7.07	4.04
	mmass	**park**	**pgene**	**pimadiab**	**spam**	**spectf**	**tictac**	**TOTAL**
configs	464	736	612	676	176	616	704	**8842**
local explorations	3	3	2	3	1	4	2	**36**
C3 explorations	1	2	2	2	1	1	1	**21**
C3 CPU hours	1167.61	1679.26	1778.35	1644.43	2396.98	1115.97	1450.55	**16519**
C3 run time (hrs)	10.44	21.00	26.68	15.46	46.91	17.28	11.30	**202**

Table 29 on page 159 shows further detail per dataset and engine: (1) how many configurations were trained, (2) how many CPU hours took to train them, (3) the best percentage of elements correctly classified on the test part of the dataset (*accuracy*), and (4) the best AUC obtained on the test part of the dataset. Finally, the bottom lines in table 29 show the best results obtained overall in our exploration (accuracy and AUC) and those found in our literature review (accuracy in all datasets, except the *mmass* dataset, where reference (Elter, Schulz-Wendtland et al. 2007) gave their results in AUC). Figure 35 shows the two ROC curves generated by BiomedTK for one classifier configuration over the *bcw* dataset. The curve on the left corresponds to the Wilcoxon-Mann-Whitney statistic and the curve on the right is its smoothed version using the bi-normal distribution method from JLABROC (Hanley 1996).

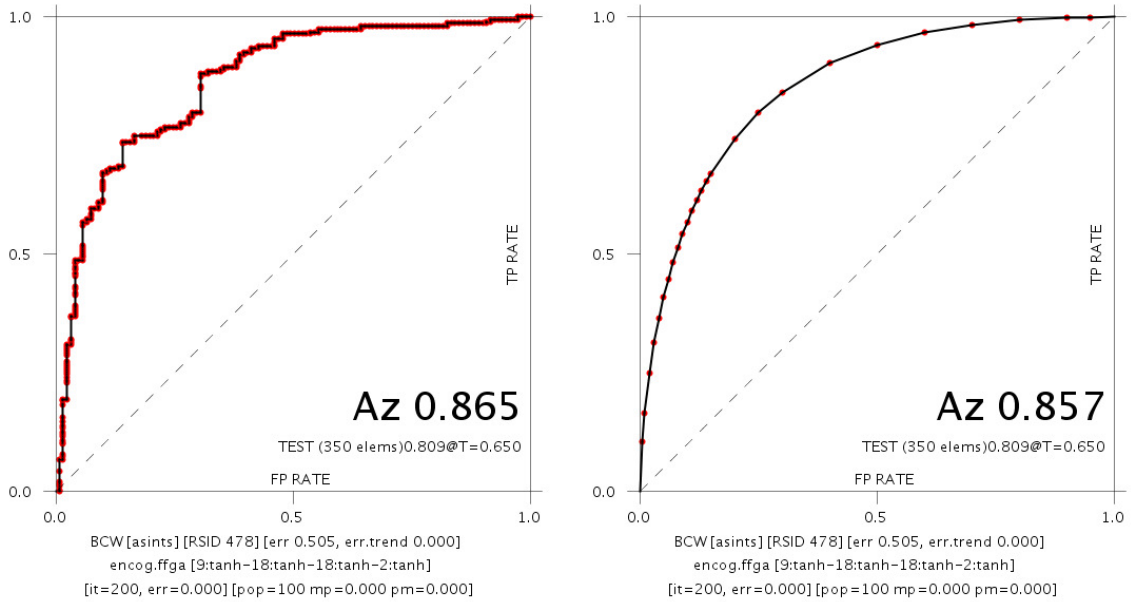


Figure 35: Empirical and smoothed ROC curves for a *bcw* dataset classifier

Note how for some datasets larger explorations were made than for others, as more exploration refinement cycles were required until satisfactory results were found. A key factor was acquiring a notion on the requirements of computing time for each classifier and dataset so that explorations can be adjusted to the available computing resources. Observe in this sense how datasets with larger number of elements or input features take longer time to train with ANN engines and, were not for the possibility to harness distributed computing resources, exploring even a few classifier configurations for them would simply be impossible. Without an appropriate supporting tool, following the method herewith proposed requiring such large explorations, would imply providing the logistics to manage different output formats for the different underlying engines, organize configuration files for each resource provider (such as for gLite jobs), etc. In addition, in the case of gLite resources, it would require monitoring the execution of the gLite jobs, keeping accounting of failed ones and resubmitting them, gather different outputs for each job and consolidate them in a single file or database, etc. It is the reduction of the effort cost of taking care of all these logistics that makes it possible to efficiently harness computing resources for machine learning in a systematic way.

In practical terms, for each dataset a single exploration configuration file was defined and maintained which is the key to be able to manage large amount of CPU hours for such a diverse exploration with reasonable effort. In addition, the whole process is easily reproducible rendering classifier exploration requiring large computing resources a systematic task. This allows researchers to focus on their core research,

devising exploration strategies, evaluating dataset preprocessing algorithms, new classification methods, etc. instead of sorting out the logistics of access methods to computing resources, preparing datasets differently for each third party engine, etc. This is what enabled the thorough validation of the AUC optimization method described in Section 3.3.

4.5 Conclusion

This Chapter described the major technological contributions of this thesis (see Section 1.4) based on two software frameworks, BiomedTK/C3, and their associated classifier exploration methods that enable systematic and agile exploitation of available distributed computing resources of different nature for massively exploring and evaluating configurations of machine learning classifiers. Experimentation showed their utility with commonly used datasets and experimental conditions, reaching in reasonable time classifier performance comparable to that reported in a variety of literature sources. Their development was motivated by the IMED project within which this thesis originated, and its need to efficiently use available computing resources to find well performing classifiers for breast cancer CAD in a timely manner as datasets are being produced by specialized radiologists annotating and classifying mammograms. These classifiers would become targets upon which CAD systems can be built.

However, as shown by experimentation, their reach lies far beyond this application domain and, together with the method to enable AUC optimization in existing machine learning classifiers described in Chapter 3, they can be used in biomedical data analysis tasks but also in other domains where machine learning classifiers can be applied.

Therefore, at this stage the material means are ready to start field work applying the artifacts obtained so far into the IMED project and this is described in next Chapter. Besides the specific contribution to the project, next Chapter should also be viewed as an example on how the results of this thesis can be applied on a real world problem.

Chapter 5

Application in Breast Cancer CAD

5.1 Introduction

The IMED project constitutes the context within which this thesis has been developed contributing to achieve part of the project objectives as described in Section 1.2. The first Portuguese breast cancer database has been built in the course of the project, with anonymous cases from medical historical archives supplied by FMUP-HSJ (*Hospital de São João–Faculty of Medicine* at University of Porto, Portugal) complying with current privacy regulations as they are also used to teach regular and postgraduate medical students. The database is referred to as the "Breast Cancer Digital Repository" (BCDR) in this thesis. BCDR is supported and hosted on the Digital Repositories Infrastructure (DRI) platform developed by the Center of Extremadura for Advanced Technologies (CETA-CIEMAT) in Spain.

As part of the project work plan, specialized radiologists segmented and diagnosed images from the BCDR as explained in Section 1.1.4 using software tools and graphical workstations developed by the project. The datasets resulting from this process constitute the input for this thesis, with the goal of applying the developments described in previous chapters to obtain well performing classifiers that can be integrated back in the medical graphical workstations for assisted diagnosis.

From the IMED project perspective, the machine learning classifiers described in this section and discovered through the artifacts produced by this thesis constitute its contribution (output) to the project (Ramos-Pollan, Guevara-López et al. 2011). This includes the multilayer perceptrons modified for AUC optimization described in Chapter 3. From this thesis perspective, the IMED project provides a real world case to demonstrate the applicability of its results (Ramos-Pollan, Rubio del Solar et al. 2009; Ramos-Pollan, Franco et al. 2010; Ramos-Pollan, Franco et al. 2010; Ramos-Pollan, Rubio del Solar et al. 2010).

5.2 The Breast Cancer Digital Repository (BCDR)

Usage of DRI to create and host the BCDR repository can be revisited in the references just mentioned. DRI simplifies hosting of digital repositories (such as for medical imaging) over distributed storage resources across different locations (hospitals, university, computer centers, etc.). Its architecture offers a simple network API (*Application Programming Interface*) facilitating development of client applications, like specialized graphical image processing interfaces such as the one shown in figure 9 (page 20), but also batch interfaces to proprietary picture archiving and communication systems (PACs), custom information sources, etc. (figure 36). Repositories are described in regular XML files, allowing agile implementation of evolving data models, such as it often happens in medical environments. In addition, reconfiguration and redistribution of repository content over different physical storage services (local storage, Grid, web based, databases, etc.) can be done transparently to the final user. With this, the project was able to start at FMUP-HSJ with an *embedded* configuration, where all components are bundled within the same physical machine into a medical doctor´s desktop. This allowed specialized radiologist segmenting and classifying mammograms straight away and, thus, enabled the generation of datasets for training machine learning classifiers for CAD shown in this section reasonably early in the project. Meanwhile, its evolution towards real distributed configurations is transparently ensured allowing us to evaluate how to best integrate it within medical doctors' workflows and institutional computing environments with the least possible impact, complying with required privacy and technical regulations.

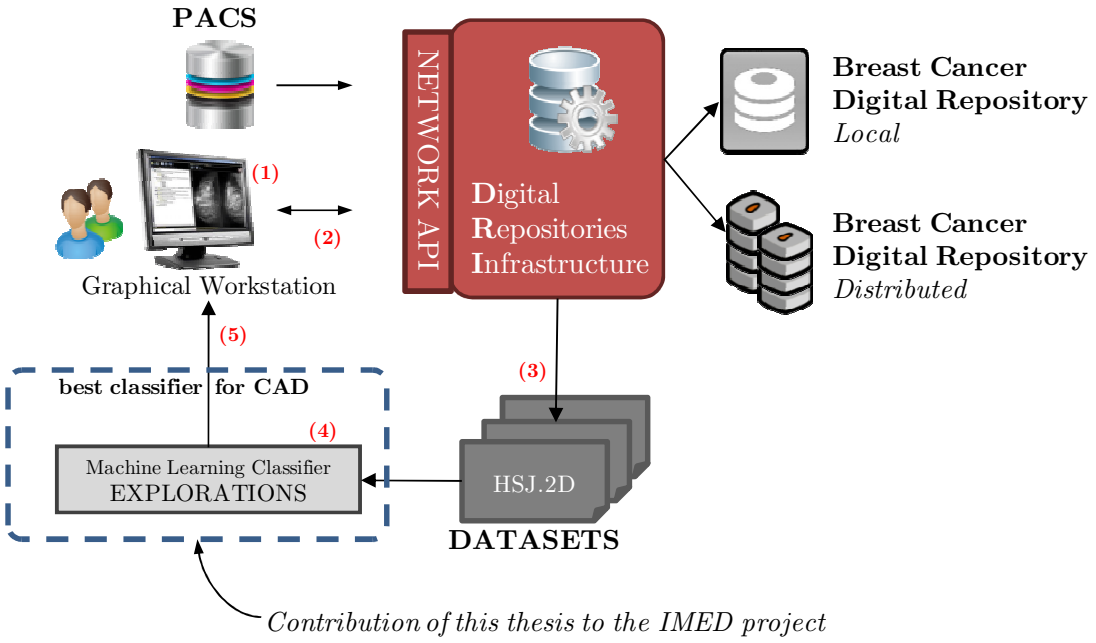


Figure 36: IMED project digital repository and CAD development lifecycle

The BCDR data model (Ramos-Pollan, Franco et al. 2010), hosted at DRI infrastructure set up by the project, is a subset of the DICOM medical file format (NEMA 2008) customized by radiologists at the FMUP-HSJ for storing and managing specific case information related to digital mammography images (see data model in figure 37). At the time of writing BCDR includes samples of all BIRADS (D'Orsi, Bassett et al. 2003) classes and it is composed of over one thousand cases, each one with the associated proven biopsy that constitutes the golden standard. This work complemented recent results in managing DICOM objects within Grid environments, such as the TRENCADIS middleware (Blanquer Espert, Hernández García et al. 2009) and others (Bellotti, Cerello et al. 2007; Glatard, Zhou et al. 2009; Maheshwari, Missier et al. 2009), by applying the DICOM standard at FMUP-HSJ. BCDR is fully integrated with lifecycle to develop machine learning classifiers (figure 36), where (1) mammography images of the BCDR are preprocessed through a graphical workstation, (2) specialized radiologists mark and classify biopsied cases which are then stored in the BCDR, (3) data features are extracted from the stored annotations, (4) MLC configurations are explored and selected and (5) selected MLC are integrated back into the workstation providing automated second opinion diagnosis to doctors. Step 4 constitutes the contribution of this thesis to the project and this is what is shown in this section.

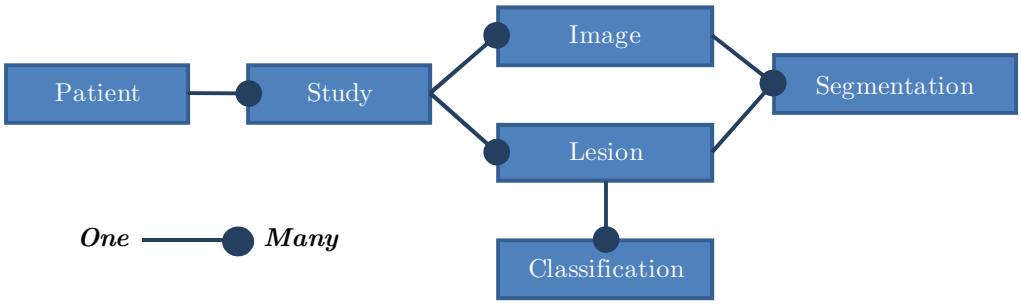


Figure 37: Data model for the Breast Cancer Digital Repository

The BCDR data model (figure 37) supports each patient undergoing one or more studies, each study composed of one or more images (such as digitized film screen mammography images) and one or more lesions. Each image may have one or more segmentations (for different lesions) and each lesion can be associated to several segmentations, typically in mediolateral oblique (MLO) and/or craniocaudal (CC) images of the same breast. Moreover, each lesion can be also linked to many classifications (by different specialists, automatic classifiers, etc.). For each segmented region 18 features are automatically computed and stored forming a features vector, which is representative of the image region statistics, shape and texture. Then, the features vector can be assigned to a certain class by an expert radiologist or a machine learning classifier. The BCDR model supports the possibility to assign to the same features vector several classifications by different clinicians and MLC under different class families. In the IMED project only the BIRADS class family (D'Orsi, Bassett et al. 2003) was considered. BCDR also allows the storage of a variety of sets of experiments of classification runs, performed both by human experts and automatic classifiers, so that later they become available for statistical analysis.

A specialized graphical workstation supports specialists segmenting and diagnosing images as explained in Section 1.1.4. Segmentation is semi-automatic, where the user segments the region assisted by the computer through an interactive technique based on deformable models such as snakes, active shape model, etc. (Cootes, Taylor et al. 1995; Chenyang and Prince 1998) and/or intelligent scissors (Liang, McInerney et al. 2006) also known as livewire.

The datasets containing the extracted features constitute the input for this thesis and well performing discovered are then to be integrated back within specialized user interfaces to be used for second opinion in assisted diagnosis.

5.3 Dataset construction and processing

From BCDR, two specialized radiologists at FMUP-HSJ used a graphical workstation to evaluate and BIRADS classify 286 cases (Ramos-Pollan, Guevara-López et al. 2011). Only cases having both CC and MLO mammography images of left and right breasts were selected, including associated critical information such as lesion type (microcalcification, calcification, mass or asymmetries), biopsies results, etc. Several image processing operations were applied and validated on all selected images to improve ROIs details. The goal was to find fast and simple image preprocessing operations for denoising and enhancing possible pathological lesions or normal tissue image regions. This validation included suitable combinations of pre-processing filters, mathematic morphology, thresholding and edge detection among others techniques. However, the most common defect of mammography images was the poor contrast resulting from a reduced, and perhaps nonlinear, image amplitude range. Then, it was found that in a first preprocessing step, ROI details can be in general improved by adjusting image intensities (a conventional contrast enhancement technique based on amplitude rescaling of each pixel). To enhance images contrast gray scale intensity values of input CC and MLO mammography images were mapped to new values such that 1% of data is saturated at low and high intensities to produce a new image in which the contrast is increased.

