BioDCV: a Distributed Computing System for the Complete Validation of Gene Profiles

Relatore:
Prof. Lorenzo Bruzzone

Correlatore: Laureando:
Dott. Cesare Furlanello Davide Albanese

Anno Accademico 2003–2004
Ai miei genitori
Acknowledgements

Thanks to Dr Cesare Furlanello and to Prof. Lorenzo Bruzzone for giving me the possibility to develop this project and for what they have taught me.

Thanks to the entire group MPBA of ITC-irst, and particularly to Dr Giuseppe Jurman, Dr Stefano Merler and Dr Silvano Paoli for helping with many technical issues.

Thanks also to Roberto Flor and Alessandro Soraruf for discussing and helping with database and clusters.

The support of the Pierotti’s group at INT-IFOM Milano, and particularly of Dr J. Reid, in supplying experimental data is gratefully acknowledged.

Special thanks goes to Manuela for helping me in the writing of this dissertation in English.

Thanks to my parents and to my grandparents for giving me the possibility to achieve this important goal.
# Contents

## Summary

## A System in Functional Genomics

1.1 Introduction ........................................................................................................... 3
   1.1.1 The Selection Bias Problem .......................................................... 3
   1.1.2 A Complete System ................................................................. 5
   1.1.3 Resampling Methods .............................................................. 6

1.2 System Elements .................................................................................................. 8
   1.2.1 A Classifier: Support Vector Machines ....................................... 8
   1.2.2 The Recursive Feature Elimination .......................................... 11
   1.2.3 The Entropy-based Recursive Feature Elimination ................... 12

1.3 System Setup .................................................................................................... 13
   1.3.1 The ONF Procedure .............................................................. 13
   1.3.2 The OFS-M Procedure ......................................................... 14
   1.3.3 The VAL Procedure ............................................................. 15

## BioDCV Components

2.1 Introduction ......................................................................................................... 16

2.2 The pr Package ................................................................................................. 17

2.3 The Database Engine ........................................................................................ 18
   2.3.1 Berkeley DB ............................................................................. 18
   2.3.2 SQLite ....................................................................................... 20

2.4 OpenMosix and the Cluster Facility .................................................................. 21

2.5 GRID Technology ............................................................................................. 23
   2.5.1 EGRID Project .......................................................................... 23

2.6 Future Tasks ....................................................................................................... 24
## Contents

3 BioDCV Design and Implementation 26
  3.1 Introduction .................................. 26
  3.2 System Design .................................. 27
  3.3 Database Design ................................. 28

4 Results 32
  4.1 Data Description ................................. 32
  4.2 Tests on Egrid .................................. 33
    4.2.1 Colon Cancer ................................. 33
    4.2.2 INT-IFOM Data ............................... 35
  4.3 Tests on workstation and on Linux openMosix Clusters . . . 35
Summary

In this dissertation, we describe BioDCV, a distributed computing system for the complete validation of gene profiles. The system is composed of a suite of software modules that allow to define, manage and analyze a complete experiment on DNA microarray data. In particular, the BioDCV system supports the high throughput computing (HTC) needed for building predictive classification models and extracting the most important genes.

The study of gene profiles is expected to enable significant advances in molecular diagnosis and prognosis. Its general aim is the classification of tissues according to their gene expression profiles, highlighting, at the same time, which genes are determining of the underlying biological process. These studies require a careful and computationally demanding methodology. In particular, it is crucial to use a complete validation process to avoid the selection bias problem [Simon et al., 2004]. In our system, complete validation is obtained by an intensive replication of classification-and-ranking steps adopting Support Vector Machines (SVM) and Recursive Feature Elimination (RFE) is considered.

The BioDCV system is totally written in C and embedded with the SQLite database engine library to guarantee speed, slim and robust code with a relational management of system outputs. This choice is specifically suitable for parallel or distributed computing system. BioDCV is designed for portability: i.e. it runs on Linux OpenMosix clusters and Egrid.it testbeds. The system allows a computing near data approach [Chen and Ripley, 2003], in order to increase performance, as data, software and model parameters may be easily distributed through the computing nodes of the testbed. The main engine of BioDCV is the p2r package (http://mpa.itc.it) a C toolbox for solving classification problems. The experimental design proposed is partially similar to those described in [Ambroise and McLachlan, 2002] and in [Furlanello et al., 2003c], with a double loop which devises the feature selection process from the classification accuracy assessment.
In this dissertation, the following issues will be discussed:

- the design and development of the complete validation system realized by BioDCV;
- the classification, feature-selection and resampling methods used;
- the embedded database library features in the classification process;
- the test on several parallel and distributed computing system.

Intensive tests of gene profiling on synthetic and public-domain microarray data were performed, with BioDCV reproducing the experiments described in [Furlanello et al., 2005]. Moreover, tests were performed on new DNA microarray data (Sarcoma database) in collaboration with Istituto Nazionale Tumori and IFOM. Finally, BioDCV has been successfully ported as a computational grid application and run on Egrid.it, a collaboration with INFM-Democritos Trieste [Paoli et al., 2005].

The BioDCV system was developed at MPBA laboratory, SSI division of ITC-irst in Sept 2004 – Jan 2005.
Chapter 1

A Complete Validation System in Functional Genomics

1.1 Introduction

The study of gene expression patterns is expected to enable significant advanced in disease diagnosis and prognosis. Generally, we work trying to classify different tissues according to their gene expression profile, highlighting, at the same time, the most important genes with purpose of classification itself. Therefore, we need:

- a classifier to train the model;
- a ranking method to find the more important genes;

Support Vector Machines (SVMs), which are considered a performing classification method for gene-expression data, were soon embedded with feature selection procedures. A recursive approach like the Recursive Feature Elimination (RFE) procedure is often adopted with SVM [Guyon et al., 2002].

1.1.1 The Selection Bias Problem

As reported in [Furlanello et al., 2003c] and in [Furlanello et al., 2003b], a serious procedural problem affects a number of results in the literature in gene profiling methodology. Initial studies on microarray data proposed classification models defined by very few genes and resulting in negligible or zero error rates. As discussed in [Ambroise and McLachlan, 2002]
and confirmed in [Furlanello et al., 2003c], the problem is that the feature-selection process has to be separated from the classification accuracy assessment, to avoid uncorrected estimates of the prediction error are obtained. This flaw in methodology is known as “selection bias”. While the problem may be reproduced with any wrapper algorithm, selection bias is a specific risk for recursive gene selection procedures, and especially for systems based on the RFE-SVM pair. As supported by [Simon et al., 2004] careful experimental schemes are therefore required by gene ranking and selection procedures in the optimization of a classification rule.

**Example 1.** (from [Furlanello et al., 2003c]) Consider a binary classification problem and a corresponding data set $S$, and its three subsets defined $S_0$, $S_1$ and $S_2$. Suppose that the best features set is obtained on $S_0$, defined by the features of a classification model with minimum cross-validation (CV) error estimated on $S_0$. The correct methodology requires that a model with the optimal features is then trained on $S_1$ and tested on $S_2$ to obtain an error estimate for new cases. If the test data set $S_2$ intersects with the $S_0$ used in the selection process (even if $S_1$ and $S_2$ are disjoint), an over-optimistic error rate will be estimated on $S_2$. The uncorrect finding may possibly lead to the conclusion that a panel of very few genes could be adequate to differentiate between classes. This methodology will be also correct if $S_0$ is a subset of $S_1$ (always referred to the hypothesis of disjoint $S_2$).

The problem of error evaluation for models developed on a reduced number of interesting genes should operate out-of-sample from the data in the selection process. The two processes also need to be intensively based on partition or resampling methods to smooth data variability. A whole learning procedure has to deal with the very small ratio between the number of samples and the number of variables. The crucial consequence is that model selection processes (including feature selection) require intensive replication of classification-and-ranking steps. An accurate modeling of microarrays data analysis will therefore entail heavy computational costs.

Therefore, a complete standard approach to gene profiling can be summarized in the following steps:

i. choose a suitable pair classifier/ranking method;

ii. filter, possibly discarding irrelevant features;

iii. rank the features;

iv. select an optimal subset of features;
v. train a classifier using only this subset and obtain a model;

vi. validate the model (gene profile + predictor) by cross validation or on additional data sets;

vii. apply to unlabeled data.

1.1.2 A Complete System

The experimental setup proposed in this dissertation is partially similar to those described in [Ambroise and McLachlan, 2002] and in [Furlanello et al., 2004]. The method is composed of three main procedures, organized as in Figure 1.1:

**ONF** (Optimal Number of Features); the procedure computes the optimal number of features ($n^*$).

**OFS-M** (Optimal Feature Set-Model); the procedure trains the model with the first $n^*$ ranked features. The model is tested on a test portion.

**VAL** (Validation); the procedure validates the OFS-M procedure over $B$ replicates according to a resampling scheme.

![Figure 1.1: General scheme of a complete procedure](image)

In summary, given a dataset (matrix of gene expressions), the VAL procedure analyzes $B$ replicated experiments (runs) according to a resampling
scheme. At each run, a training/test split (\(TR^b, TS^b\)) is created and only the training portion is used by OFS-M procedure. The ONF procedure identifies an optimal feature subset and the corresponding model is constructed in OFS-M procedure. The model is tested on the test portion \(TS^b\) (unused in the development of the model). Thus an average test (predictive) error

\[
ATE = \frac{1}{B} \sum_{b=1}^{B} TE^b
\]

can be computed from the \(B\) test error values \(TE^b\) in the VAL procedure.

1.1.3 Resampling Methods

Resampling methods are a keystone of a complete validation system. By randomly defining a training-test split of the available data, they allow us to train classification machine using only a portion of data, and secondly to validate (to verify the accuracy) the model on the remaining portion. In general, training/test pairs (replicates) have to be randomly created so that the error gives information about the model generalization.

K-fold Cross Validation

Given a dataset \(T\) of size \(N\), we split it into \(K\) subset \(T_1, \ldots, T_K\). The machine is trained on \(\bigcup_{i \neq j} T_i\) and then it is validated on \(T_j\). The splitting of the subsets is in general carried out independently of the class label. When the splitting procedure is corrected to approximate class proportions in the \(T_i\), we speak of stratified k-fold cross validation. The operation is carried out \(K\) times, obtaining \(K\) training/test pairs. Figure 1.2 shows a scheme

![Diagram showing K-fold cross validation](image)

Figure 1.2: K-fold cross validation with \(K = 4\). The colors indicates the elements of the two classes.
for the k-fold cross validation method, and Figure 1.3 shows the stratified k-fold cross validation method.