Special effort was made by specialists trying to locate possible ROIs for the same lesion in both CC and MLO associated mammography images for each case (see figure 38). This double-segmentation was successfully performed in 126 cases producing each one two features vectors (one for each CC and MLO image), whereas in the remaining 160 cases only one ROI was segmented, either in the CC or in the MLO image, producing one single features vector. This was attributable to various reasons, including technical issues, difficulties in ROI identification in both CC and MLO images or casual contingencies. For each segmentation (in MLO and/or CC images) the extracted features vector contained 18 features including statistics *(skewness, kurtosis, perimeter, area, standard deviation, minimum, maximum, mode and mean)*, shape *(elongation, roughness, form, circularity)* and texture *(correlation, angular second moment, contrast, inverse difference moment, entropy)*. See (Haralick, Dinstein et al. 1973; Rodenacker 2001; López, Novoa et al. 2008)

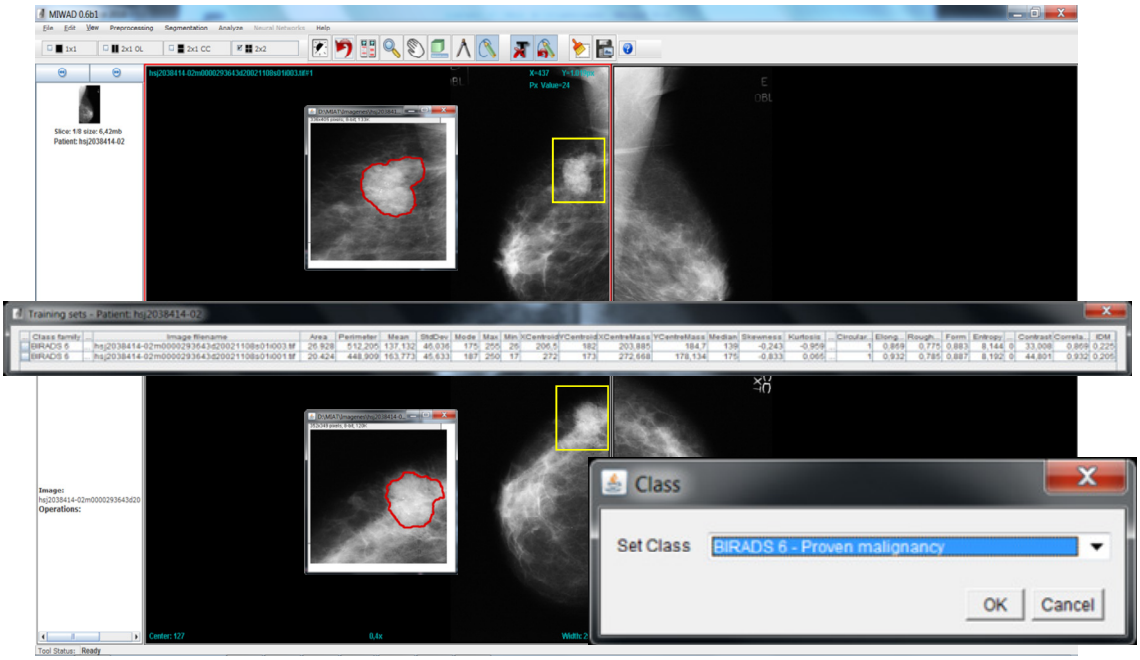


Figure 38: Double segmentation, feature extraction and BIRADS classification

From this raw data three primary datasets were constructed, as shown in figure 39. Dataset HSJ.2D holds all 412 features vectors with 18 features extracted from the 286 cases, where the 126 double-segmentation cases produced 252 vectors and the 160 single-segmentation cases produced one vector each. Dataset HSJ.3DSNGL (for *single*) contains only the 252 vectors produced by the double-segmentation cases. Finally, for each features vector pair from double-segmentation cases, a single vector was formed joining the 18 features segmented from the MLO image and the 18 features segmented from the CC image. This resulted in the HSJ.3DJOIN dataset, containing 126 vectors with 36 features each. With this, the aim was understanding if relating MLO and CC segmentations on same lesion could be exploited to gain classification accuracy. In addition, as shown in figure 39, five datasets were further derived from HSJ.2D in order to understand if including all 18 features or only a selected group of features (shape, texture, statistic or a heuristic selection) would prominently contribute to MLC accuracy over the others. This was not done for the HSJ.3DSNGL and HSJ.3DJOIN since experimentation on HSJDS.2D derived datasets showed that it was best to keep all features. Thus, the five datasets derived from HSJ.2D together with HSJ.3DSNGL and HSJ.3DJOIN, makes a total of seven base datasets.

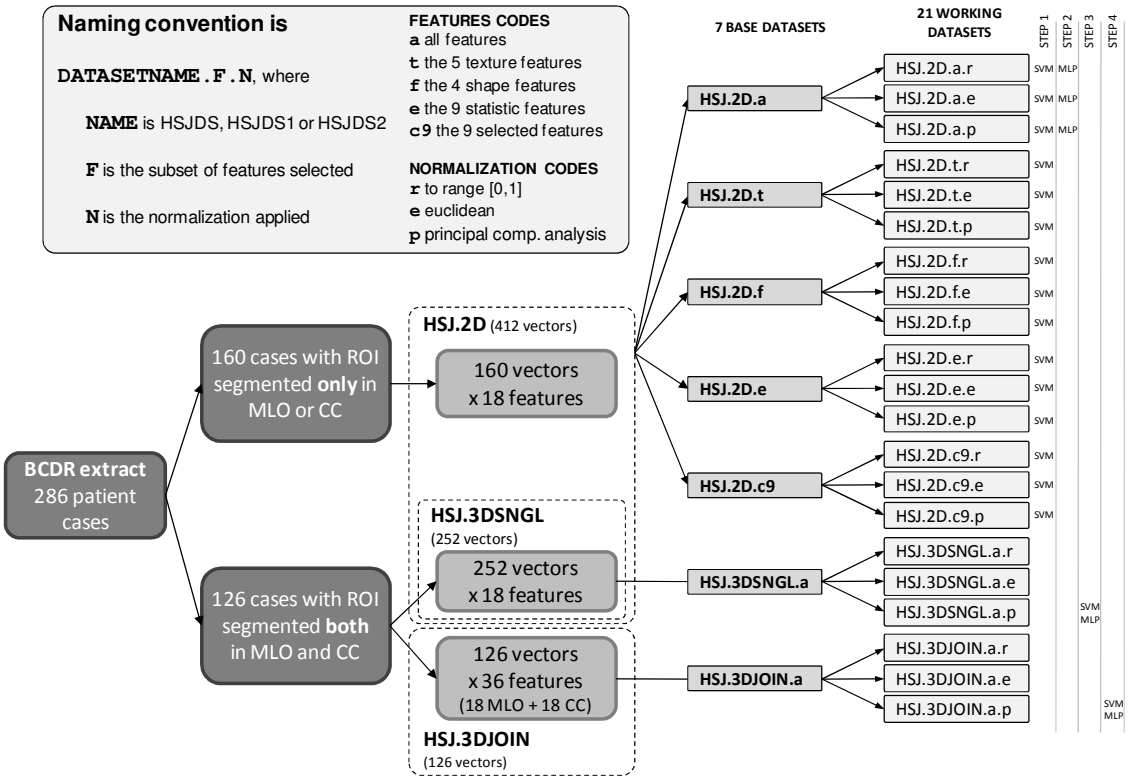


Figure 39: Datasets built from BCDR after classification by specialists

Table 18 shows the distribution of BIRADS classes for each original dataset. Since some BIRADS classes were rather scarce and the primary medical interest was to distinguish benign cancers from malignant ones, BIRADS classes 1, 2 and 3 were all tagged as *benign*, while BIRADS classes 4, 5 and 6 were tagged as *malign*, making therefore all datasets binary.

Table 18: Class distribution for mammography datasets

class	HSJ.2D	HSJ.3DSNGL	HSJ.3DJOIN
BIRADS 1	8	4	2
BIRADS 2	172	146	73
BIRADS 3	75	32	16
BENIGN	**255**	**182**	**91**
BIRADS 4	55	20	10
BIRADS 5	26	16	8
BIRADS 6	76	34	17
MALIGN	**157**	**70**	**35**
TOTAL	**412**	**252**	**126**

Dataset normalization is required before feeding data to any MLC, since different features of the same vector usually take values over ranges of different sizes and nature. This affects MLC performance and in this study each of the seven datasets just described was normalized using three different techniques, seeking to understand how to better preprocess them, producing a total of 21 datasets to explore as shown in figure 39. The three normalization procedures used were **Euclidian, range to [0,1]** and **principal component analysis**.

Euclidean normalization was calculated by $V' = V/||V||$ where $V = (v_1, v_2, \dots, v_n)$ represents the original features vector and $V' = (v_1', v_2', \dots, v_n')$ the normalized resulting vector. $||V||$ is the vector norm defined as $||V|| = \sqrt{V \cdot V} = \sqrt{v_1^2 + \cdots + v_n^2}$

Range normalization [0,1] processes individually each vector's feature v_i to guarantee they all fall within the [0,1] interval and it was calculated as $v_i' = \frac{v_i}{Max_i - Min_i}$ where v_i and v_i' are the original and normalized feature values respectively and Min_i and Max_i are the minimum and maximum values of feature i in all elements of the dataset.

Principal Component Analysis (PCA) was done using the Weka toolkit (Mark Hall, Eibe Frank et al. 2009), reducing the dimensionality of each dataset to account for 99% of its variability.

In summary, first from HSJ.2D, HSJ.3DSNGL and HSJ.3DJOIN seven base datasets were built: HSJ.2D.a (412 features vector, all 18 features), HSJ.2D.t (412 features vectors, with only the 5 texture features), HSJ.2D.f (412 features vectors, with only the 4 shape features), HSJ.2D.e (412 features vectors, with only the 9 statistic features), HSJ.2D.c9 (412 features vectors, with only 9 heuristically selected statistic, shape and texture features), HSJ.3DSNGL.a (252 features vector, all 18 features) and HSJ.3DJOIN.a (126 features vector, all 36 features, 18 from each original vector). Second, we created 21 working datasets that were produced after normalizing each dataset with **range [0,1]**, **euclidian** and **PCA** normalization procedures. All working datasets were finally named as described in figure 39.

5.4 Experimentation and Validation

5.4.1 Goals and metrics

Experimentation is aimed at massively exploring classifier configurations to obtain well performing classifiers for the datasets just described. This includes also gaining understanding on what data preprocessing operations are more convenient (*normalization* and *feature selection*) and what datasets yield better classification results. Besides the specific classifiers obtained, knowledge gained in this endeavor will also be key to design further experiments and data collection processes within the IMED project.

Due to the medical domain within which this data analysis process takes place, AUC is the major metric through which classifier performance is to be measured, compared and, most importantly, communicated to medical staff to share the degree of success of discovered classifiers before they can be used in the CAD construction and validation process.

Together with AUC, strong statistical validation of obtained results is essential since data is typically scarce in these domains and, specially, at the beginning of the project. Special emphasis was put into using strong validation methods profiting from the capabilities offered by BiomedTK/C3 to exploit computing resources. Therefore leave-one-out validation was used whenever possible and bootstrapping in all cases leaving the door open for stronger statistical in further classifier discovery tasks as more data becomes available from the IMED project.

5.4.2 Experimental setup

BiomedTK/C3 was used to explore SVM and MLP based classifiers search spaces for the datasets described above (Ramos-Pollan, Guevara-López et al. 2011). The goal was to find well performing MLC configurations for each dataset and understanding what feature set and normalization procedure would produce best classification. Following the method described in Section 4.2, both for SVMs and MLPs the strategy was first to make general explorations with a wide range of parameters and then performing more fine grained explorations around the MLC configurations yielding better classification performance. Validation was done through the bootstrapping method tagging for testing 40% of each dataset before training each classifier configuration and repeating

the process 5 times to allow statistical smoothing. This is referred to as *testpct40* in the reported results (preserving BiomedTK notation). In addition, since SVMs are computationally more affordable to train, leave-one-out validation was also used for all SVM configurations (denoted as *one2all* in experiments) where a dataset of size n is trained n times, each one with n-1 elements used for training, the one left out for testing and averaging the results. For MLPs, all available engines in BiomedTK were used, which includes the original Encog ones (*ffbp*, *ffrp*, *ffsa* and *ffga*) and their counterparts modified for AUC optimization (*ffbproc*, *ffrproc*, *ffsaroc*, *ffgaroc*) as explained in Section 3.3. This way, the behavior of AUC optimized MLPs resulting from this thesis could also be observed in this application case.

Well performing classifiers resulting from exploring the 21 working datasets would become the targets to be integrated into CAD systems, with the corresponding image and data preprocessing. In total, for each dataset, 1200 SVM configurations and another 240 MLP configurations were set up in the following way:

- 600 SVM configurations with *one2all* validation (480 configurations for a general exploration and the rest for finer ones). Observe that with *one2all* validation each configuration is trained once per element of the target dataset.
- 600 SVM configurations with *testpct40* validation (the same 600 configurations as with *one2all* validation, trained 5 times each one with *testpct40*).
- 240 MLP configurations with *testpct40* validation. This is, 30 configurations for each of the 8 MLP engines, trained 5 times each one with *testpct40* (20 configurations for a general exploration and 10 configurations for the finer ones).

Not all configurations were trained for each one of the 21 datasets, since as experimentation begun, it was rapidly observed that certain datasets would systematically yield worse classification results, so they were left out of the rest of the experiments for a more rationale usage of the computing resources. At the end, datasets were explored in the following sequence:

Step 1. HSJ.2D with **SVM**. The 15 normalized datasets derived from HSJ.2D.a, HSJ.2D.t, HSJ.2D.f, HSJ.2D.e and HSJ.2D.c9 with SVMs, including all (412) features vectors.

Step 2. HSJ.2D.a with **MLP**. In the step above it was observed that selecting all available features (working datasets derived from HSJ2D.a) consistently produced better classification results so, since MLPs are much more computationally expensive to train, explorations were restricted only to three datasets: HSJ.2D.a.p, HSJ.2D.a.r and HSJ2D.a.e. In Step 1 it was also observed that Euclidean normalization tends to yield the worse results within the same dataset. However, as this seems to be secondary to feature selection, it was decided that Step 2 would still include datasets with the three normalization procedures.

Step 3. HSJ.3DSNGL.a.p with **SVM** and **MLP**. At this step exploration of the datasets containing only CC and MLO related segmentations started with the dataset including only 256 selected features vectors and all (18) features with **PCA** normalization. In addition to what was concluded by Step 1 above (it is best to use all available features), Step 2 confirmed that Euclidean normalization performs consistently worse and that both PCA and range normalization gave similar results. Therefore, only dataset HSJ.3DSINGLE.a.p was explored.

Step 4. HSJ.3DJOIN.a.p with **SVM** and **MLP**. Dataset including only 126 selected features vectors and 36 features with all joint features and **PCA** normalization. Only dataset HSJ.3DJOIN.a.p was explored for the same reasons as above.

In total, 17 out of the 21 generated datasets were explored with SVMs (training 1200x17=20400 configurations) and five with MLPs (training 240x5=1250 configurations). Marks in figure 39 show which dataset was trained with SVMs and which one with both SVMs and MLPs. This took around 200 days of CPU time, requiring over four physical days on the public gLite Grid computer cluster at CETA-CIEMAT with 50 CPU cores with BiomedTK/C3. After training and validation, each MLC configuration produced two measures for classifier performance, on the test part of the dataset: **test.PCT**, the percentage of elements correctly classified (*accuracy*) and **test.AUC,** the area under the ROC curve.

5.4.3 Results and discussion

Tables 19 summarizes the results for the explorations performed with SVMs on the 15 working datasets derived from HSJ.2D for Step 1 as described above. Table 20 summarizes the best 10 results obtained with MLP engines on HSJ.2D.a derived

datasets for Step 2. Tables 21 and 22 show the specific configurations of the best classifiers obtained for the HSJ.3DSNGL and HSJ.3DJOIN for Steps 3 and 4 respectively. ROC curves for best **test.AUC** marked in gray in all tables are plotted in figure 40. It is important to remark that, although for simplicity results from *one2all* and *testpct40* validation methods are shown together (specially in tables 21 and 22) their comparative interpretation must be undertaken with care. In *one2all* a dataset of size n is trained n times, each one labeling n-1 elements for training and one for testing. Each time, the accuracy of the test part of the dataset (only one element) is either 0% or 100%, but then it is averaged over all elements of the dataset once the n training processes are completed. With this, test.PCT and test.AUC measure classifier performance on the whole dataset (since each element is used for testing once). In particular, test.PCT represents more a proper probability rather than an averaged classifier score. On the other hand, with *testpct40* classifier performance (test.PCT and test.AUC) refers only to the 40% selected as test instances. Moreover, it is more subject to outliers, since random selection of the test elements may eventually favor those easier to classify.

Table 19: Results summary for HSJ.2D datasets (SVMs/Step 1 by test.AUC)

		test.PCT			test.AUC		
dataset	**configs**	**max**	**avg**	**stdev**	**max**	**avg**	**stdev**
one2all validation							
HSJDS.2D.a.p	600	0,689	0,667	0,010	**0,683**	0,656	0,011
HSJDS.2D.a.r	600	0,687	0,662	0,013	**0,677**	0,654	0,010
HSJDS.2D.c9.p	600	0,675	0,634	0,017	**0,661**	0,589	0,036
HSJDS.2D.f.e	600	0,655	0,635	0,010	**0,649**	0,518	0,067
HSJDS.2D.c9.r	600	0,667	0,634	0,014	**0,647**	0,602	0,028
HSJDS.2D.a.e	600	0,653	0,621	0,018	**0,632**	0,506	0,063
HSJDS.2D.c9.e	600	0,648	0,599	0,022	**0,632**	0,498	0,084
HSJDS.2D.e.e	600	0,648	0,614	0,014	**0,625**	0,440	0,096
HSJDS.2D.e.r	600	0,658	0,632	0,013	**0,623**	0,583	0,018
HSJDS.2D.e.p	600	0,655	0,631	0,011	**0,623**	0,582	0,026
HSJDS.2D.t.r	600	0,629	0,609	0,008	**0,594**	0,407	0,081
HSJDS.2D.f.p	600	0,653	0,620	0,017	**0,587**	0,506	0,040
HSJDS.2D.f.r	600	0,653	0,630	0,009	**0,578**	0,504	0,049
HSJDS.2D.t.p	600	0,646	0,610	0,020	**0,538**	0,403	0,059
HSJDS.2D.t.e	600	0,631	0,614	0,008	**0,530**	0,431	0,052
testpct 40 validation							
HSJDS.2D.a.r	600	0,750	0,654	0,031	**0,778**	0,660	0,053
HSJDS.2D.a.p	600	0,733	0,655	0,030	**0,770**	0,664	0,055
HSJDS.2D.c9.r	600	0,744	0,637	0,070	**0,755**	0,601	0,065
HSJDS.2D.c9.p	600	0,738	0,638	0,045	**0,743**	0,605	0,056
HSJDS.2D.a.e	600	0,709	0,613	0,021	**0,733**	0,511	0,104
HSJDS.2D.c9.e	600	0,671	0,611	0,027	**0,724**	0,519	0,108
HSJDS.2D.t.r	600	0,655	0,607	0,020	**0,723**	0,504	0,092
HSJDS.2D.e.p	600	0,709	0,633	0,041	**0,721**	0,596	0,062
HSJDS.2D.f.p	600	0,695	0,624	0,035	**0,720**	0,568	0,081
HSJDS.2D.f.r	600	0,689	0,625	0,036	**0,712**	0,570	0,088

HSJDS.2D.e.r	600	0,709	0,635	0,048	**0,707**	0,599	0,056
HSJDS.2D.t.p	600	0,659	0,608	0,031	**0,704**	0,504	0,088
HSJDS.2D.e.e	600	0,677	0,613	0,028	**0,702**	0,519	0,074
HSJDS.2D.f.e	600	0,691	0,626	0,039	**0,700**	0,555	0,080
HSJDS.2D.t.e	600	0,667	0,611	0,034	**0,666**	0,489	0,109

Classification Performance of HSJ.2D datasets. Table 19 summarizes the performance of obtained SVM classifiers for the HSJ.2D datasets (Step 1) per classifier engine and validation type. Each line summarizes the number of MLC configurations trained for each working dataset derived from HSJ.2D, separating results from explorations performed with *one2all* or *testpct40* validation. For ach dataset, it shows the number of SVM configurations trained, along with the maximum, the average and the standard deviation for test.AUC and test.PCT obtained with those SVM configurations. The maximum value refers to a specific configuration, and the best test.AUC and test.PCT were in some cases obtained by the different configurations. SVM-based classifiers using the *one2all* validation method yielded a maximum test.AUC of 0,683 and 0,778 with *testpc40*. Table 20 shows similar information for MLPs. In this case (Step 2), only the three datasets derived from HSJ.2D.a were explored with each one of the eight MLP available engines (*ffbp*, *ffbproc*, *ffrp*, *ffrproc*, *ffsa*, *ffsaroc*). This makes 24 combinations and table 20 shows the top ten combinations producing best maximum test.AUC. The best test.AUC was 0.788. These results are marked in gray in both tables and their ROC curves are plotted in figure 40 (left). This allows us to consider that: (1) range [0,1] and PCA normalization are more suitable for HSJ.2D than Euclidean normalization (2) there is a non negligible difference between SVM results using *one2all* and *testpct40* validation, recalling the remarks made above; and (3) similar classification performance results were obtained in SVM and MLP based classifiers (with *testpct40*).

Table 20: Top ten results for HSJ.2D datasets (MLPs/Step 2 by test.AUC)

			test.PCT			**test.AUC**		
dataset	**engine**	**configs**	**max**	**avg**	**stdev**	**max**	**avg**	**stdev**
HSJDS.2D.a.p	ffsaroc	30	0,709	0,656	0,025	**0,788**	0,704	0,044
HSJDS.2D.a.r	ffsa	30	0,733	0,659	0,046	**0,771**	0,700	0,043
HSJDS.2D.a.p	ffbproc	30	0,721	0,640	0,054	**0,758**	0,679	0,032
HSJDS.2D.a.r	ffsaroc	30	0,726	0,650	0,038	**0,752**	0,683	0,044
HSJDS.2D.a.p	ffrproc	30	0,712	0,639	0,041	**0,750**	0,668	0,043
HSJDS.2D.a.p	ffgaroc	30	0,703	0,619	0,037	**0.746**	0,667	0,059
HSJDS.2D.a.p	ffbp	30	0,709	0,635	0,039	**0,744**	0,662	0,045
HSJDS.2D.a.p	ffga	30	0,709	0,618	0,052	**0,737**	0,622	0,060
HSJDS.2D.a.r	ffbproc	30	0,673	0,631	0,021	**0,735**	0,669	0,038
HSJDS.2D.a.r	ffsaroc	30	0,695	0,637	0,036	**0,733**	0,675	0,039

Classification Performance of HSJ.3DSNGL.a.p and HSJ.3DJOIN.a.p datasets. As explained, Steps 3 and 4 explored only the HSJ.3DSINGLE.p.a and HSJ.3DJOIN.p.a datasets, normalized by PCA and using both SVM and MLP based classifiers. In order to show concrete classifier configurations discovered, tables 21 and 22 detail the best classifiers respectively obtained for the HSJ.3DSINGLE.p.a and HSJ.3DJOIN.p.a datasets with SVM and MLP configurations, including the classifier parameters used. A significant increase of test.PCT and test.AUC values in classifiers for both datasets can be observed with respect to all HSJ.2D datasets. The highest test.AUC values (0,996 in HSJ.3DJOIN.a.p and 0,992 in HSJ.3DSINGLE.a.p) were produced by MLP-based classifiers (with *testpct40* validation). However, SVM-based classifiers also produced high test.AUC values (0,984 in HSJ.3DJOIN.a.p and 0,953 in HSJ.3DSINGLE.a.p). Figure 40 (center and right plots) shows the ROC curves corresponding to these test.AUC values.

Table 21: Best configurations discovered for HSJ.3DSNGL.a.p dataset

engine	config params	config values	validation	test.PCT	test.AUC
SVM CONFIGURATIONS					
svm	kernel degree gamma shrink coef0 cost weight probestimates	pol 2 0.0048 true 1.0 64.0 0.5 true	one2all	0,917	**0,953**
svm	kernel degree gamma shrink coef0 cost weight probestimates	pol 2 0.0010 true 0.6 512.0 1.0 true	one2all	0,933	**0,950**
svm	kernel degree gamma shrink coef0 cost weight probestimates	sigm 2 0.0010 true 0.6 512.0 1.0 true	testpct 40	0,901	**0,949**
svm	kernel degree gamma shrink coef0 cost weight probestimates	pol 2 0.0010 true 0.1 1.0 0.1 true	testpct 40	0,885	**0,946**
MLP CONFIGURATIONS					
ffbp	layers and neurons learnrate momentum epochs	[18:27:14:7:2] 0.1 0.2 500	testpct 40	0,950	**0,992**
ffsaroc	layers and neurons starttempendtemp cycles epochs	[18:27:14:7:2] 100.0 2.0 100 200	testpct 40	0,920	**0,966**
ffbproc	layers and neurons learnrate momentum epochs	[18:27:14:7:2] 0.1 0.2 500	testpct 40	0,931	**0,956**
ffsa	layers and neurons starttempendtemp cycles epochs	[18:27:14:7:2] 100.0 2.0 100 200	testpct 40	0,911	**0,951**

In general, starting this exploration endeavor with SVMs (Step 1) was beneficial because it allows a first filter on datasets with affordable computer power, allowing a more rational effort when exploring MLPs which, in the case of HSJ.3DSNGL and HSJ.3DJOIN datasets ended up yielding slightly better performance. Additionally, it can be seen that the AUC optimization proposed in Chapter 3 for MLPs (*ffbproc, ffsaroc*, etc.) improves the obtained AUC when performance is far from optimal (tables 19 and 20), consistently with results obtained by experiments in Section 3.3.

Table 22: Best configurations discovered for HSJ.3DJOIN.a.p dataset

engine	config params	config values	validation	test.PCT	test.AUC
SVM CONFIGURATIONS					
svm	kernel degree gamma shrink coef0 cost weight probestimates	pol 2 0.0048 true 1.0 64.0 0.5 true	one2all	0,937	**0,984**
svm	kernel degree gamma shrink coef0 cost weight probestimates	pol 2 0.0010 true 0.1 1.0 0.1 true	testpct 40	0,937	**0,982**
svm	kernel degree gamma shrink coef0 cost weight probestimates	sigm 2 0.0010 true 0.1 1.0 0.1 true	one2all	0,921	**0,981**
svm	kernel degree gamma shrink coef0 cost weight probestimates	rbf 2 0.01 true 0.6 1.0 1.0 true	testpct 40	0,913	**0,981**
MLP CONFIGURATIONS					
ffbp	layers and neurons learnrate momentum epochs	[36:54:27:14:2] 0.1 0.2 500	testpct 40	0,941	**0,996**
ffsa	layers and neurons starttempendtemp cycles epochs	[36:54:27:14:2] 100.0 2.0 100 200	testpct 40	0,940	**0,983**
ffbproc	layers and neurons learnrate momentum epochs	[36:54:27:14:2] 0.1 0.2 500	testpct 40	0,902	**0,964**
ffsaroc	layers and neurons starttempendtemp cycles epochs	[36:54:27:14:2] 100.0 2.0 100 200	testpct 40	0,902	**0,960**

Validation method. Results in (Efron 1983) indicate that *one2all* (leave-one-out) validation gives a nearly unbiased estimator for classifier accuracy, but often with high variability, specially in small datasets. Our experiments showed little variability on all classifiers (both on accuracy and AUC), which encourages us to place stronger confidence in our results, at the expense of additional computing power required to train classifiers. Interestingly enough, results of *testpct40* and *one2all* validation differ very little in the HSJ.3DSINGLE.a.p and HSJ.3DJOIN.a.p datasets, which might suggest that *testpct40* could be used for these datasets instead of *one2all*, reducing significantly computing time. The fact that this is not exactly the case in HSJ.2D datasets, but still *one2all* and *testpct40* results are quite close, suggests that using a different percentage in *testpct* might lead to similar results, reducing as well the computing requirements of explorations for those datasets. This might be the subject for further analysis, specially for biomedical datasets, since more data is being generated as the project at the FMUP-HSJ continues, and statistically consistent results at reduced computing costs will be key to allow us place increasingly stronger confidence on the automatic diagnoses made by the systems developed by the project.

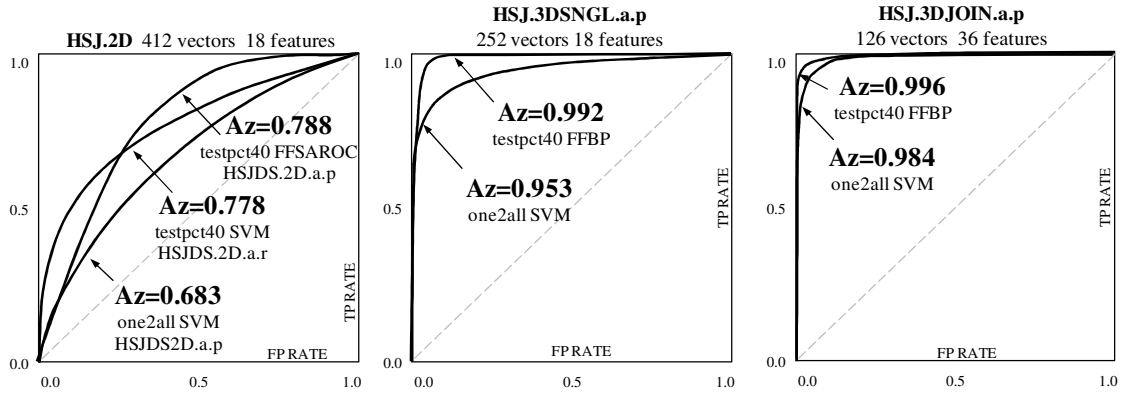


Figure 40: ROC Curves for best classifiers on HSJ datasets

The results just described (Ramos-Pollan, Guevara-López et al. 2011) can be compared with those reported in Section 2.3 - *Machine learning classifiers for breast cancer CAD*. However, interpretation of this comparison must be undertaken with care, accounting for the different datasets and experimental conditions with which the different results are obtained across the literature sources. Comparison of these results with the ones obtained with the FMUP-HSJ datasets is not straight forward although the final classification goal is analogous, due to the different nature of the datasets. In addition, results reported in Section 2.3 can also be compared with those obtained for the *bcw* UCI dataset in Chapter 4 to validate the BiomedTK/C3 software frameworks and reported in table 29 on page 159

5.5 Conclusion

The aim of this Chapter was to show the developments of this thesis applied in practice to a real world problem. With this, three major contributions are provided by this work to the IMED project within which this thesis came into existence:

1. The specific classifier configurations discovered through the exploration process enabled by the BiomedTK/C3 software frameworks. Additionally, these configurations include the ones resulting from modifying multilayer perceptrons for AUC optimization as described in Chapter 3. These configurations are to be included within the graphical workstations used by medical doctors to build CAD systems for assisted diagnosis or automated second opinion.

2. The knowledge gained on what dataset preprocessing operations (*normalization* and *feature selection* in this case) seem to be more

appropriate for the problem in hand. In fact, it is the capability to harness computing power to train constructed datasets in an agile manner that allowed gaining this knowledge and enabled tuning further exploration strategies as knowledge was being built.

3. The data analysis process itself using BiomedTK/C3 constitutes now a fine tuned workflow ready to continue discovering new classifier configurations as new datasets are made available through the IMED project, produced by further segmentations and annotations by specialized radiologists.

Besides this, through the application domain addressed in this Chapter (discovering machine learning classifiers for breast cancer CAD), the aim was also to show the diversity and complexity of situations that can be handled with the contributions of this thesis, leading to a better knowledge of the application domain. In particular, the results herewith described, open questions for further research within the IMED project such as:

- Is there an appropriate split percentage when using *testpct* validation (bootstrap) on HSJ datasets so that its results are comparable to those of *one2all* (leave one out) but at a reduced computational cost?
- Is the increased performance found when using HSJ.3D datasets statistically significant?
- If so, since the performance improvement is so noticeable, is it worthwhile to focus the IMED project objectives into building a CAD system enforcing double segmentation to ensure better CAD performance when the user (a medical doctor) resorts to it for second opinion?

Regardless how the IMED project finally settles these (and other) issues, it is due to the results obtained with this thesis that they were raised and their resolution can contribute to the success of the project. So beyond the specific classifier configurations and dataset preprocessing criteria obtained, these issues illustrate how the experimental reach of the application domain is augmented by the artifacts produced by this work (BiomedTK/C3 and enabling AUC optimization).

Chapter 6

Conclusions

6.1 Development of this thesis

The different developments and findings resulting from this thesis were motivated by the need to find better machine learning classifiers for the CAD methods and systems that were being developed within the IMED project. Retrospectively, working towards this driving aim, most contributions resulted by from the need to overcome specific difficulties as they arose and the opportunity to extend their reach beyond the scope of the project. The following points summarize briefly the historical development of the contributions described in this document:

1. The need to find better machine learning classifiers for CAD methods was defined within the IMED project. See Section 1.2 (Calanducci, Ramos-Pollan et al. 2008; Ramos-Pollan, Guevara López et al. 2009).

2. AUC was acknowledged to be a commonly used metric in medical environments and, thus, its usage appropriate for the IMED project. Additionally, literature review showed evidence that error rate minimization (as typically targeted by machine learning methods) does not necessarily yield to AUC optimization, which lead to think that an opportunity to enable a more systematic use of AUC within machine learning in general could exist.

3. An AUC optimization method affordable to use in existing machine learning methods was defined. See Section 3.3 (Ramos-Pollan, Guevara Lopez et al. 2010)

4. The multilayer perceptron training algorithms delivered through the Encog toolkit were modified for AUC optimization using the method previously defined (Section 3.3). This included the feedforward backpropagation, resilient propagation, simulated annealing and genetic algorithms (Ramos-Pollan, Guevara Lopez et al. 2010).

5. Confronted with the large amounts of computing power available at the research institutions where the IMED project evolves and this thesis was developed, it was determined to exploit them to (1) validate the AUC optimization method just defined and (2) massively explore the search space of classifiers configurations to discover well performing configurations suitable for CAD. Therefore, a software framework (BiomedTK) was developed to manage explorations of classifier configurations over Grid eInfrastructures serviced by the gLite middleware (Section 4.2).