**Leave-one-out Cross Validation**

Given a dataset T composed of \( N = K \) samples, \( s_1, \ldots, s_K \), the machine is trained on \( \bigcup_{i \neq j} s_i \) and then it is validated on \( s_j \), with \( j = 1, \ldots, K \), i.e. training on \( N - 1 \) samples at each step. This method is exemplified in Figure 1.4 for \( k = 5 \).

**Fixed Size Resampling Cross Validation**

Given a dataset T we split it into \( F \) subset \( T_1, \ldots, T_F \). The machine is trained on \( \bigcup_{i \neq F} T_i \) and then it is validated on \( T_F \). The splitting of the subsets is carried out maintaining the class proportion. This operation is carried out \( N \) times, obtaining \( N \) training/test pairs. This method is shown in Figure 1.5.
Bootstrap

An alternative resampling method is the bootstrap [Efron, 1982], whose application is described in detail in [Warren and Gregory, 2001] and in [Hastie et al., 2001]. The bootstrap is also implemented in our system.

1.2 System Elements

1.2.1 A Classifier: Support Vector Machines

The concept of Support Vector Machine was introduced by Vapnik in the late 1970’s and described in detail in [Cristianini and Shawe-Taylor, 2000]. The aim of Support Vector classification is to find “good” separating hyperplanes in a high dimensional feature space and with a computationally efficient learning procedure.

The maximal margin classifier is the simplest model of Support Vector Machine and it was also the first to be introduced [Boser et al., 1992]. Unfortunately, it works only for data which are linearly separable in the feature space, and hence cannot be used in many real-world situations.

Support Vector Machines allow a change of representation of the data, also common preprocessing strategy in machine learning:

\[ x = (x_1, \ldots, x_n) \mapsto \phi(x) = (\phi_1(x), \ldots, \phi_N(x)). \]

This step is equivalent to mapping the input space \( X \) into a new space, \( F = \{ \phi(x) \mid x \in X \} \), in general of higher dimension, or of characteristics useful for the classification problem.
The quantities introduced to describe the data are usually called features, while the original quantities are sometimes called attributes. The task of choosing the most suitable representation is known as feature selection. The space $X$ is referred to as the input space, while $F = \{ \phi(x) \mid x \in X \}$ is called the feature space.

Figure 1.6 shows an example of a non-linear feature mapping between a two-dimensional feature space, the classes cannot be separated by a linear function in the input space, but separation happens to be possible in the transformed feature space.

Figure 1.6: A feature map can simplify the classification task

Different approaches to feature selection exist. Frequently one seeks to identify the smallest set of features that still conveys the essential information contained in the original attributes. This is known as dimensionality reduction,

$$x = (x_1, \ldots, x_n) \mapsto \phi(x) = (\phi_1(x), \ldots, \phi_d(x)), \quad d < n,$$

and can be very beneficial as both computational and generalization performance can degrade as the number of features grows. Another important feature selection task is the detection of irrelevant features and their subsequent elimination. Feature selection should be viewed as a crucial part of the learning process itself, and should be automated as much as possible.

Summarizing, in order to learn non-linear relations with a linear machine, we need to select a set of non-linear features and to rewrite the data
in the new representation. This is equivalent to applying a fixed non-linear mapping of the data to a feature space, in which the linear machine can be used. Hence, the set of hypotheses we consider will be functions of the type:

$$f(x) = \sum_{i=1}^{\infty} w_i \phi_i(x) + b,$$

(1.1)

where $\phi : X \rightarrow F$ is a non-linear map from the input space to some feature space. Moreover, we may aim at selecting minimal set of features. This means that we will build non-linear machines in two steps: first a fixed non-linear mapping transforms the data into a feature space $F$, and then a linear machine is used to classify them in the feature space.

One important property of a learning machines in the SVM paradigm is that they can be expressed in a dual representation; the hypothesis is expressed as a linear combination of the training points, so that the decision rule can be evaluated using in terms of the inner products between the test point and the training points:

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i \langle \phi(x_i) \cdot \phi(x) \rangle + b.$$  

If you have a way of computing the inner product $\langle \phi(x_i) \cdot \phi(x) \rangle$ in feature space directly as a function of the original input points, it becomes possible to merge the two steps needed to build a non-linear learning machine. We call such a direct computation method a kernel function. A kernel is a function $K$, such that for all $x, z \in X$

$$K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle,$$

where $\phi$ is a mapping from $X$ to an (inner product) feature space $F$.

Kernel representations offer a solution by projecting the data into a high dimensional feature space to increase a computational power of linear learning machines. The use of linear machines in the dual representation makes it possible to perform this step implicitly. The advantage is that, in this representation, the number of tunable parameters does not depend on the number of attributes being used. Another advantage of the kernel method is that the learning algorithms and theory can largely be decoupled from the specifics of the application area, which must be encoded into the design of an appropriate kernel function.
1.2.2 The Recursive Feature Elimination

The Recursive Feature Elimination (RFE) is a well-known feature selection method for Support Vector Machines firstly introduced in [Guyon et al., 2002]. This method has been evaluated in experimental analysis and it is considered a relevant method for gene selection and classification on microarrays.