6. It was observed that performance of the modified multilayer perceptron training algorithms degraded largely when using the new AUC optimization method. This was due to the fact that they are heavily iterative algorithms and AUC calculating occurred constantly during the iterative processes. In some cases this degradation rendered the algorithms impractical to use.

7. A method was defined to provide a fast computation of the AUC, by making an error bounded approximation, having the error to be as small as desired by the researcher. After definition, the method was implemented and validated experimentally (Section 3.2).

8. The modified training algorithms were updated to use the fast AUC method just developed. With this, the AUC optimization method as applied to multilayer perceptrons was experimentally validated (Section 3.3).

9. Grid resources were started to be used through BiomedTK during this experimental validation, but it was soon observed a large dependency on the underlying gLite middleware. Seeking to isolate BiomedTK from gLite changes and instabilities, the possibility to generalize access to eInfrastructures was identified to be feasible and affordable (in development

effort) by using Java industry standards to implement a job agents/colony model.

10. A software framework was then developed to enable this decoupling, providing a simple and fast mechanism to exploit computing resources available throughout eInfrastructures of different nature for users and applications in general (like BiomedTK). This software framework was named C3 (for *Cloud Computing Colonies*) and it was developed initially to use gLite Grid, Amazon EC2 Cloud and SSH available distributed computing resources (Section 4.3). BiomedTK was updated to use C3 rather than gLite directly. Together, BiomedTK and C3 were thoroughly tested. See Section 4.4 (Ramos-Pollan, Guevara-López et al. 2011).
11. Finalized experimental validation for AUC optimization, now over BiomedTK/C3 (Section 3.3.2)
12. Application in breast cancer CAD was now overtaken as data started being available from specialized radiologists annotating and classifying mammograms in the IMED project. In a first stage this initial raw data had to be processed to construct the datasets so that machine learning classifiers could be evaluated upon them. See Section 5.3 (Ramos-Pollan, Rubio del Solar et al. 2009; Ramos-Pollan, Franco et al. 2010; Ramos-Pollan, Franco et al. 2010; Ramos-Pollan, Rubio del Solar et al. 2010)
13. Massive machine learning classifier exploration was undertaken on the datasets resulting from the step above over BiomedTK/C3. Discovered classifier configurations are finally handed over to the IMED project and become targets upon which CAD systems are being built and validated. Section 5.4 (Ramos-Pollan, Guevara-López et al. 2011).

6.2 Main conclusions

Through the different Chapters of this document it has been shown how the initial objectives set forth have been accomplished by developing the contributions announced in Section 1.3. These contributions are now reviewed in the light of the results exposed so far:

Contribution 1: *A new AUC based error definition (loss function) for machine learning algorithms.* The definition is based on determining the contribution of the

score of each dataset element to the Wilcoxon-Mann-Whitney statistic and measuring how far it is from the maximum possible contribution. Section 3.3 (Ramos-Pollan, Guevara Lopez et al. 2010). This measure can now be used in a variety of situations where per-element ROC analysis is required, such as for AUC optimization in machine learning (see Contribution 3 below)

Contribution 2: *An efficient error-bounded AUC approximation method with arbitrary precision.* The method is based on discretizing the space for element scores into fixed length intervals, counting the positive and negative elements that correspond to each interval and then, approximating the contribution of each dataset element by adding up the number of elements in intervals below the one it belongs to. Additionally, the method produces per-element AUC error measures as defined in Contribution 1 above so that they can be readily used by machine learning algorithms for AUC optimization. Experiments described in Section 3.2 compare the time required by the proposed method with the AUC computation algorithm provided by Weka, based on sorting dataset elements using fast Java data structures. Measured speedups (number of times the proposed method runs faster) vary with datasets. Synthetically generated datasets based on drawing positive and negative elements from different normal distributions yielded speedups between 3.5 and 6.5. Experiments with a diversity of binary UCI datasets yielded speed ups between 2 and 7.

Contribution 3: *A methodology to integrate the AUC based error definition and the AUC approximation procedure into existing multilayer perceptrons, applicable to other ML methods.* The previous AUC based error definition was used to define a new loss function that can substitute the ones used by machine learning classifiers and, thus, enabling them for AUC optimization. It was also shown that this new loss function retains the property that the error of the whole dataset is the arithmetic mean of the error of all dataset elements. In the case of multilayer perceptrons, a further step was required to map the loss function to particular values of output neurons (Section 3.3). Experimentation included multilayer perceptrons with different kinds of training algorithms using both per-element error and global dataset error over a variety of UCI datasets (Ramos-Pollan, Guevara Lopez et al. 2010). An average global improvement in AUC of 5.86% was achieved when the loss function is substituted when training multilayer perceptrons with backpropagation, resilient propagation, simulated annealing and genetic algorithms.

Contribution 4: *A software framework for integrating third party machine learning classifiers, enabling the exploration of the search space formed by the possible parameters configurations of the model fitting processes implemented by the integrated classifiers.* The software framework developed (BiomedTK) is described in Section 4.2. It allows managing complex explorations consisting of many classifier configurations from third-party machine learning toolkits over different datasets through a reduced set of configuration artifacts (exploration configuration files). This, together with the C3 framework resulting from Contribution 5 below, enabled the exploitation of distributed computing resources for massively evaluating machine learning classifiers for selected UCI datasets, reaching in reasonable time results comparable to those reported in different literature sources (Ramos-Pollan, Guevara-López et al. 2011).

Contribution 5: *A software framework developed upon industry standards allowing (1) launching and maintaining colonies of Job Agents over computing resources and (2) submitting jobs to the Job Agents colonies through command line and API interfaces.* The software framework developed (named C3, for *Cloud Computing Colonies*) is described in Section 4.3 and it can be used seamlessly by applications through its API interface to exploit computing resources available through gLite Grid middleware, Amazon EC2 Clouds and SSH access. It shields applications and users from specific details of accessing a particular eInfrastructure offering a unified interface regardless the final service or hardware where the colonies of C3 Job Agents are maintained by C3. Any application can use the C3 API to interface with C3 Job Agents anywhere they are deployed, and BiomedTK constituted the first application through which C3 has been thoroughly tested.

Contribution 6: *An exploration methodology for the rational usage of the software frameworks (produced in contributions 4 and 5) over local and distributed computing resources.* This methodology is described in Chapter 4 and is based on iteratively gaining understanding on what classifier configurations work better with a certain dataset in hand. The method benefits from the agility provided by BiomedTK/C3 to easily define classifier explorations and use efficiently computing resources to evaluate them (Ramos-Pollan, Guevara-López et al. 2011). This way, researchers can gradually build exploration strategies by evaluating many third party classifiers indistinctly on local or remote computing resources as appropriate in their knowledge acquisition endeavor.

Contribution 7: *An application of the above contributions to search for well performing ML classifiers for breast cancer CADx based on medical data extracted from mammograms.* This constitutes the final contribution of this thesis to the IMED project within which it was conceived. Chapter 5 details the whole process by which datasets produced by the project were processed and fed into a massive classifier exploration process over distributed computing resources (using BiomedTK/C3) that yielded well performing classifiers with AUC greater than 0.9 in certain cases (Ramos-Pollan, Guevara-López et al. 2011). In addition, this process shed light on other issues which may become relevant for the IMED project such as the kind of dataset preprocessing operations that might be more convenient (feature selection, normalization) and for what medical workflows CAD systems might be better suit (such as for double MLO and CC segmentation of mammograms).

In the same way, development of these contributions provided the evidence to sustain the hypothesis sought for as stated in Section 1.3. This can be reviewed in detail in the *Experimentation and Validation* sections of each chapter and it is summarized here:

Hypothesis 1: *Multilayer perceptrons can be improved through new AUC based error measures, providing guidance to existing training methods to yield better AUC classifier performance.* Experimental results in Section 3.3.2 show how the AUC of multilayer perceptrons trained with different algorithms for a variety of datasets is consistently improved by injecting the AUC based error metrics defined in Section 3.3.1 into the existing loss function used by the algorithms. Details can be found in tables 25, 26 and 27 in Appendix I.

Hypothesis 2: *Computing power harnessed by eInfrastructres enables systematic exploration of search spaces of machine learning classifiers configurations for given datasets, specifically biomedical datasets.* The software frameworks developed within this thesis (BiomedTK/C3) constitute the basis to sustain this hypothesis. Experimentation carried out through Chapter 3, 4 and 5 shows how distributed computing resources can be effectively used for large explorations of configurations of machine learning classifiers for different purposes. These computing resources are provided by eInfrastructures of different nature (Grid, Cloud) which can be accessed in a seamless and unified manner through BiomedTK/C3. In Chapter 3 the goal was to validate and measure the performance of modified training algorithms for AUC optimization. In Chapter 4, classifier configurations for commonly used datasets were

explored with the aim to reach, in reasonable time, results comparable to those reported in existing literature sources. In Chapter 5, computing resources were used to discover well performing classifiers to be used in breast cancer CAD systems. In all cases, thousands of classifier configurations were evaluated with a few configuration artifacts across a large variety of datasets consuming a wealth of CPU hours.

Hypothesis 3: *Computing power harnessed by eInfrastructures enables thorough validation of new classification methods.* The AUC optimization method proposed in this thesis was validated through the experimentation described in Section 3.3. Experiments were designed to include several datasets, multilayer perceptron engines and configurations to gather enough statistical evidence of the obtained results. This required access to a substantial amount of computing power to evaluate all devised combinations of classifier configurations and datasets. Without such computing power confidence on the proposed method would remain partial. This was provided by different eInfrastructures (mostly Grid) which, managed through the software frameworks developed in this thesis (BiomedTK/C3), offered the material means to implement a comprehensive methodology for classifier evaluation.

Hypothesis 4: *Given the above three hypothesis, it is possible to develop more precise and robust breast cancer CADx methods.* The machine learning classifier explorations performed in Chapter 5 illustrate how the different results of this thesis can contribute to find well performing classifiers for datasets derived from breast cancer data. Furthermore, results reported here raise additional questions which contribute to focus the construction of CADx method regarding their suitability for certain medical workflows and the data preprocessing operations that are more appropriate. However, it is not only the specific classifier configurations discovered in the context of this thesis which supports this hypothesis (whose performance is already promising at AUC $>$ 0.9), but also the data analysis workflow that was enabled by the different contributions of this thesis. Through this workflow, classifiers for breast cancer data can be evaluated and discovered in a continuous and systematic manner as new annotated data is being generated in the medical environments where CADx systems are being developed.

In summary, the contributions of this thesis suggest that (1) it is possible to reuse existing machine learning algorithms for AUC optimization, as demonstrated in the case of multilayer perceptrons and (2) performing massive explorations of machine learning configurations over distributed infrastructure of different nature (Grid, Cloud)

is feasible with little configuration and management effort. In both cases, efficiency is understood in the sense of economy in the effort required to undertake a research endeavor by possessing the appropriate tools, but also by devising the adequate methods to use them. Their reach is thus expanded by both the conceptual advances and the material means through which they can be taken into practice. In retrospective, it is the tools developed and shown in Chapter 4 that enabled the validation of the theoretical constructs developed in Chapters 3 and their practical application in Chapter 5.

From the IMED project perspective, the results obtained in this thesis contribute to its evolution to new phases enabling the final stages to construct CAD methods, but also helping in building confidence on the data acquisition and analysis processes put into place by the different professionals participating in the project (medical doctors, computer engineers, etc.)

6.3 Future work

Future work will be focused on two issues. First, the classifier discovery workflow herewith described will continue analyzing datasets as they are being generated from radiologists annotating mammograms, producing more classifier configurations suitable for CAD, shedding light on the existing research questions and, probably, raising new ones. Second, a process will start to clinically validate the CAD methods constructed with the classifiers obtained and their associated dataset preprocessing operations. Besides having a clinical assessment of specific classifier and CAD methods the aim is to establish a continuous validation workflow that can run as new data and classifiers are generated.

In addition, two further issues will be addressed. On one side, other widely used machine learning engines will be incorporated to BiomedTK, enlarging the capabilities of its explorations. The ones available through the Weka toolkit are the first priority. Then, the AUC optimization method described in Chapter 3 will be injected in additional machine learning algorithms as considered appropriate either for purely research reasons or project driven such as, for instance, considering that a certain family of classifiers raises the interest of the IMED project.

Chapter 7

Bibliographic References

Abonyi, J. and F. Szeifert (2003). "Supervised fuzzy clustering for the identification of fuzzy classifiers." Pattern Recognition Letters **24**(14): 2195-2207.

Agarwal, S., T. Graepel, et al. (2005). "Generalization bounds for the area under the ROC curve." Journal of Machine Learning Research **6**: 393-425.

Airola, A., T. Pahikkala, et al. (2009). A comparison of AUC estimators in small-sample studies. Proceedings of the 3rd International workshop on Machine Learning in Systems Biology (MLSB 09), Ljubljana, Slovenia.

Airola, A., T. Pahikkala, et al. (2011). "An experimental comparison of cross-validation techniques for estimating the area under the ROC curve." Computational Statistics & Data Analysis **55**(4): 1828-1844.

Akaike, H. (1974). "A new look at the statistical model identification." IEEE Transactions on Automatic Control **19**(6): 716-723.

Alpaydin, E. (2010). Introduction to machine learning, The MIT Press.

Amazon. (2011). "Amazon Elastic Computing Cloud (Amazon EC2)." Retrieved August 2011, from http://aws.amazon.com/ec2/.

Ataman, K., N. Street, et al. (2006). Learning to rank by maximizing auc with linear programming. Proceedings of the IEEE International Joint Conference on Neural Networks.

Atkins, D. E., K. K. Droegemeier, et al. (2003). Revolutionizing Science and Engineering Through Cyberinfrastructure: Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure, National Science Foundation

Bagnasco, S. and et al. (2008). "AliEn: ALICE environment on the GRID." Journal of Physics: Conference Series **119**(6): 062012.

Balakumaran, T., I. L. A. Vennila, et al. (2010). "Microcalcification detection in digital mammograms using novel filter bank." Procedia Computer Science **2**: 272-282.

Bandos, A. I., H. E. Rockette, et al. (2009). "Area under the Free-Response ROC Curve (FROC) and a Related Summary Index." Biometrics **65(1)**: 247-256.

Barbera, R., R. Ramos-Pollan, et al. (2009). gLibrary/DRI: A Grid-Based Platform to Host Muliple Repositories for Digital Content. Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine, and Healthcare. M. Cannataro, IGI Global. **2:** 664-686.

Bellotti, R., P. Cerello, et al. (2007). "Distributed medical images analysis on a Grid infrastructure." Future Generation Computer Systems **23**(3): 475-484.

Bellotti, R., F. De Carlo, et al. (2006). "A completely automated CAD system for mass detection in a large mammographic database." Medical physics **33**(8): 3066-3075.

Blanquer Espert, I., V. Hernández García, et al. (2009). "Content-based organisation of virtual repositories of DICOM objects." Future Generation Computer Systems **25**(6): 627-637.

Bose, R. and Ray-Chaudhuri (1960). "On a class of error-correcting binary group codes." Information Control **3**: 68-79.

Boyer, B., C. Balleyguier, et al. (2009). "CAD in questions/answers: Review of the literature." European Journal of Radiology **69**(1): 24-33.

Bradley, A. P. (1997). "The use of the area under the roc curve in the evaluation of machine learning algorithms." Pattern Recognition **30**(7): 1145-1159.

Brefeld, U. and T. Scheffer (2005). AUC Maximizing Support Vector Learning. In Proc. ICML workshop on ROC Analysis in Machine Learning.

Brown, J., S. Bryan, et al. (1996). "Mammography screening: an incremental cost effectiveness analysis of double versus single reading of mammograms." BMJ (Clinical research ed.) **312**(7034): 809-812.

Butler, S. M., G. I. Webb, et al. (2003). A Case Study in Feature Invention for Breast Cancer Diagnosis Using X-Ray Scatter Images. AI 2003: Advances in Artificial Intelligence. T. D. Gedeon and L. C. C. Fung, Springer Berlin / Heidelberg. **2903:** 677-685.

Calanducci, A., R. Ramos-Pollan, et al. (2008). Enabling Digital Repositories on the Grid. Advanced Engineering Computing and Applications in Sciences, 2008. ADVCOMP '08. The Second International Conference on.

Calders, T. and S. Jaroszewicz (2007). "Efficient AUC optimization for classification." Knowledge Discovery in Databases: PKDD 2007, Proceedings **4702**: 42-53.

Campanini, R., D Dongiovanni, et al. (2004). "A novel featureless approach to mass detection in digital mammograms based on support vector machines." Physics in Medicine and Biology **49**(6): 961.

Campbell, C. and Y. Ying (2011). Learning with Support Vector Machines, Morgan & Claypool.

Caruana, R. and A. Mizil (2006). An empirical comparison of supervised learning algorithms. ICML '06: Proceedings of the 23rd international conference on Machine learning, Pittsburgh, Pennsylvania, ACM.

Casaseca-de-la-Higuera, P., J. I. Arribas, et al. (2005). A Comparative Study on Microcalcification Detection Methods with Posterior Probability Estimation based on Gaussian Mixture Models. Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the.

Castro, C. L. and A. P. Braga (2008). Optimization of the Area under the ROC Curve. Neural Networks, 2008. SBRN '08. 10th Brazilian Symposium on.

CCIF. (2011). "The Cloud Computing Interoperability Forum." Retrieved August 2011, from http://www.cloudforum.org/.

Centor, R. M. and J. S. Schwartz (1985). "An evaluation of methods for estimating the area under the Receiver Operating Characteristic (ROC) curve." Medical Decision Making **5**: 149-156.

Ciatto, S., N. Houssami, et al. (2007). "Computer-Aided Screening Mammography." N Engl J Med **357**(1): 83-85.

Clémençon, S., N. Vayatis, et al. (2009). AUC optimization and the two-sample problem. Neural Information Processing Systems Conference, Vancouver, Canada.

Cootes, T. F., C. J. Taylor, et al. (1995). "Active shape models\—their training and application." Comput. Vis. Image Underst. **61**(1): 38-59.

Cordys. (2011). "CORDYS - Enterprise Cloud Orchestration." Retrieved August 2011, from http://www.cordys.com/.

Cortes, C. and M. Mohri (2004). "AUC optimization vs. error rate minimization." Advances in Neural Information Processing Systems 16 **16**(1621): 313-320.

Cortes, C., M. Mohri, et al. (2007). An alternative ranking problem for search engines. Proceedings of the 6th international conference on Experimental algorithms. Rome, Italy, Springer-Verlag**:** 1-22.

Chang, C.-C. and C.-J. Lin. (2001). "LIBSVM: a library for support vector machines." from http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chatelain, C., S. Adam, et al. (2010). "A multi-model selection framework for unknown and/or evolutive misclassification cost problems." Pattern Recogn. **43**(3): 815-823.

Cheng, H. D., X. Cai, et al. (2003). "Computer-aided detection and classification of microcalcifications in mammograms: a survey." Pattern Recognition **36**(12): 2967-2991.

Cheng, H. D., X. J. Shi, et al. (2006). "Approaches for automated detection and classification of masses in mammograms." Pattern Recognition **39**(4): 646-668.

Chenyang, X. and J. L. Prince (1998). "Snakes, shapes, and gradient vector flow." Image Processing, IEEE Transactions on **7**(3): 359-369.

D'Orsi, C. J., L. W. Bassett, et al. (2003). Breast Imaging Reporting and Data System: ACR BI-RADS-Mammography, American College of Radiology.

Dehghan, F. and H. Abrishami-Moghaddam (2008). Comparison of SVM and neural network classifiers in automatic detection of clustered microcalcifications in digitized mammograms. Machine Learning and Cybernetics, 2008 International Conference on, Kunming, China.

DeRoure, D., M. A. Baker, et al. (2003). The evolution of the grid. Grid Computing: Making the Global Infrastructure a Reality, John Wiley & Sons**:** 65-100.

Dietterich, T. and G. Bakiri (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), Anaheim, CA, AAAI Press.

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. Proceedings of the First International Workshop on Multiple Classifier Systems, Springer-Verlag**:** 1-15.

DIRAC. (2010). "The DIRAC project." from http://lhcbweb.pic.es/DIRAC/.

Domeniconi, C. and B. Yan (2004). Nearest Neighbor Ensemble. Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1 - Volume 01, IEEE Computer Society.

Dorfman, D. D., K. S. Berbaum, et al. (1997). "Proper receiver operating characteristic analysis. The bigamma model." Academic Radiology **4**: 138-149.

Eadie, L. H., P. Taylor, et al. (2011). "A systematic review of computer-assisted diagnosis in diagnostic cancer imaging." European Journal of Radiology **In Press, Corrected Proof**.

Efron, B. (1983). "Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation." Journal of the American Statistical Association **78**(382): 316-331.

Efron, B. and G. Gong (1983). "A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation." The American Statistician **37**(1): 36-48.

Egan, J. P. (1975). Signal detection theory and ROC-analysis Academic Press.

EGI. (2011). "The European Grid Infrastructure." Retrieved August 2011, from http://www.egi.eu.

El-Naqa, I., Y. Yongyi, et al. (2002). "A support vector machine approach for detection of microcalcifications." Medical Imaging, IEEE Transactions on **21**(12): 1552-1563.

Elkan, C. (2001). The Foundations of Cost-Sensitive Learning. IJCAI.

Elter, M., R. Schulz-Wendtland, et al. (2007). "The prediction of breast cancer biopsy outcomes using two CAD approaches that both emphasize an intelligible decision process." Med. Phys. **Nov;34(11)**: 4164-4172.

Eltonsy, N. H., G. D. Tourassi, et al. (2007). "A Concentric Morphology Model for the Detection of Masses in Mammography." Medical Imaging, IEEE Transactions on **26**(6): 880-889.

EMI. (2011). "The European Middleware Initiative." Retrieved August 2011, from http://www.eu-emi.eu.

Eng, J. (2011). "ROC analysis: web-based calculator for ROC curves." Retrieved August 2011, from http://www.jrocfit.org.

Escalera, S., O. Pujol, et al. (2008). Coronary damage classification of patients with the Chagas disease with Error-Correcting Output Codes. Intelligent Systems, 2008. IS '08. 4th International IEEE Conference.

Escalera, S., D. M. J. Tax, et al. (2008). "Subclass Problem-Dependent Design for Error-Correcting Output Codes." Pattern Analysis and Machine Intelligence, IEEE Transactions on **30**(6): 1041-1054.

Esmeir, S. and S. Markovitch (2004). Lookahead-based algorithms for anytime induction of decision trees. Proceedings of the twenty-first international conference on Machine learning, Banff, Alberta, Canada, ACM.

Estrela da Silva, J., J. Marques de Sá, et al. (2000). "Classification of breast tissue by electrical impedance spectroscopy." Medical and Biological Engineering and Computing **38**(1): 26-30.

EUIndiaGrid. (2011). "EU India Grid." Retrieved August 2011, from http://www.euindiagrid.eu/.

Fan, C.-Y., P.-C. Chang, et al. (2011). "A hybrid model combining case-based reasoning and fuzzy decision tree for medical data classification." Applied Soft Computing **11**(1): 632-644.

Faraggi, D. and B. Reiser (2002). "Estimation of the area under the ROC curve." Statistics in Medicine **21**(20): 3093-3106.

Fawcett, T. (2003). ROC Graphs: Notes and Practical Considerations for Researchers. HP Labs Tech Report.

Fawcett, T. (2006). "An introduction to ROC analysis." Pattern Recognition Letters **27**(8): 861-874.

Fawcett, T. and F. Provost (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. Third international conference on knowledge discovery and data mining (KDD-97), Menlo Park, CA.

FECYT (2004). Libro blanco de la eCiencia en España, Fundación Española para la Ciencia y la Tecnología.

Ferri, C., P. A. Flach, et al. (2002). Learning Decision Trees Using the Area Under the ROC Curve. Proceedings of the Nineteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc.**:** 139-146.

Flach, P. A. (2003). The Geometry of ROC Space: Understanding Machine Learning Metrics. Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC.

Foster, I. (2005). Globus Toolkit version 4: Software for service-oriented systems. Proceedings of the IFIP International Conference on Network and Parallel Computing, LNCS 3779, Springer.

Foster, I. and C. Kesselman (2004). The Grid 2, Second Edition: Blueprint for a New Computing Infrastructure, Elsevier.

Foster, I. T. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, Springer-Verlag**:** 1-4.

Frank, A. and A. Asuncion (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA, University of California, School of Information and Computer Science.

Freund, Y. and R. E. Schapire (1995). A decision-theoretic generalization of on-line learning and an application to boosting. Proceedings of the Second European Conference on Computational Learning Theory, Springer-Verlag**:** 23-37.

Fung, G., M. Dundar, et al. (2004). A fast iterative algorithm for fisher discriminant using heterogeneous kernels. Proceedings of the twenty-first international conference on Machine learning, Banff, Alberta, Canada, ACM.

Ghahramani, Z. (2004). Unsupervised learning. Advanced Lectures on Machine Learning.

Giger, M. L. and K. Suzuki (2008). Computer-Aided Diagnosis. Biomedical Information Technology. F. David Dagan. Burlington, Academic Press**:** 359-374.

Glatard, T., X. Zhou, et al. (2009). A comparison between ARC and gLite for medical image processing on Grids MICCAI workshop on HealthGrid, London.

gLite. (2011). "The gLite middleware." Retrieved August 2011, from http://glite.web.cern.ch.

Global-Grid-Forum (2005). Open Grid Services Architecture: Glossary of Terms. GFD-I.044. OGF Document Series, Global Grid Forum

Goddard, M. J. and I. Hinberg (1990). "Receiver operator characteristic (ROC) curves and non-normal data: an empirical study." Statistics in Medicine **9**: 325-227.

Google. (2011). "Google App Engine." Retrieved August 2011, from http://code.google.com/appengine/.

Green, D. and J. Swets (1966). Signal Detection Theory and Psychophysics, John Wiley & Sons, New York.

Gurcan, M. N., L. E. Boucheron, et al. (2009). "Histopathological Image Analysis: A Review." Biomedical Engineering, IEEE Reviews in **2**: 147-171.

Gurcan, M. N., Y. Yardimci, et al. (1997). "Detection of microcalcifications in mammograms using higher order statistics." Signal Processing Letters, IEEE **4**(8): 213-216.

H2. (2011). "The H2 Database Engine." Retrieved August 2011, from http://www.h2database.com.

Hajian-Tilaki, K. O. and J. A. Hanley (2002). "Comparison of Three Methods for Estimating the Standard Error of the Area under the Curve in ROC Analysis of Quantitative Data." Academic Radiology **9**(11): 1278-1285.

Hanczar, B., J. P. Hua, et al. (2010). "Small-sample precision of ROC-related estimates." Bioinformatics **26**(6): 822-830.

Hand, D. J. (1998). Consumer credit and statistics. Statistics in Finance. D. J. Hand and S. D. Jacka, Edward Arnold**:** 69-82.

Hanley, J. A. (1996). "The use of the 'Binormal' model for parametric ROC analysis of quantitative diagnostic tests." Statistics in Medicine **15**(14): 1575-1585.

Hanley, J. A. and B. J. McNeil (1982). "The meaning and use of the area under a receiver operating characteristic (ROC) curve." Radiology **143**(1): 29-36.

Haralick, R., Dinstein, et al. (1973). "Textural features for image classification." IEEE Transactions on Systems, Man, and Cybernetics **SMC-3**: 610-621.

Hastie, T., R. Tibshirani, et al. (2009). The Elements of Statistical Learning, Springer.

He, H., J. M. Lyness, et al. (2009). "Direct estimation of the area under the receiver operating characteristic curve in the presence of verification bias." Statistics in Medicine **28**(3): 361-376.

Heath, M., K. Bowyer, et al. (2001). The Digital Database for Screening Mammography. Fifth International Workshop on Digital Mammography, Medical Physics Publishing.

Heaton, J. (2010). Programming Neural Networks with Encog 2 in Java, Heaton Research, Inc.

Herschtal, A. and B. Raskutti (2004). Optimising area under the ROC curve using gradient descent. Proceedings of the twenty-first international conference on Machine learning. Banff, Alberta, Canada, ACM**:** 49.

Hey, T. and G. Fox (2005). "Special Issue: Grids and Web Services for e-Science: Editorials." Concurr. Comput. : Pract. Exper. **17**(2-4): 317-322.

Hocquenghen, A. (1959). "Codes correcteurs d'erreurs." Chiffres **2**: 147-156.

Hsieh, F. and B. W. Turnbull (1996). "Nonparametric and semiparametric estimation of the receiver operating characteristic curve." Annals of Statistics **24**: 25-40.

Huang, J. and C. X. Ling (2005). "Using AUC and Accuracy in Evaluating Learning Algorithms." IEEE Trans. on Knowl. and Data Eng. **17**(3): 299-310.

Hui, L., D. Groep, et al. (2006). Job Failure Analysis and Its Implications in a Large-Scale Production Grid. e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference on.

Huiqun, D., G. Stathopoulos, et al. (2009). Error-Correcting Output Coding for the Convolutional Neural Network for Optical Character Recognition. Document Analysis and Recognition, 2009. ICDAR '09. 10th International Conference on.

Iavindrasana, J., G. Cohen, et al. (2009). "Clinical data mining: a review." Yearb Med Inform: 121-133.

IBM. (2011). "IBM Cloud Computing." Retrieved August 2011, from http://www.ibm.com/cloud.

IGALC. (2011). "Iniciativa Grid de América Latina - Caribe." Retrieved August 2011, from www.igalc.org.

InsightCorp. (2011). "Grid Computing: A Vertical Market Perspective 2006-2011." Press Release Retrieved August 2011, from http://www.insight-corp.com/reports/grid06.asp.

J. Suckling, S. Astley, et al. (1994). *"The Mammographic Image Analysis Society Digital Mammogram Database".* , Exerpta Medica. International Congress Series **1069:** 375-378.

JAAS. (2011). "The Java Authentication and Authorization Service." from http://java.sun.com/products/jaas/.

Jain, A. K., R. P. W. Duin, et al. (2000). "Statistical pattern recognition: a review." Pattern Analysis and Machine Intelligence, IEEE Transactions on **22**(1): 4-37.

Jensen, K., H. H. Müller, et al. (2000). "Regional confidence bands for ROC curves." Statistics in Medicine **19**(4): 493-509.

Jetty. (2011). "The JETTY web container." from XXXX ADD URL XXXX.

Jiang, J., P. Trundle, et al. (2010). "Medical image analysis with artificial neural networks." Computerized Medical Imaging and Graphics **34**(8): 617-631.

Jiang, J., B. Yao, et al. (2007). "A genetic algorithm design for microcalcification detection and classification in digital mammograms." Computerized medical imaging and graphics : the official journal of the Computerized Medical Imaging Society **31**(1): 49-61.

Jiang, Y. and Z.-H. Zhou (2004). "Editing training data for knn classifiers with neural network ensemble." Lecture Notes in Computer Science **3173**: 356-361.

JMS. (2011). "The Java Message Service (JMS)." Retrieved August 2011, from http://java.sun.com/products/jms/.

Joachims, T. (2005). A Support Vector Method for Multivariate Performance Measures. Proceedings of the International Conference on Machine Learning (ICML).

JWS. (2011). "Java Web Start (JWS)." Retrieved August 2011, from http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp.

Kacsuk, P. (2007). Extending the Services and Sites of Production Grids by the Support of Advanced Portals Proceedings of High Performance Computing for Computational Science - VECPAR 2006, Rio de Janeiro, Brazil, Springer Berlin.

Kang, H., J. Ro, et al. (2009). "Detection of Microcalcifications in Digital Mammograms Using Foveal Method." J Korean Soc Med Inform **Mar 15**(1): 165-172.

Karssemeijer, N. (1993). Adaptive noise equalization and image analysis in mammography. Information Processing in Medical Imaging. H. Barrett and A. Gmitro, Springer Berlin / Heidelberg. **687:** 472-486.

Kim, J.-H. (2009). "Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap." Computational Statistics & Data Analysis.

Kom, G., A. Tiedeu, et al. (2007). "Automated detection of masses in mammograms by local adaptive thresholding." Computers in Biology and Medicine **37**(1): 37-48.

Komori, O. and S. Eguchi (2010). "A boosting method for maximizing the partial area under the ROC curve." Bmc Bioinformatics **11**(1): 314.

Kopans, D. B. (2006). Breast Imaging, Lippincott Williams & Wilkins

Kotsiantis, S., I. Zaharakis, et al. (2006). "Machine learning: a review of classification and combining techniques." Artificial Intelligence Review **26**(3): 159-190.

Kowalik, A. and E. Konstanty (2010). "Basic tests in mammography as a tool in quality improvement." Reports of Practical Oncology & Radiotherapy **15**(5): 145-152.

Krupinski, E. A. (2008). Proceedings of the 9th international workshop on Digital Mammography, Tucson, AZ, USA, Springer-Verlag.

Lachiche, N. and P. Flach (2003). Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. Proceedings of the 20th International Conference on Machine Learning (ICML'03).

Lemaur, G., K. Drouiche, et al. (2003). "Highly regular wavelets for the detection of clustered microcalcifications in mammograms." Medical Imaging, IEEE Transactions on **22**(3): 393-401.

Li, J. and L. Wong (2003). Using Rules to Analyse Bio-medical Data: A Comparison between C4.5 and PCL. Advances in Web-Age Information Management, Springer Berlin / Heidelberg. **2762:** 254-265.

Liang, J., T. McInerney, et al. (2006). "United Snakes." Medical Image Analysis **10**(2): 215-233.