As we have seen before, SVM realizes a classification function

\[ f(x) = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b, \]

where coefficients \( \alpha = (\alpha_i) \) and \( b \) are obtained by training over a set of examples \( S = \{(x_i, y_i)_{i=1,...,N}, x_i \in \mathbb{R}^n, y_i \in \{-1, 1\}\} \) and \( K(., .) \) is the chosen kernel. In the linear case, the SVM expansion defines the hyperplane

\[ f(x) = \langle w, x \rangle + b, \]

with

\[ w = \sum_{i=1}^{N} \alpha_i y_i x_i. \]

The idea is to define the importance of a feature for SVM in terms of its contribution to a cost function \( J(\alpha) \). At each step of the RFE procedure, a SVM is trained on the given data set, \( J \) is computed and the feature that less contributes to \( J \) is discarded. In the case of linear SVM, the difference due to the elimination of the \( i \)-th feature is

\[ \delta J(i) = w_i^2, \]

and in the non-linear case is

\[ \delta J(i) = \frac{1}{2} \alpha^t Z \alpha - \frac{1}{2} \alpha^t Z (-i) \alpha, \]

where \( Z_{i,j} = y_i y_j K(x_i, x_j) \).

The heavy computational cost of RFE is a function of the number of original variables, because a SVM must be trained each time a variable is removed. The elimination of a single variable at each step (as in the basic RFE procedure) is, however, inefficient. Indeed, at the first loops of the RFE algorithm, many weights are generally similar and concentrated nearby zero. The removal of a group of variables at every loop represents a feasible approach, and it was suggested in [Guyon et al., 2002]. A possible choice is to remove \( \sqrt{\#R} \) features at each step, where \( R \) is the set of
the remaining features, so as to obtain the SQRT-RFE procedure. Different, parametric, acceleration techniques (as SQRT-RFE) or gradient based methods have been proposed in several machine learning studies, showing that it would be possible to obtain accuracy close to basic RFE.

1.2.3 The Entropy-based Recursive Feature Elimination

The entropy-based recursive feature elimination (E-RFE), is a non parametric procedure for gene ranking, which accelerates — without reducing accuracy — the standard recursive feature elimination (RFE) method for SVMs. This strategy was introduced in [Furlanello et al., 2003a].

The aim of the E-RFE procedure is to provide a more flexible feature elimination mechanism in which the ranking is obtained by discarding groups of genes which contribute least to the SVM classifier. In E-RFE we cautiously discard, according to the entropy of the weight distribution, several (possibly many) genes at each step to drive the weight distribution in a high entropy structure of a few equally important variables.

An entropy function \( H \) is considered as a measure of the weight distribution. In order to compute the entropy, we split the range of the weights, normalized in the unit intervals (with \( n_{int} = \sqrt{\#R} \), and we compute the relative frequencies for each interval:

\[
p_i = \frac{\#\delta J(i)}{\#R}, \quad i = 1, \ldots, n_{int},
\]

where \( \delta J(i) \) counts the variation due to the elimination of the \( i \)-th feature. Entropy is then defined as the following function:

\[
H = - \sum_{i=1}^{n_{int}} p_i \log_2 p_i.
\]

The following inequality immediately descends from the definition of entropy:

\[
0 \leq H \leq \log_2 n_{int}.
\]

The two bounds correspond to the situations:

- \( H = 0 \): all the weights lie in one interval;
- \( H = \log_2 n_{int} \): all the intervals contain the same number of weights.

The entropy-based RFE algorithm eliminates a group of genes at every loop, with two different procedures applied to lower or higher values of
H. The distinction is needed to remove many genes that have a similar (low) weight while still preserving the residual distribution structure. The procedure is non-parametric, allowing for differences in H between different microarray classification problems. Let \( \{pw_i\}_{i \in R} \) be the projected weights, i.e. the weights linearly projected in the interval \([0, 1]\); let \( H \) be their entropy and \( H_t \) a threshold to discriminate feature importance. We set
\[
H_t = \frac{1}{2} \log_2 n_{int}
\]
to equally split the entropy values range. When \( H > H_t \) the weights are not concentrated: nevertheless, in some cases, many of them have approximately the same low value.

To take care of the situation where many weights are close to 0, it is necessary to introduce a further discriminating measure. Let \( M \) be the mean of the projected weights and \( M_t \) a suitable threshold for such a measure. This threshold must be chosen to decide which project weights should be eliminated: in fact, the situations where \( M \leq M_t \) are precisely those when many features should be discarded.

When \( H > H_t \) and \( M > M_t \), the features whose weight lies in the interval \([0, \frac{1}{n_{int}}]\) are eliminated. In the remaining cases (\( H > H_t \) and \( M \leq M_t \), or \( H \leq H_t \)), we cautiously discard the features whose weight is in the leftmost quantile are discarded through a bisection procedure. The stopping condition is that no more than half of the features with weights in \([0, \frac{1}{2}M]\) should be discarded. One-step RFE is applied when the number of variables has been reduced below a threshold value \( R_t \). This approach has to be chosen as a suitable compromise between the computational cost and the estimated size of supposed optimal features subset.