Ling, C., J. Huang, et al. (2003). AUC: A Better Measure than Accuracy in Comparing Learning Algorithms. Advances in Artificial Intelligence. Y. Xiang and B. Chaib-draa, Springer Berlin / Heidelberg. **2671:** 991-991.

Ling, C., J. Huang, et al. (2003). AUC: a Statistically Consistent and more Discriminating Measure than Accuracy. IJCAI.

Lobo, J., A. Jiménez-Valverde, et al. (2008). "AUC: a misleading measure of the performance of predictive distribution models." Global Ecology and Biogeography **17**(2): 145-151.

Long, P. and R. Servedio (2007). Boosting the Area Under the ROC Curve. Proceedings of the 20th Conference on Neural Information Processing Systems (NIPS).

López, Y., A. Novoa, et al. (2008). Computer Aided Diagnosis System to Detect Breast Cancer Pathological Lesions. Progress in Pattern Recognition, Image Analysis and Applications, Springer Berlin / Heidelberg. **Volume 5197/2008:** 453-460.

López, Y., A. Novoa, et al. (2008). Breast Cancer Diagnosis Based on a Suitable Combination of Deformable Models and Artificial Neural Networks Techniques. Progress in Pattern Recognition, Image Analysis and Applications, Springer Berlin / Heidelberg. **Volume 4756/2008:** 803-811.

Lorena, A. C., A. de Carvalho, et al. (2008). "A review on the combination of binary classifiers in multiclass problems." Artificial Intelligence Review **30**(1-4): 19-37.

Lloyd, C. J. (1998). "Using smoothed receiver operating characteristic curves to summarize and compare diagnostic systems." Journal of the American Statistical Association **93**: 1356-1364.

Lloyd, L. J. and Z. Yong (1999). "Kernel estimators of the ROC curves are better than empirical." Statistics & Probability Letters **44**: 221-228.

Macskassy, S. A., F. Provost, et al. (2005). ROC confidence bands: an empirical evaluation. Proceedings of the 22nd international conference on Machine learning. Bonn, Germany, ACM**:** 537-544.

Maheshwari, K., P. Missier, et al. (2009). Medical Image Processing Workflow Support on the EGEE Grid with Taverna. Intl Symposium on Computer Based Medical Systems (CBMS'09), Alburquerque, New Mexico, USA.

Mann, H. and D. Whitney (1947). "On a test of whether one of two random variables is stochastically larger than the other." Annals of Mathematical Statistics **18**: 50-60.

Manning, C. D., P. Raghavan, et al. (2008). Introduction to Information Retrieval,, Cambridge University Press.

Mark Hall, Eibe Frank, et al. (2009). "The WEKA Data Mining Software: An Update." SIGKDD Explorations **11**(1).

Marrocco, C., R. P. W. Duin, et al. (2008). "Maximizing the area under the ROC curve by pairwise feature combination." Pattern Recognition **41**(6): 1961-1974.

Martí, J., A. Oliver, et al. (2010). Proceedings of the 10th International Workshop in Digital Mammography, Girona, Spain, Springer-Verlag.

Marzban, C. (2004). "The ROC Curve and the Area under It as Performance Measures." Weather and Forecasting **19**(6): 1106-1114.

McLoughlin, K. J., P. J. Bones, et al. (2004). "Noise equalization for detection of microcalcification clusters in direct digital mammogram images." Medical Imaging, IEEE Transactions on **23**(3): 313-320.

Mell, P. and T. Grance (2009). The NIST definition of Cloud Computing v15, National Institute for Standards and Technology.

Metz, C. (1978). "Basic Principles of ROC Analysis." Seminars in Nuclear Medicine **8**: 283-298.

Metz, C. (2008). "ROC analysis in medical imaging: a tutorial review of the literature." Radiological Physics and Technology **1**(1): 2-12.

Metz, C. E., B. A. Herman, et al. (1998). "Maximum likelihood estimation of receiver operating characteristic (ROC) curves from continuously-distributed data." Statistics in Medicine **17**: 1033-1053.

Microsoft. (2011). "Microsoft Azure." Retrieved August 2011, from http://www.microsoft.com/windowsazure/.

Mitchell, T. (1997). Machine Learning, McGraw Hill.

Moin, M. S. (2009). Exploring AUC Boosting Approach in Multimodal Biometrics Score Level Fusion.

Moon, W. K., Y.-W. Shen, et al. (2011). "Computer-Aided Diagnosis for the Classification of Breast Masses in Automated Whole Breast Ultrasound Images." Ultrasound in Medicine & Biology **37**(4): 539-548.

Mozer, M. C., R. Dodier, et al. (2001). Prodding the ROC curve: constrained optimization of classifier performance. Proceedings of NIPS'2001.

MPI-Forum (2009). MPI: A Message-Passing Interface Standard, Version 2.2, High Performance Computing Center Stuttgart (HLRS).

MQ. (2011). "The Open Message Queue." Retrieved August 2011, from https://mq.dev.java.net/.

Nakayama, R., Y. Uchiyama, et al. (2006). "Computer-aided diagnosis scheme using a filter bank for detection of microcalcification clusters in mammograms." Biomedical Engineering, IEEE Transactions on **53**(2): 273-283.

NAREGI. (2011). "Japan National Research Grid Initiative." Retrieved August 2011, from http://www.naregi.org.

NASA, NSF, et al. (2008). "WTEC Panel Report, International Assessment of Research and Development in Simulation-Based Engineering and Science." Proceedings of the World Techonology Evaluation Center Workshop, http://www.wtec.org/sbes/.

NEMA. (2008). "The DICOM Standard."

Newman, H. B., M. H. Ellisman, et al. (2003). "Data-intensive e-science frontier research." Commun. ACM **46**(11): 68-77.

Ng KH, M. M. (2003). " Advances in mammography have improved early detection of breast cancer. ." J Hong Kong Coll Radiol **6(3)**: 126-131.

NGIs. (2011). "European National Grid Initiatives." Retrieved August 2011, from http://knowledge.eu-egi.eu/knowledge/index.php/Main_Page.

NIMBUS. (2011). "The NIMBUS Project." Retrieved August 2011, from http://www.nimbusproject.org/.

Nishikawa, R., M. Giger, et al. (1995). "Computer-aided detection of clustered microcalcifications on digital mammograms." Medical and Biological Engineering and Computing **33**(2): 174-178.

Nishikawa, R. M. (2007). "Current status and future directions of computer-aided diagnosis in mammography." Computerized Medical Imaging and Graphics **31**(4-5): 224-235.

NordUGrid. (2011). "NordUGrid." Retrieved August 2011, from http://www.nordugrid.org.

Obuchowski, N. (1998). "Confidence Intervals for the Receiver Operating Characteristic Area in Studies with Small Samples." Academic Radiology **5**: 561-571.

Oliver, A., J. Freixenet, et al. (2010). "A review of automatic mass detection and segmentation in mammographic images." Medical Image Analysis **14**(2): 87-110.

Onega, T., E. J. Aiello Bowles, et al. (2010). "Radiologists' Perceptions of Computer Aided Detection Versus Double Reading for Mammography Interpretation." Academic Radiology **17**(10): 1217-1226.

OpenNebula. (2011). "Open Nebula." Retrieved August 2011, from http://www.opennebula.org/.

Pahikkala, T., A. Airola, et al. (2008). Efficient AUC Maximization with Regularized Least-Squares. Proceeding of the 2008 conference on Tenth Scandinavian Conference on Artificial Intelligence: SCAI 2008, IOS Press**:** 12-19.

Papadopoulos, A., D. I. Fotiadis, et al. (2008). "Improvement of microcalcification cluster detection in mammography utilizing image enhancement techniques." Computers in Biology and Medicine **38**(10): 1045-1055.

Paquerault, S., P. T. Hardy, et al. (2010). "Investigation of Optimal Use of Computer-Aided Detection Systems: The Role of the "Machine" in Decision Making Process." Academic Radiology **17**(9): 1112-1121.

Parameswaran, A. V. and A. Chaddha (2009). "Cloud Interoperability and Standardization." SETLabs Briefings **7**(7): 19-26.

Park, S. C., J. Pu, et al. (2009). "Improving Performance of Computer-aided Detection Scheme by Combining Results From Two Machine Learning Classifiers." Academic Radiology **16**(3): 266-274.

Park, S. H., J. M. Goo, et al. (2004). "Receiver Operating Characteristic (ROC) Curve: Practical Review for Radiologists." Korean Journal of Radiology **5(1)**: 11-18.

Parker, B., S. Gunter, et al. (2007). "Stratification bias in low signal microarray studies." Bmc Bioinformatics **8**(1): 326.

Passerini, A., M. Pontil, et al. (2004). "New results on error correcting output codes of kernel machines." Neural Networks, IEEE Transactions on **15**(1): 45-54.

Pisano, E. D., C. Gatsonis, et al. (2005). "Diagnostic Performance of Digital versus Film Mammography for Breast-Cancer Screening." N Engl J Med **353**(17): 1773-1783.

Provost, F. and T. Fawcett (2001). "Robust Classification for Imprecise Environments." Machine Learning **42**(3): 203-231.

Provost, F. J., T. Fawcett, et al. (1998). The Case against Accuracy Estimation for Comparing Induction Algorithms. Proceedings of the Fifteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc.**:** 445-453.

Qin, J. and B. Zhang (2003). "Using logistic regression procedures for estimating receiver operating characteristic curves." Biometrika **90**: 585-596.

Rakotomamonjy, A. (2004). Optimizing area under roc curve with SVMs. Proceedings of the First Workshop on ROC Analysis and Artificial Intelligence.

Ramos-Pollan, R. and A. Barreiro (2009). Enabling the Grid for experiments in distributed information retrieval. Proceedings of the first EELA-2 Conference, Bogotá, Colombia, CIEMAT pubs, pp 21-30.

Ramos-Pollan, R., J. P. G. Callejon, et al. (2007). Management of a grid infrastructure in GLITE with Virtualization. Santiago Compostella, Cesga-Centro Supercomputacion Galicia.

Ramos-Pollan, R., J. M. Franco, et al. (2010). Grid Computing for Breast Cancer CAD. A Pilot Experience in a Medical Environment. Ibergrid: 4th Iberian Grid Infrastructure Conference Proceedings. A. Proenca, A. Pina, J. G. Tobio and L. Ribeiro, Netbiblo**:** 307-318.

Ramos-Pollan, R., J. M. Franco, et al. (2010). "Grid infrastructures for developing mammography CAD systems." Conf Proc IEEE Eng Med Biol Soc **1**: 3467-3470.

Ramos-Pollan, R., M. Guevara-López, et al. (2011). "A Software Framework for Building Biomedical Machine Learning Classifiers through Grid Computing Resources." Journal of Medical Systems: 1-13.

Ramos-Pollan, R., M. Guevara-López, et al. (2011). "Discovering Mammography-based Machine Learning Classifiers for Breast Cancer Diagnosis." Journal of Medical Systems: 1-11.

Ramos-Pollan, R., M. A. Guevara Lopez, et al. (2010). Introducing ROC curves as error measure functions. A new approach to train ANN-based biomedical data classifiers. 15th Iberoamerican Congress on Pattern Recognition, Sao Paolo, Brasil, Springer.

Ramos-Pollan, R., M. A. Guevara López, et al. (2009). Building Medical Image Repositories and CAD Systems on Grid Infrastructures: A Mammograms Case. 15th edition of the Portuguese Conference on Pattern Recognition., University of Aveiro. Aveiro, Portugal.

Ramos-Pollan, R., M. Rubio del Solar, et al. (2010). Exploiting eInfrastructures for medical image storage and analysis: A Grid application framework for mammography CAD. The Seventh IASTED International Conference on Biomedical Engineering, February 17-19, Innsbruck, Austria. Biomedical Engineering (Vol. 1 and Vol. 2) ISBN: I: 978-0-88986-825-0/II: 978-0-88986-827-4.

Ramos-Pollan, R., M. Rubio del Solar, et al. (2009). Grid-based architecture to host multiple repositories: A mammography image analysis use case. 3rd Iberian Grid Infrastructure Conference Proceedings, Valencia, Spain.

Rangayyan, R. M. (2005). Biomedical Image Analysis. Boca Ratón, FL, USA, CRC Press.

Rangayyan, R. M., F. J. Ayres, et al. (2007). "A review of computer-aided diagnosis of breast cancer: Toward the detection of subtle signs." Journal of the Franklin Institute **344**(3-4): 312-348.

Regentova, E., L. Zhang, et al. (2007). "Microcalcification detection based on wavelet domain hidden Markov tree model: Study for inclusion to computer aided diagnostic prompting system." Medical physics **34**(6): 2206-2219.

Reiser, B. and D. Faraggi (1997). "Confidence intervals for the generalized ROC criterion." Biometrics **53**: 644-652.

Ren, J., D. Wang, et al. (2011). "Effective recognition of MCCs in mammograms using an improved neural classifier." Engineering Applications of Artificial Intelligence **24**(4): 638-645.

Rimal, B. P., C. Eunmi, et al. (2009). A Taxonomy and Survey of Cloud Computing Systems. INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on.

Risk, M., R. Ramos-Pollan, et al. (2009). CardioGrid: a framework for the analysis of cardiological signals in Grid computing. Network Operations and Management Symposium, 2009. LANOMS 2009. Latin American.

Rissanen, J. (1983). "A Universal Prior for Integers and Estimation by Minimum Description Length." Annals of Statistics **11**(2): 416-431.

Robilliard, D., V. Marion-Poty, et al. (2007). An empirical boosting scheme for ROC-based genetic programming classifiers. Proceedings of the 10th European conference on Genetic programming. Valencia, Spain, Springer-Verlag**:** 12-22.

Rodenacker, K. (2001). "A Feature Set For Cytometry On Digitized Microscopic Images." Cell. Pathol **25**: 1-36.

Rosset, S. (2004). Model selection via the AUC. Proceedings of the twenty-first international conference on Machine learning. Banff, Alberta, Canada, ACM**:** 89.

Sadaf, A., P. Crystal, et al. (2011). "Performance of computer-aided detection applied to full-field digital mammography in detection of breast cancers." European Journal of Radiology **77**(3): 457-461.

Sanchez Gómez, S., M. Torres Tabanera, et al. (2010). "Impact of a CAD system in a screen-film mammography screening program: A prospective study." European Journal of Radiology **In Press, Corrected Proof**.

Schwarz, G. (1978). "Estimating the Dimension of a Model." The Annals of Statistics **6**(2): 461-464.

Schwiegelshohn, U., R. Badia, et al. (2009). Perspectives on Grid Computing. Dagstuhl Seminar Proceedings, Leibniz.

Sebban, M., R. Nock, et al. (2003). "Stopping criterion for boosting based data reduction techniques: from binary to multiclass problem." J. Mach. Learn. Res. **3**: 863-885.

Setiono, R. (2000). "Generating concise and accurate classification rules for breast cancer diagnosis." Artificial Intelligence in Medicine **18**(3): 205-219.

Shapiro, J. H. (1999). "Bounds on the area under the ROC curve." J. Opt. Soc. Am. A **16**(1): 53-57.

Shi, X., H. D. Cheng, et al. (2010). "Detection and classification of masses in breast ultrasound images." Digital Signal Processing **20**(3): 824-836.

Silverman, B. W. (1986). Density Estimation for Statistics and Data Analysis, London: Chapman and Hall.

Singh, S., V. Kumar, et al. (2006). SVM Based System for classification of Microcalcifications in Digital Mammograms. Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE.

Soares, C. (2003). Is the UCI Repository Useful for Data Mining? Progress in Artificial Intelligence, Springer Berlin / Heidelberg. **2902:** 209-223.

Sohns, C., B. C. Angic, et al. (2010). "CAD in full-field digital mammography--influence of reader experience and application of CAD on interpretation of time." Clinical Imaging **34**(6): 418-424.

Soltanian-Zadeh, H., F. Rafiee-Rad, et al. (2004). "Comparison of multiwavelet, wavelet, Haralick, and shape features for microcalcification classification in mammograms." Pattern Recognition **37**(10): 1973-1986.

Song, J. H., S. S. Venkatesh, et al. (2005). "Comparative analysis of logistic regression and artificial neural network for computer-aided diagnosis of breast masses." Academic Radiology **12**(4): 487-495.

Songyang, Y. and G. Ling (2000). "A CAD system for the automatic detection of clustered microcalcifications in digitized mammogram films." Medical Imaging, IEEE Transactions on **19**(2): 115-126.

Sorribas, A., J. March, et al. (2002). "A new parametric method based on S-distributions for computing receiver operating characteristic curves for continuous diagnostic tests." Statistics in Medicine **21**(9): 1213-1235.

Spackman, K. A. (1989). Signal detection theory: valuable tools for evaluating inductive learning. Proceedings of the sixth international workshop on Machine learning. Ithaca, New York, United States, Morgan Kaufmann Publishers Inc.**:** 160-163.

Steck, H. (2007). Hinge Rank Loss and the Area Under the ROC Curve. Machine Learning: ECML 2007. J. Kok, J. Koronacki, R. Mantaraset al, Springer Berlin / Heidelberg. **4701:** 347-358.

Stine, R. A. and J. F. Heyse (2001). "Non parametric estimates of overlap." Statistics in Medicine **20**(215-236).

Sweilam, N. H., A. A. Tharwat, et al. (2010). "Support vector machine for diagnosis cancer disease: A comparative study." Egyptian Informatics Journal **11**(2): 81-92.

Swets, J. (1988). "Measuring the accuracy of diagnostic systems." Science **240**(4857): 1285-1293.

Swets, J. A., R. M. Dawes, et al. (2000). "Better decisions through science." Scientific American **283**(4): 82-87.

Takenouchi, T. and S. Eguchi (2009). "Extension of Roc Curve." 2009 IEEE International Workshop on Machine Learning for Signal Processing: 434-439.

The, J. S., K. J. Schilling, et al. (2009). "Detection of breast cancer with full-field digital mammography and computer-aided detection." American Journal of Roentgenology **192**(2): 337-340.

Toh, K. A., J. Kim, et al. (2008). "Maximizing area under ROC curve for biometric scores fusion." Pattern Recognition **41**(11): 3373-3392.

Top500. (2011). "The TOP 500 list of Supercomputing Sites." Retrieved August 2011, from http://www.top500.org.

Tristan, G., L. Diane, et al. (2007). Impact of the execution context on Grid job performances. International Workshop on Context-Awareness and Mobility in Computing (WCAMG'07), Rio de Janeiro, Brazil, IEEE.

Tzikopoulos, S. D., M. E. Mavroforakis, et al. (2011). "A fully automated scheme for mammographic segmentation and classification based on breast density and asymmetry." Computer Methods and Programs in Biomedicine **102**(1): 47-63.

UNICORE. (2011). "The UNICORE Middleware." Retrieved August 2011, from http://www.unicore.eu.

UnifiedCloud. (2011). "The Unified Cloud Interface." Retrieved August 2011, from http://code.google.com/p/unifiedcloud/wiki/UCI_Requirements.

Urbanowicz, R. J. and J. H. Moore (2009). "Learning classifier systems: a complete introduction, review, and roadmap." J. Artif. Evol. App. **2009**: 1-25.

Vanderlooy, S. and E. Hullermeier (2008). "A critical analysis of variants of the AUC." Mach. Learn. **72**(3): 247-262.

Vanderlooy, S., I. Sprinkhuizen-Kuyper, et al. (2006). An analysis of reliable classifiers through ROC isometrics. Proceedings of the ICML 2006 Workshop on ROC Analysis (ROCML 2006), Pittsburg, USA.

Vanderlooy, S., I. G. Sprinkhuizen-Kuyper, et al. (2009). "The ROC isometrics approach to construct reliable classifiers." Intell. Data Anal. **13**(1): 3-37.

Vapnik, V. (1998). Statistical Learning Theory, Wiley.

Vapnik, V. and A. Chervonenkis (1974). Theory of pattern recognition: Statistical problems of learning, Moscow, Nauka.

Velikova, M., M. Samulski, et al. (2009). "Improved mammographic CAD performance using multi-view information: a Bayesian network framework." Phys Med Biol **54**(5): 1131-1147.

Vergara, I. A., T. Norambuena, et al. (2008). "StAR: a simple tool for the statistical comparison of ROC curves." Bmc Bioinformatics **9**: -.

Vlachos, M., C. Domeniconi, et al. (2002). Non-linear dimensionality reduction techniques for classification and visualization. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, Edmonton, Alberta, Canada, ACM.

Waegeman, W., B. De Baets, et al. (2008). "ROC analysis in ordinal regression learning." Pattern Recognition Letters **29**(1): 1-9.

Wang, R. and K. Tang (2009). Feature Selection for Maximizing the Area Under the ROC Curve. Proceedings IEEE International Conference on Data Mining Workshops.

Wang, X., D. Lederman, et al. (2010). "Computerized Detection of Breast Tissue Asymmetry Depicted on Bilateral Mammograms: A Preliminary Study of Breast Risk Stratification." Academic Radiology **17**(10): 1234-1241.

Wang, Y., S. Jiang, et al. (2010). "CAD Algorithms for Solid Breast Masses Discrimination: Evaluation of the Accuracy and Interobserver Variability." Ultrasound in Medicine & Biology **36**(8): 1273-1281.

Wei, L., Y. Yang, et al. (2005). "A study on several Machine-learning methods for classification of Malignant and benign clustered microcalcifications." Medical Imaging, IEEE Transactions on **24**(3): 371-380.

Wei, L., Y. Yongyi, et al. (2005). Relevance vector machine learning for detection of microcalcifications in mammograms. Image Processing, 2005. ICIP 2005. IEEE International Conference on.

Wei, Q., M. Fei, et al. (2002). "An improved method of region grouping for microcalcification detection in digital mammograms." Computerized medical imaging and graphics : the official journal of the Computerized Medical Imaging Society **26**(6): 361-368.

Wilcoxon, F. (1945). "Individual comparisons by ranking methods." Biometrics **1**: 80-83.

Wilson, D. R. and T. R. Martinez (1997). Improved Center Point Selection for Probabilistic Neural Networks. Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, (ICANNGA'97).

Wu, S. and P. Flach (2005). A scored AUC metric for classifier evaluation and selection. Proceedings of the ICML 2005 Workshop on ROC Analysis in Machine Learning, Bonn, Germany.

Wu, S., P. Flach, et al. (2007). An Improved Model Selection Heuristic for AUC. Proceedings of the 18th European Conference on Machine Learning.

Yan, L., R. Dodier, et al. (2003). Optimizing Classifier Performance Via the Wilcoxon-Mann-Whitney Statistics. Proceedins of the

Yongyi, Y., W. Liyang, et al. (2007). Microcalcification Classification Assisted by Content-Based Image Retrieval for Breast Cancer Diagnosis. Image Processing, 2007. ICIP 2007. IEEE International Conference on, San Antonio, TX.

Yoon, H. J. (2007). "Evaluating computer-aided detection algoriths." Medical Physics **34**: 2024-2038.

Yu, S.-N. and Y.-K. Huang (2010). "Detection of microcalcifications in digital mammograms using combined model-based and statistical textural features." Expert Systems with Applications **37**(7): 5461-5469.

Yu, S.-N., K.-Y. Li, et al. (2006). "Detection of microcalcifications in digital mammograms using wavelet filter and Markov random field model." Computerized Medical Imaging and Graphics **30**(3): 163-173.

Zou, K. H. and W. J. Hall (2000). "Two transformation models for estimating a ROC curve derived from continuous data." Journal of Applied Statistics **27**: 621-631.

Zou, K. H., W. L. Hall, et al. (1997). "Smooth non-parametric receiver-operating characteristic (ROC) curves for continuous diagnostic tests." Statistics in Medicine **16**: 2143-2156.

Zou, K. H., C. M. Tempany, et al. (1998). "Original smooth receiver operating characteristic curves exstimation from continuous data: statistical methods for analyzing the predictive value of spiral CT of ureteral stones." Academic Radiology **5**: 680-687.

Zou, K. H., S. K. Warfield, et al. (2004). "Statistical validation of image segmentation quality based on a spatial overlap index." Academic Radiology **11**: 178-189.

Zweig, M. and G. Campbell (1993). "Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine " Clinical Chemistry **39**: 561-577.

Appendix I. Tables and figures

Table 23: Definitions for binary classifiers

$\mathcal{X} \subset \mathbb{R}^p$	Domain of elements consisting of input vectors (with p features)
$\mathcal{Y} = \{P, N\}$	The two classes into which input vectors are classified
$(x, y), x \in \mathcal{X}, y \in \mathcal{Y}$	Element with its associated class (for supervised training)
$SD = \{(x_1, y_1), \ldots, (x_n, y_n)\}$	Dataset (for supervised training)
$S_P = \{x \mid (x, P) \in SD\}$	Positive elements of dataset
$S_N = \{x \mid (x, N) \in SD\}$	Negative elements of dataset
$S = S_P \cup S_N$	Elements of a dataset
$S(x) = y$	Class associated to element x through dataset S
$\lvert S \rvert = n$	Size of dataset
$\mathcal{F} = \{ f: \mathcal{X} \to \mathcal{Y} \}$	Set of functions representing binary classifiers
$h \in \mathcal{F}$	A binary classifier
$h(x) \in \mathcal{Y}$	Output of binary classifier h when applied to element x
$h_{sc}(x) \in \mathbb{R}$	Score assigned by binary classifier h to element x it is typically used by h to determine $h(x)$ by applying some threshold, and obtain ROC curves
$\mathcal{E}(S, h)$	A global error measure of classifier h when applied to training set S
$e(x, h)$	An individual error measure of classifier h when applied to element x
$AUC(S, h)$	Area under the ROC curve (AUC) of dataset S when classified with classifier h
$\mathcal{O} = [nn_{min}, nn_{max}] \subset \mathbb{R}$	Range of output values for the two output neurons of a multilayer perceptron based binary classifier (binary MLP)
$h_P(x), h_N(x) \in \mathcal{O}$	Output of the positive and negative neurons of the binary MLP h upon element x
$d_P(x), d_N(x) \in \mathcal{O}$	Ideal value for positive and negative neurons for x
$e_P(x, h), e_N(x, h)$	Error measures for the output of the positive and negative neurons of binary MLP h upon element x

Table 24: UCI datasets used in this theseis

name	description	# elements	# features	% positive
bcw	Cell based metricsi for breast cancer	699	9	35%
bcwd	Diagnostic breast cancer Wisconsin database	569	30	63%
btcat	Carcinoma in breast tissue	106	9	34%
echocard	Data for classifying if patients will survive for at least one year after a heart attack	131	8	33%
glass	From USA Forensic Science Service; 6 types of glass; defined in terms of their oxide content (i.e. Na, Fe, K, etc.) (*merged into two classes*)	214	9	76%
haber	Survival of patients who had undergone surgery for breast cancer	306	3	26%
heartsl	Patient indicators for presence of heart disease	270	13	44%
liver	BUPA Medical Research Ltd. database on liver desease	345	6	58%
magic	MAGIC Gamma Telescope Data Set	19020	11	65%
mmass	Benign/malignant mammographic masses based on BIRADS attributes and the patient's age	961	5	54%
park	Oxford Parkinson's Disease Detection Dataset	195	22	75%
pgene	E. Coli promoter gene sequences (DNA) with partial domain theory	106	57	50%
pimadiab	Patient indicators for presence of diabetes	768	8	35%
spam	Classifying email as spam or not spam	4601	57	39%
spectf	Data on cardiac Single Proton Emission Computed Tomography (SPECT) images	267	44	21%
statlog.land	Statlog (Landsat Satellite) Data Set (*first three classes against the rest*)	6435	36	56%
statlog.shuttle	Statlog (Shuttle) Data Set (*first class against the rest*)	58000	9	80%
tictac	Binary classification task on possible configurations of tic-tac-toe game	958	9	65%
yeast	Yeast Protein Localization Sites (*first class against the rest*)	1484	8	31%
waveform	Waveform Database Generator (*first two classes merged*)	5000	21	34%

Table 25: AUC performance of MLP algorithms modified for AUC optimization.

	FFBP				FFRP				FFSA				FFGA			
	ORIGINAL		MODIFIED		ORIGINAL		MODIFIED		ORIGINAL		MODIFIED		ORIGINAL		MODIFIED	
	avg	stddev	avg	stddev		avg	stddev		avg	stddev	avg	stddev		avg	stddev	
				improv	avg	stddev	avg	stddev	improv	avg	stddev	avg	stddev	improv	avg	stddev	improv
pgene	0,6928	0,163	0,7009	0,181	0,81%	0,8794	0,034	0,8293	0,086	-5,70%	0,7891	0,084	0,7595	0,103	-3,76%	0,6085	0,118	0,6366	0,119	4,61%
mmass	0,5679	0,235	0,6541	0,216	15,17%	0,8489	0,034	0,9108	0,023	7,30%	0,8873	0,043	0,8850	0,033	-0,26%	0,8230	0,039	0,8592	0,034	4,39%
heartsl	0,6283	0,233	0,7161	0,180	13,98%	0,8533	0,045	0,9111	0,016	6,77%	0,8797	0,078	0,8932	0,054	1,53%	0,8188	0,100	0,8960	0,040	9,44%
liver	0,5371	0,131	0,6526	0,137	21,50%	0,6729	0,082	0,7155	0,098	6,32%	0,7252	0,077	0,7767	0,096	7,10%	0,5553	0,110	0,6853	0,071	23,42%
bcwd	0,6734	0,225	0,6629	0,261	-1,56%	0,9904	0,006	0,9894	0,004	-0,10%	0,9887	0,005	0,9928	0,008	0,41%	0,9494	0,036	0,9689	0,019	2,05%
pimadiab	0,5251	0,131	0,5343	0,151	1,76%	0,7449	0,057	0,7961	0,058	6,87%	0,8303	0,034	0,8311	0,050	0,10%	0,7190	0,084	0,8056	0,051	12,04%
tictac	0,6309	0,140	0,6988	0,147	10,76%	0,8639	0,029	0,8897	0,030	2,99%	0,8105	0,047	0,8400	0,071	3,64%	0,6295	0,092	0,7168	0,088	13,86%
echocard	0,4980	0,152	0,5820	0,135	16,86%	0,5161	0,081	0,6299	0,161	22,05%	0,6091	0,134	0,6497	0,178	6,66%	0,6439	0,137	0,6341	0,168	-1,52%
haber	0,5723	0,121	0,5507	0,139	-3,76%	0,5881	0,038	0,7200	0,152	22,42%	0,6970	0,105	0,7268	0,104	4,28%	0,6599	0,123	0,7013	0,104	6,28%
park	0,6774	0,220	0,7115	0,233	5,03%	0,9244	0,038	0,8979	0,049	-2,87%	0,9373	0,084	0,9175	0,065	-2,12%	0,8225	0,115	0,8608	0,097	4,66%
glass	0,6655	0,318	0,7235	0,258	8,72%	0,9259	0,051	0,9535	0,073	2,98%	0,9510	0,061	0,9545	0,098	0,37%	0,9620	0,043	0,9673	0,043	0,55%
spectf	0,4542	0,120	0,4922	0,129	8,37%	0,7783	0,064	0,8295	0,086	6,58%	0,7987	0,093	0,8039	0,098	0,65%	0,6628	0,152	0,7251	0,108	9,40%
averages	0,5936	0,182	0,6400	0,180	8,14%	0,7989	0,047	0,8394	0,070	6,30%	0,8253	0,070	0,8359	0,080	1,55%	0,7379	0,096	0,7881	0,078	7,43%

Table 26: AUC performace in dataset and element based MLP algorithms

	OVERALL AUC			**ELEMENT BASED AUC (*ffbp* and *ffrp*)**			**DATASET BASED AUC (*ffsa* and *ffga*)**		
	ORIGINAL	MODIFIED	improv	ORIGINAL	MODIFIED	improv	ORIGINAL	MODIFIED	improv
pgene	0,7425	0,7316	-1,01%	0,7861	0,7651	-2,44%	0,6988	0,6980	0,43%
mmass	0,7818	0,8273	6,65%	0,7084	0,7825	11,23%	0,8552	0,8721	2,07%
heartsl	0,7950	0,8541	7,93%	0,7408	0,8136	10,37%	0,8492	0,8946	5,48%
liver	0,6226	0,7075	14,59%	0,6050	0,6840	13,91%	0,6402	0,7310	15,26%
bcwd	0,9005	0,9035	0,20%	0,8319	0,8261	-0,83%	0,9691	0,9809	1,23%
pimadiab	0,7048	0,7418	5,19%	0,6350	0,6652	4,32%	0,7747	0,8184	6,07%
tictac	0,7337	0,7863	7,81%	0,7474	0,7943	6,87%	0,7200	0,7784	8,75%
echocard	0,5668	0,6239	11,01%	0,5070	0,6059	19,45%	0,6265	0,6419	2,57%
haber	0,6293	0,6747	7,30%	0,5802	0,6354	9,33%	0,6784	0,7141	5,28%
park	0,8404	0,8469	1,18%	0,8009	0,8047	1,08%	0,8799	0,8891	1,27%
glass	0,8761	0,8997	3,15%	0,7957	0,8385	5,85%	0,9565	0,9609	0,46%
spectf	0,6735	0,7127	6,25%	0,6163	0,6609	7,47%	0,7308	0,7645	5,02%
averages	0,7389	0,7758	5,86%	0,6962	0,7397	7,22%	0,7816	0,8120	4,49%

Table 27: Some correlations in AUC optimization experiments

		ffbp	***ffrp***	***ffsa***	***ffga***	**all**
(1)	improvement vs. class skew	-0,2578	0,2591	0,1038	-0,2089	-0,0267
(2)	improvement vs. ORIGINAL AUC avg	-0,3652	-0,8724	-0,6649	-0,5537	-0,7170
(3)	improvement vs. ORIGINAL AUC stddev	0,0635	0,4545	0,3624	0,2087	0,1783
(4)	ORIGINAL AUC stddev vs. MODIFIED AUC stddev	0,8834	0,6312	0,8843	0,8286	0,6823
(5)	ORIGINAL AUC avg vs. MODIFIED AUC avg	0,8488	0,9564	0,9816	0,9568	0,9715