1.3 System Setup

1.3.1 The ONF Procedure

Given a training set TR, this procedure is applied to select the optimal number of features based on a ranking method. A resampling procedure is iterated \( K \) times, producing each time a (TR\(_k\), TS\(_k\)) split of TR. A feature ranking is applied to TR\(_k\). Then, \( n \) subsets are created with the first \( F_i \) features of the feature list (i.e. \( F_1 = 1, F_2 = 5, F_3 = 10, \ldots, F_n = 1000 \)). Therefore, for each \( k \) a model family \((M_{ki}, i = 1, \ldots, n)\) is produced, one for each increase of \( F_i \). The \( M_{ki} \) models are evaluated on the TS\(_k\) test data, computing TE\(_{ki}\) test errors, and we obtain the average error curve.
\[ \text{TE}_i = \frac{1}{K} \sum_{k=1}^{K} \text{TE}_{ki}. \]  

An exponential fit is applied to \( \text{TE} \), and the \( n^* \) value leading to saturation in terms of the exponential curve is returned as the ONF result. The complete scheme of the procedure is shown in Figure 1.7.

### 1.3.2 The OFS-M Procedure

Given a training set \( \text{TR} \), a feature ranking method produces a list of ranked features, from which an optimal feature set \( \text{OFS} \) of size \( n^* \) is selected. Based on ONF procedure, a model \( M \) is developed by a suitable learning method. The accuracy of OFS-M is validated by the VAL procedure. A scheme is shown in Figure 1.8.
1.3.3 The VAL Procedure

The OFS-M procedure is validated on $B$ replicated experiments (runs) using a resampling scheme. The model with $n^*$ features is tested on the test set, in order to minimize risk of data overfitting, obtaining a $TE^b$ error. The procedure returns the expected test error

$$ATE = \frac{1}{B} \sum_{b=1}^{B} TE^b.$$ 

A scheme for the procedure is shown in Figure 1.9.

![Figure 1.9: The VAL procedure](image-url)
Chapter 2

BioDCV Components

2.1 Introduction

From the specifications of the complete validation system we underlined in the previous chapter, two are the main requirements:

a the system has to elaborate and store a high quantity of data (i.e. models, predictions and errors); thus it is unfeasible to use flat text files to store outputs of single experiments.

b the system has a high computing cost, hence it has to be portable for a parallel (distributed) computing system to reduce the elaboration time.

We focused on open source, embeddable, C database engines. With “open source”, we mean that the database engine has to be distributed under a suitable license as the GNU Public License (GPL). This license guarantees that the licensed software is freely available for use and redistribution in other open source products. With “embedded”, we mean that it may be linked into the application, thus running in the same address space of the embedding application. As a result, no inter-process communication is required for database operations, either over the network or between processes on the same machine. And we use C because the main elements of the system are developed in C language (including the $\texttt{p}$ package, the main engine of the classification process).

If we want to make suitable the system for a distributed computing system, we should, as described in [Chen and Ripley, 2003], “take as much out of the monolithic system as possible and design a modular system where different tasks run as separate processes on different processors.
(servers) outside the statistical package (the statistics server)”. With “parallel computing system”, we mean for example an OpenMosix cluster (such as the MPBA cluster at ITC/IRST) and computational resources such as those of a virtual grid organization.

The crucial aspect is to transfer at the same time data and software, on every machine being used as computational node. In the computing near data approach, we will have to reduce the amount of data that needs to be transferred. As many operations as possible should be performed at the location where the data resides, and only the reduced summary information should be communicated across the statistical system [Chen and Ripley, 2003].

2.2 The pr Package

The pr package (http://mpa.itc.it/merler/merler.html) is a C toolbox for solving classification problems. It is the main engine of BioDCV. It implements $k$-nearest neighbor (multiclass), classification trees (multiclass, only numerical variables allowed), maximum likelihood (multiclass), support vector machines (binary), bagging versions of $k$-nearest neighbor, classification trees and support vector machines, AdaBoost for binary classification trees and support vector machines. It includes the implementation of feature preprocessing techniques (normalization of the examples, standardization of the features and principal components computation) and tools for resampling (cross-validation and bootstrap) and cost-sensitive techniques for both classification trees and support vector machines. The package is currently composed of the following software programs:

- **pr.features** allows preprocessing of input data, such as feature standardization. It also allows computation of principal components. The user has to run **pr.features** (possibly without modifying the input data), in order to produce the output file to be used as input file for **pr.model**.

- **pr.model** receives in input a file containing a feature set (the output of **pr.features**) and implements $k$-nearest neighbor, maximum likelihood, support vector machines and classification trees. Moreover, it also implements ensemble methods like bagging and AdaBoost.

- **pr.test** performs model evaluation on a given test set.

- **pr.reset** allows generation of cross-validation and bootstrap replicated data sets (for replicated experiments).
\texttt{pr\_feature\_selection} ranks the features. It implements RFE, E-RFE, ONE-RFE, SQRT-RFE.

\texttt{pr\_features\_extract} receives in input two files, one containing a feature set (the output of \texttt{pr\_features}) and the other with a ranked feature (the output of \texttt{pr\_feature\_selection}). It creates a feature set with \texttt{nvar} reorder variables.

The BioDCV project aims at creating a system that integrates efficiently these modules in terms of CPU, memory and I/O costs, but it has to be also flexible in order to cover more experimental design scenery combining different values of parameters.