Table 28: Summary of BiomedTK commands

command	description
	dataset manipulation commands
dataset	generates biomedtk dataset set from a CSV file, or creates a simmulated one
bindatasets	generates a positive/negative dataset for each class of an existing dataset
arff.export	exports a dataset into a WEKA ARFF file
arff.import	imports a dataset from a WEKA ARFF file
libsvm.export	exports a dataset into a LIBSVM file
split.dataset	splits a dataset into test/train data
cmatrix.datasets	generates binary datasets from a multiclass dataset following a decomposition specified in a code matrix
dataset.metrics	shows dataset metrics
norm.dataset	normalizes a dataset
	classifier evaluation and exploration commands
jobset	generates a jobset to perform an exploration
launch	executes sequentially the jobs of a jobset in this machine
ensembles	creates ensemble classifiers from binary classifiers
	classifier reusage commands
apply	applies to a dataset a classifier obtained from a previous training process
summary	summarizes results stored in the DB for a particular dataset
verify	verifies results stored in DB by reconstructing engines and applying them to datasets
	C3 commands
c3.prepare	prepares an exploration for c3
c3.submit	submits a prepared exploration to c3
c3.launch	prepares and submits an exploration to c3
c3.status	retrieves the status of an exploration submitted to c3
	utility commands
db	start H2 db server
show.classifiers	shows available classifiers and parameters accepted by each classifier

Figure 41: Sample BiomedTK session

Table 29: BiomedTK/C3 experimental exploration summary

engine		bcw	bcwd	btcat	echocard	haber	heartsl	hepat	liver	mmass	park	pgene	pimadiab	spam	spectf	tictac
ffbp	configs	32	20	72	96	128	48	16	48	96	48	18	60	16	12	36
	CPU hours	77.82	102.04	95.96	64.42	36.92	123.30	50.80	46.13	180.65	213.00	164.17	192.80	305.38	64.22	149.01
	best accuracy	86.59%	95.00%	74.40%	68.30%	69.82%	89.52%	88.63%	71.33%	83.73%	93.47%	89.09%	70.50%	78.76%	57.95%	81.10%
	best AUC	0.860	0.943	0.729	0.680	0.717	0.908	0.884	0.695	0.849	0.942	0.889	0.682	0.779	0.598	0.799
ffbproc	configs	32	20	72	96	128	48	16	48	96	48	18	60	16	12	36
	CPU hours	118.55	171.54	128.33	118.53	57.13	211.50	83.97	65.74	293.50	363.29	333.84	313.40	433.12	104.67	241.50
	best pct	90.17%	95.06%	80.40%	77.33%	73.93%	91.41%	89.11%	80.85%	91.74%	98.28%	93.38%	74.24%	82.90%	67.18%	88.06%
	best AUC	0.889	0.958	0.819	0.744	0.718	0.932	0.873	0.817	0.913	0.991	0.918	0.715	0.823	0.646	0.875
ffrp	configs	4	4	4	4	4	4	4	4	4	4	4	2	2	4	4
	CPU hours	8.81	17.78	4.86	2.81	1.13	9.47	9.85	3.48	5.95	15.09	36.49	4.94	30.64	19.13	18.45
	best accuracy	81.41%	95.09%	80.07%	63.80%	62.72%	93.08%	92.33%	75.71%	88.43%	98.29%	94.36%	79.90%	83.63%	90.80%	88.64%
	best AUC	0.804	0.966	0.786	0.614	0.634	0.907	0.939	0.772	0.890	0.970	0.921	0.813	0.809	0.885	0.899
ffrproc	configs	4	4	4	4	4	4	4	4	4	4	4	2	2	4	4
	CPU hours	13.70	31.10	7.04	4.57	1.84	12.78	15.25	4.64	11.60	23.57	65.80	7.93	57.74	34.45	24.60
	best accuracy	88.10%	96.47%	82.29%	81.06%	92.34%	93.12%	93.59%	80.02%	94.96%	98.19%	96.14%	81.73%	85.86%	93.74%	94.09%
	best AUC	0.892	0.938	0.804	0.822	0.902	0.931	0.943	0.813	0.938	0.957	0.932	0.818	0.856	0.933	0.926
ffsa	configs	32	18	60	48	54	36	16	36	48	36	16	48	16	24	32
	CPU hours	116.76	150.39	124.15	51.69	25.50	131.22	68.97	45.26	141.41	247.85	296.47	249.00	430.88	243.26	246.76
	best accuracy	97.14%	92.70%	90.56%	75.93%	82.17%	90.03%	94.69%	80.78%	95.23%	97.32%	88.50%	83.12%	80.42%	90.30%	84.91%
	best AUC	0.933	0.901	0.822	0.770	0.822	0.901	0.937	0.818	0.939	0.967	0.890	0.809	0.793	0.910	0.867
ffsaroc	configs	32	18	60	48	54	36	32	36	48	36	16	48	16	24	32
	CPU hours	131.53	147.65	148.70	71.76	35.25	166.53	197.77	55.38	195.51	328.29	289.83	328.65	599.21	259.80	259.39
	best accuracy	92.75%	95.04%	90.56%	85.90%	84.65%	96.30%	95.33%	81.62%	90.83%	98.03%	87.74%	82.41%	90.81%	90.83%	94.85%
	best AUC	0.923	0.969	0.901	0.863	0.852	0.958	0.955	0.822	0.925	0.996	0.883	0.815	0.902	0.922	0.925
ffga	configs	24	24	36	32	29	36	16	24	36	24	12	36	6	12	24
	CPU hours	104.84	270.42	90.20	52.04	15.94	197.31	103.45	46.91	148.84	233.33	270.20	261.35	249.07	179.40	226.25
	best accuracy	82.26%	92.41%	77.94%	80.55%	82.01%	96.30%	87.63%	67.05%	88.50%	94.40%	73.37%	81.23%	68.69%	85.67%	72.57%
	best AUC	0.865	0.936	0.770	0.808	0.807	0.994	0.877	0.687	0.870	0.960	0.751	0.801	0.698	0.845	0.74
ffgaroc	configs	24	24	36	32	29	36	16	24	36	24	12	36	6	12	24
	CPU hours	121.58	323.06	114.30	55.01	21.01	218.34	113.09	48.85	187.89	252.11	320.13	278.42	281.44	207.38	271.21
	best accuracy	86.77%	96.89%	80.21%	82.57%	81.74%	92.91%	87.19%	76.04%	92.98%	96.52%	80.27%	83.35%	78.68%	86.86%	80.49%
	best AUC	0.854	0.955	0.786	0.835	0.826	0.944	0.875	0.770	0.900	0.977	0.780	0.808	0.766	0.854	0.822
c-svc	configs	48	48	256	256	256	160	128	256	48	256	256	192	48	256	256
	CPU hours	0.92	0.86	0.67	0.83	1.98	1.23	0.53	2.11	1.17	1.54	0.68	3.59	5.01	1.70	7.37
	best accuracy	97.14%	97.65%	90.33%	86.60%	93.75%	94.71%	95.25%	80.26%	92.34%	97.67%	93.16%	82.25%	91.93%	93.38%	95.71%
	best AUC	0.930	0.977	0.815	0.864	0.940	0.931	0.951	0.804	0.939	0.953	0.952	0.810	0.919	0.935	0.931
nu-svc	configs	48	48	256	256	256	160	128	256	48	256	256	192	48	256	256
	CPU hours	0.86	0.83	0.64	2.00	2.18	1.02	0.49	2.57	1.11	1.18	0.73	4.35	4.49	1.95	6.02
	best accuracy	96.91%	96.64%	89.40%	84.70%	93.75%	95.47%	96.44%	81.01%	95.83%	96.94%	95.62%	81.14%	88.55%	93.20%	96.24%
	best AUC	0.924	0.962	0.812	0.857	0.931	0.944	0.957	0.814	0.965	0.962	0.946	0.809	0.894	0.947	0.947
total	configs	280	228	856	872	942	568	376	736	464	736	612	676	176	616	704
total	CPU hours	695.38	1215.67	714.87	423.65	198.89	1072.72	644.16	321.06	1167.61	1679.26	1778.35	1644.43	2396.98	1115.97	1450.55
best AUC	overall	0.933	0.977	0.901	0.864	0.940	0.994	0.957	0.822	0.965	0.996	0.952	0.818	0.919	0.947	0.947
best accuracy	overall	97.14%	97.65%	90.56%	86.60%	93.75%	96.30%	96.44%	81.62%	95.83%	98.29%	96.14%	83.35%	91.93%	93.74%	96.24%
best accuracy	in literature	98.94%	83.83%	86%	90.9%	80.48%	89.5%	93.51%	72.5%	0.890*	95.8%	91.5%	81.83%	93.88%	82.8%	90.9%

* only AUC reported in source

Appendix II. Algorithms

Algorithm 1: Approximate AUC for given interval

Datastruct:

intervalType {

subintervals[]	intervalType
nPositive	number of positive elements within this interval (defaults to 0)
nNegative	number of negative elements within this interval (defaults to 0)
nBelow	number of negative elements in intervals below this one (defaults to 0)
maxScore	maximum score of elements within this interval (defaults to 0)
minScore	minimum score of elements within this interval (defaults to 1)
approxAz	the approximated Az calculation for this interval (defaults to 0)
maxError	the maxError calculation for this interval (defaults to 1)
data[]	elements of the dataset belonging to this interval

}

Inputs:

interval	an intervalType with a list of dataset elements in its *data*[] field.
intervalBegin	lower limit of the interval
intervalEnd	upper limit of the interval
numberOfSubIntervals	number of subintervals into which split the data

Outputs: the following fields of input parameter *interval* filled in: *approxAz, maxError, subintervals* (list of subintervals each one with all fields filled in except the *subintervals* field)

Algorithm:

```
1:  subintervals ← new array of intervals of size numberOfSubIntervals
2:  subintervalSize ← (intervalEnd−intervalBegin) / numberOfSubintervals
3:  numberOfPositiveElements ← 0
4:  numberOfNegativeElements ← 0
5:  for i=1 to data.length
6:      subintervalNumber ← integerPart (intervalBegin + data[i]/subintervalSize) + 1
7:      if data[i] is a positive example then
8:          subintervals[subintervalNumber].nPositive++
9:          numberOfPositiveElements++
10:     else
11:         subintervals[subintervalNumber].nNegative++
12:         numberOfNegativeElements++
13:     end if
14:     if interval.data[i] > subintervals[subintervalNumber].maxScore then
15:         subintervals[subintervalNumber].maxScore ← interval.data[i]
16:     end if
17:     if data[i] < subintervals[subintervalNumber].minScore then
18:         subintervals[subintervalNumber].minScore ← interval.data[i]
19:     end if
20:     add interval.data[i] to subintervals[subintervalNumber].data
21: end for
22: interval.approxAz ← 0
23: interval.maxError ← 0
24: for i=1 to numberOfSubintervals
25:     if i>1 then
26:         subintervals[i].nBelow ← subintervals[i-1].nNegative + subintervals[i-1].nBelow
                                    + interval.nBelow
27:     end if
27:     subintervals[i].maxError ← subintervals[i].nPositive × subintervals[i].nNegative
                                   / numberOfPositiveElements x numberOfNegativeElemetns
28:     subintervals[i].approxAz ← subintervals[i].nPositive × (subintervals[i].nNegative+2×subintervals[i].nBelow)
                                   / 2 ×numberOfPositiveElements ×numberOfNegativeElements
29:     interval.approxAz ← interval.approxAz + subintervals[i].approxAz
30:     interval.maxError ← interval.maxError + subintervals[i].maxError
31: end for
32: interval.subintervals ← subintervals
```

Algorithm 2: Approximate ROC Az for a dataset with a maximum error bound

Inputs:	*elements*	set of elements to calculate ROC Az for
	numberOfSubintervals	number of subintervals for subdividing each interval
	errorBound	maximum error allowed
Outputs:	*approxAz*	the approximated area under the ROC curve
	maxError	the maximum error of the approximation

Algorithm:

```
1:  initialInterval ← create new intervalType
2:  initialInterval.data ← elements
3:  maxError ← 1.0
4:  approxAz ← 0.0
5:  while maxError > errorBound do
6:      nextInterval ← select interval with greatest maxError
7:      call Algorithm 1 with input
8:          interval ← nextInterval
9:          intervalBegin ← nextInterval.minScore
10:         intervalEnd ← nextInterval.maxScore
11:         numberOfSubintervals
12:     end call
13:     update maxError with nextInterval.maxError
14:     update approxAz with nextInterval.approxAz
15: end while
```

Appendix III. ROC definitions

The Wilcoxon-Mann-Whitney statistic for empirical AUC for dataset S and scores assigned by classifier h

$$AUC(S,h) = \frac{\sum_{p\in S_P}\sum_{n\in S_N}\mathbf{1}[h_{sc}(n) < h_{sc}(p)]}{|S_P|\cdot|S_N|}$$

Where $\mathbf{1}[L]$ denotes the *indicator function*, yielding 1 if L is true and 0 otherwise. From here, the following definitions were developed in this thesis:

Definition 1. Contribution of dataset element x to $AUC(S,h)$

$$AUC(x,h) = \begin{cases} \frac{\sum_{n\in S_N}\mathbf{1}[h_{sc}(n) < h_{sc}(x)]}{|S_P|\cdot|S_N|} & if\ x \in S_P \\ \frac{\sum_{p\in S_P}\mathbf{1}[h_{sc}(x) < h_{sc}(p)]}{|S_P|\cdot|S_N|} & if\ x \in S_N \end{cases}$$

Definition 2. Maximum contribution of dataset element x to $AUC(S,h)$

$$AUC_{max}(x,h) = \begin{cases} \frac{|S_N|}{|S_P|\cdot|S_N|} = \frac{1}{|S_P|} & if\ x \in S_P \\ \frac{|S_P|}{|S_P|\cdot|S_N|} = \frac{1}{|S_N|} & if\ x \in S_N \end{cases}$$

Definition 3: AUC error incurred by classifier h when scoring dataset S, with respect to the ideal AUC=1.0

$$\mathcal{E}^{AUC}(S,h) = 1 - AUC(S,h)$$

Definition 4: AUC error incurred by the score assigned to element x by classifier h

$$e^{AUC}(x,h) = 1 - \frac{AUC(x,h)}{AUC_{max}(x,h)}$$

Lemma 1: $\mathcal{E}^{AUC}(S,h)$ is the mean of $e^{AUC}(x,h)$ over the dataset elements.

The proof of this lemma simply develops the summation of $e^{AUC}(x,h)$ over all detaset elements by adding up the positive and negative elements separately as follows:

$$\sum_{x\in S} e^{AUC}(x,h) = \sum_{p\in S_P} e^{AUC}(p,h) + \sum_{n\in S_N} e^{AUC}(n,h)$$

$$= \sum_{p\in S_P}\left(1-\frac{AUC(p,h)}{AUC_{max}(p)}\right)+\sum_{n\in S_N}\left(1-\frac{AUC(n,h)}{AUC_{max}(n)}\right)$$

$$= |S_P| - \sum_{p\in S_P}\left(|S_P|\cdot AUC(p,h)\right) + |S_N| - \sum_{n\in S_N}\left(|S_N|\cdot AUC(n,h)\right)$$

$$= |S_P| - \sum_{p\in S_P}\left(|S_P|\cdot\frac{\sum_{n\in S_N}\mathbf{1}[h_{sc}(n)<h_{sc}(p)]}{|S_P|\cdot|S_N|}\right) + |S_N| - \sum_{n\in S_N}\left(|S_N|\cdot\frac{\sum_{p\in S_P}\mathbf{1}[h_{sc}(n)<h_{sc}(p)]}{|S_P|\cdot|S_N|}\right)$$

$$= |S_P| - |S_P|\cdot\frac{\sum_{p\in S_P}\sum_{n\in S_N}\mathbf{1}[h_{sc}(n)<h_{sc}(p)]}{|S_P|\cdot|S_N|} + |S_N| - |S_N|\cdot\frac{\sum_{n\in S_N}\sum_{p\in S_P}\mathbf{1}[h_{sc}(n)<h_{sc}(p)]}{|S_P|\cdot|S_N|}$$

$$= |S_P| - |S_P|\cdot AUC(S,h) + |S_N| - |S_N|\cdot AUC(S,h)$$

$$= |S_P|\cdot\left(1-AUC(S,h)\right) + |S_N|\cdot\left(1-AUC(S,h)\right)$$

$$= (|S_P|+|S_N|)\cdot\left(1-AUC(S,h)\right) = |S|\cdot\left(1-AUC(S,h)\right) = |S|\cdot\mathcal{E}^{AUC}(S,h)$$

$$\Rightarrow \mathcal{E}^{AUC}(S,h) = \frac{\sum_{x\in S} e^{AUC}(x,h)}{|S|}$$

■