2.3 The Database Engine

Data management can be very simple. In some cases, a flat text file is enough. More often, though, programs need to store and search a large amount of data, or structurally complex data, as in a distributed computing system in functional genomics. Database management systems are tools that programmers can use to do this work quickly and efficiently using off-the-shelf software. Software developers can choose from hundreds of good, commercially-available database systems. The problem is selecting the one that best solves the problems that their applications face.

Firstly, we have considered to use Berkeley DB library, which is very efficient but not suitable to perform complex query. In a second time, we have focused on SQLite library, which is less efficient but it is able to perform very complex SQL queries. And for this reason, SQLite library has been the final choice.

2.3.1 Berkeley DB

Berkeley DB (www.berkeleydb.com) is an open source embedded database library that provides-scalable, high-performance, transaction-protected data management services to applications. Berkeley DB provides a simple function call API (Application Program Interface) for data access and management for a number of programming languages, including C and PHP. All database operations happen inside the library. Multiple processes, or multiple threads in a single process, can all use the database at the same time as each uses the Berkeley DB library. Low-level services like locking, transaction logging, shared buffer management, memory management, and so on are all handled transparently by the library.
The database library itself is quite compact (under 300 kilobytes of text space on common architectures), but it can manage databases up to 256 terabytes in size. It also supports high concurrency, with thousands of users operating on the same database at the same time. Berkeley DB is small enough to run in tightly constrained embedded systems, but can take advantage of gigabytes of memory and terabytes of disk on high-end server machines.

Berkeley DB generally outperforms relational and object-oriented database systems in embedded applications for a couple of reasons. First, because the library runs in the same address space, no inter-process communication is required for database operations. The cost of communicating between processes on a single machine, or among machines on a network, is much higher than the cost of making a function call. Second, because Berkeley DB uses a simple function-call interface for all operations, there is no query language to parse, and no execution plan to produce.

Unfortunately, in contrast to most other database systems, Berkeley DB provides relatively simple data access services. Records in Berkeley DB are \((key, value)\) pairs (Table 2.1).

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>apple</td>
</tr>
<tr>
<td>sport</td>
<td>cricket</td>
</tr>
<tr>
<td>drink</td>
<td>water</td>
</tr>
</tbody>
</table>

Table 2.1: Example of key/data pairs

Berkeley DB supports only a few logical operations on records. They are:

- Insert a record in a table;
- Delete a record from a table;
- Find a record in a table by looking up its key;
- Update a record that has already been found.

Both keys and values can be arbitrary byte strings, either fixed-length or variable-length. As a result, programmers can put native programming language data structures into the database without converting them to a foreign record format first. Storage and retrieval are very simple, but the
application needs to know what the structure of a key and a value is in advance. Berkeley DB is able to operate on any data type the application uses, no matter how complex.

But although the interfaces are fairly simple, they are non-standard. If a programmer can predict in advance how an application will access data, then writing a low-level program to get and store records can be faster. It eliminates the overhead of query parsing, optimization, and execution. The programmer must understand the data representation, and must write the code to do the work, but once that’s done, the application can be very fast.

It is also possible to build a relational system on top of Berkeley DB. In fact, the popular MySQL relational system uses Berkeley DB for transaction-protected table management, and takes care of all the SQL parsing and execution. It uses Berkeley DB for the storage level, and provides the semantics and access tools. Moreover, requiring MySQL to be available on any computing resource being accessed by BioDCV may strongly reduce its portability.

### 2.3.2 SQLite

SQLite (www.sqlite.org) is a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine (Figure 2.1). The SQLite library complies with most of the standard SQL language. But it omits some features while at the same time it adds few features of its own. Therefore, it is semantically rich and offers high-level database access.

The database library itself is quite compact: less than 30K lines of C code, less than 250KB code space. SQLite is also simple, easy to use API and self-contained: no external dependencies. It is faster than popular client/server database engines for most common operations.

In SQLite transactions are atomic, consistent, isolated, and durable (ACID) even after system crashes and power failures. Database files can be freely shared between machines with different byte orders and they can be up to 2 terabytes ($2^{41}$ bytes) in size.

Sizes of strings and BLOBs limited only by available memory and database files can be freely shared between machines with different byte orders. TCL bindings are also included. Bindings for many other languages are available separately.

The original author of SQLite has dedicated the code to the public domain. Everyone is free to copy, modify, publish, use, compile, sell, or dis-
tribute the original SQLite code, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

![Block diagram of SQLite](from www.sqlite.org)

The concurrent access to the SQLite database uses a advisory POSIX lock (system call fcntl). The use of database in an environment with shared files on the network requires a correct implementation of the locking service of the network file system. We have tested locking by developing two simple software programs.

SQLite is a good compromise among performances (embedded), versatility (complex queries) and compatibility (can be accessed by ODBC), and hence it is suitable to the BioDCV development.

Another important advantage of SQLite is the availability of an API interface with PHP for web applications (see 2.6 for specifications).

### 2.4 OpenMosix and the Cluster Facility

Clustering technologies allow two or more networked Linux systems — called “nodes” — to combine their computing resources in order to solve common computing problems faster than it would be possible if they were tackling the problem on their own. Computing problems in question can be anything from complex CPU-intensive scientific computations to a
horde of miscellaneous processes with no underlying commonality. Choice of hardware is flexible, and the operating system must not be limited to any single hardware platform. Probably the best-known type of Linux-

![Figure 2.2: OpenMosix cluster at ITC/IRST](image)

based clusters is the Beowulf cluster. A Beowulf cluster consists of multiple machines connected to one another on a high speed LAN. In order for these systems to be able to pool their computing resources, special cluster-enabled applications must be written using clustering libraries. The most popular clustering libraries are PVM and MPI, both considered solution mature for producing. By using PVM or MPI, programmers can design applications that can span across an entire cluster’s computing resources rather than being confined to the resources of a single machine. For many applications, PVM and MPI allow computing problems to be solved at a rate that scales almost linearly in relation to the number of machines in the cluster.

The primary drawback of Beowulf clusters is that they require specially designed software (written with explicit PVM or MPI support) in order to take advantage of cluster resources.

However, those who “roll their own code” are actually a very small percentage of the computing public. Everyone else — all those who simply want to set up a cluster and see some kind of performance benefit using standard Linux applications — have a very real problem. Since the
applications that they use on a regular basis haven’t been written to be PVM or MPI aware, they imply can’t take advantage of a cluster. This is unfortunate, since it limits the use of clustering to a very small group of users. We resolved to use a technology that allow standard Linux applications to take advantage of a cluster without any need for them to be rewritten or even recompiled.

OpenMosix (http://openmosix.sourceforge.net/) is the GPLv2, Open Source, project which extends the MOSIX project. OpenMosix is a Linux kernel extension for single-system image clustering. This kernel extension turns a network of ordinary computers into a supercomputer for Linux applications. Once we have installed openMosix, the nodes in the cluster start talking to one another and the cluster adapts itself to the workload. Processes originating from any one node, if that node is too busy compared to others, can migrate to any other node. OpenMosix continuously attempts to optimize the resource allocation. Moreover, Live-CD distributions are available that allow to boot OpenMosix computing nodes any type of pc.

2.5 GRID Technology

Today Grid Computing [Foster and Kesselman, 2004] represents frontier of research activity in the field of parallel computing architectures. A large number of big companies of computing systems, software, informatics, have their own “Grid solution”, or by national and international actions they are linked to the key word “Grid”.

Grid technology allows us to have geographically distributed machines which provide services for computing and for the disk space. The technology also gives us an access to these services without choosing the machine: allowing to use them automatically and dynamically. The flexibility, which has been introduced by automatism, allows us to get more computing and disk resources when the needs increase.

2.5.1 EGRID Project

Egrid (www.egrid.it) is a project developed at the Abdus Salam International Center of Theoretical Physics - ICTP (www.ictp.it) in Trieste. It has realized a national grid for financial and economic research.

The Egrid infrastructure is based on Globus/EDG/LCG (www.globus.org) middleware, it is the same used by CERN. The Egrid project has
developed simple systems for use and installation of a grid system by using live cd. The grid of Egrid project can rely on the main site in Padova and other 5 peripheral sites.

According to the Egrid terminology, the grid consists of a machine set, and every machine — which is called “node” — has its different function:

- several nodes provide the elaboration ability and disk space;
- another set of nodes has to supervise the previous one and to update a working log continuously;
- other two specific nodes have the function to search the most suitable machines for a specific program run and for the localization of the required data;
- finally, a node has to receive requests from the user and then it has to submit these requests to the previous nodes for the execution.

A security mechanism, which identifies the user and the operations that he or she can execute, checks the node access.

The release 1.0 of the software dates back to the 7th October 2004, while the release 2.0 is under development and it will include a set of new features. The Egrid group co-operates with the following institute: Democritos/INFM (www.democritos.it), ITC/IRST (www.itc.it/irst/) and INAF Trieste (www.inaf.it).

### 2.6 Future Tasks

The final goal is the development of a complete platform for (genomic) data analysis integrating the computing power of a distributed system (either a cluster or a grid solution) with the flexibility of Web-connected server (Figure 2.3). This setup would allow field researchers to study their problems through an unbiased experimental setup without the hassle of configuring it. At the same time, such solution would let the user perform further computation on the obtained results by employing an on-site statistical engine even powered by a database server where to store sample-related information, offering a complete management of the problem.
Figure 2.3: Example of structure of a bioinformatics data analysis platform consisting of webserver endowed with the BioDCV solution (and other analysis tools), interlaced with a distributed computing system.
Chapter 3

BioDCV Design and Implementation

3.1 Introduction

The system developed (BioDCV, Biological Distributed Complete Validation) is based on the complete validation system shown in Section 1.3. Its main engine is a part of pr package, described in detail in Section 2.2. Programs in the pr package have been used like black box, because they are well structured. Though, it has been necessary to introduce changes, which were substantial in some cases. Eventually, it has been possible to use these programs in a multi-purpose software package like BioDCV. Besides the creation of the SQL wrapper, it has been necessary to introduce the following main limitations currently apply:

1. we have only used Support Vector Machines;
2. we have not used principal components;
3. we have introduced many changes in order to optimize memory management;
4. we have fixed the stratified bootstrap resampling algorithm;
5. we have implemented the fixed size cross validation resampling algorithm in C language, which was previously written in Perl language.

In BioDCV, each data produced in the elaboration is stored, and when the experiment finishes, we can retrace all the computing stages and carry out a more exhaustive analysis on the experiment itself.
CHAPTER 3. BIODCV DESIGN AND IMPLEMENTATION

There are three essential differences between BioDCV and the system shown in Section 1.3:

- the model and the prediction are computed on the entire data set, and also a prediction can be computed on an additional test set.
- the model and the prediction are also computed on \( k \) replicates.
- the replicates are also created with the first \( F_i \) (with \( i = 1, \ldots, n \)) features from the \( b \) replicates.

In conclusion, the main characteristics included in BioDCV system are the following:

**preprocessing:** row normalization and/or column normalization;

**SVM:** linear, gaussian and polynomial kernel;

**resampling methods:** \( k \)-fold cross validation (stratified or not), fixed size cross validation and bootstrap (stratified or not);

**feature ranking methods:** RFE, E-RFE, ONE-RFE, SQRT-RFE and BIS-RFE;

### 3.2 System Design

A single experiment is formed by three main steps (Figure 3.1 and 3.2), which correspond to the three main programs:

- **exp** prepares the experiment. Given a configuration file and a matrix of data (and eventually an additional test) it builds the setup database.

- **run** performs the validation procedure with \( b = \text{initial-run, \ldots, final-run} \). Given the setup database, it builds one local database, and for each \( b \) it builds one flat text file containing the ranking procedure weight. This is the program which works in parallel.

- **unify** joins the setup database and the local databases in a unique database containing the entire experiment.

Firstly, **Exp** builds the setup database (Figure 3.3). The setup database contains the matrix of the original data and the indices of all \( b \) replicates, in order that each **run** can build the \((TR_b, TS_b)\) pairs and can continue with the experiment. The command is:

```
exp configuration-file setup-database.
```
Secondly, given a setup database, run executes the validation procedure for $b = \text{initial-run}, \ldots, \text{final-run}$ and it builds a local database. It is important that all the runs $b = 1, \ldots, B$ are completed before joining all the local databases. For example, if we want to execute an experiment with $B = 100$ external replicates, having 10 CPUs, we can submit 10 run with initial-run = 1, 11, 21, 31, ..., 91 and final-run = initial-run + 9. Moreover, run returns a flat text file for each $b$: this file contains the weights generated by the classifier and they are associated with each feature for each feature step. The command is:

\[
\text{run initial-run final-run setup-database.}
\]

Figure 3.1: Exp prepares the experiment and runs performs the jobs

At the end, unify joins the local databases and it inserts them into the setup database creating the complete database. The replicates table of the setup database is replaced by the replicates table union of the local databases.

The flat text configuration file is composed by key, values pairs in the following way:

\[
\text{KEY: value1 value2 value3 ... valueN ;}
\]

### 3.3 Database Design

As discussed before, the setup database has to include both the matrix of original and preprocessed data, and the sample indices (i.e. the indices of the rows referred to the original matrix). In this way, every run can
build the needed \((TR_b, TS_b)\) pairs, and it continues the procedure. This is possible thanks to a \textit{replicates} table (Figure 3.3. This table includes a column with the \(b\) index (labeled \(n\text{\_ext\_rep}\)) and two columns labeled \(ind\text{\_tr}\) e \(ind\text{\_ts}\). The setup database is sketched in Figure 3.3.

After this first step, the setup database is read by every \textit{run}. The latter creates a local database during the computing procedures of the assigned \(b = \text{initial-run}, \ldots, \text{final-run}\) models. In this database every information about the validation procedure is available. The table is labeled \textit{replicates}
(Figure 3.4) and it is the only one included in the local database.

![LOCAL DATABASE](image)

Figure 3.4: The local database

The models (and thus the rows of the table) are indexed by \((b, k, F_i)\) set, which corresponds to the three columns:

- **\(n_{\text{ext\_rep}}\)** is \(b\);
- **\(n_{\text{int\_rep}}\)** is \(k\);
- **\(n_{\text{int\_fs}}\)** is \(F_i\).

Besides the definition of \((b, k, F_i)\) set uses the following nomenclature:

- if \(k, F_i > 0\), we refer to the \(b, k\) model built with the first \(F_i\) features of the feature list;
- if \(k = 0\) and \(F_i > 0\) we refer to the \(b\) model built with the first \(F_i\) features of the feature list;
- if \(F_i = 0\) we refer to the model built with all the features.

**\(\text{pred\_tr, pred\_ts}\)** are the predictions respectively for the training and for the test;

**\(\text{mod\_b}\)** contains the offset of the SVM model;
mod_a contains the $\alpha$ of the SVM model;

mod_w contains the weights of the SVM models;

feat_list contains the ranked feature list;

ind_tr, ind_ts contain the indices of samples;

n_star is the $n^*$;

err_tr contains the training error;

err_ts contains the TE (test error).
Chapter 4

Results

4.1 Data Description

Tests on several real DNA microarray data are performed on different platforms. Synopsis of the data set is provided in the Figure 4.1.

<table>
<thead>
<tr>
<th><strong>Tumor vs. Metastases</strong></th>
<th><strong>Colon cancer</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>76 (64 primary tumoral + 12 metastatic) samples</td>
<td>62 (22 normal + 40 tumor) samples</td>
</tr>
<tr>
<td>18063 genes</td>
<td>2000 genes</td>
</tr>
<tr>
<td>Affimetrix oligonucleotide array platform</td>
<td>Affimetrix oligonucleotide array platform</td>
</tr>
<tr>
<td>[Ramaswamy et al., 2003] and</td>
<td>[Alon et al., 1999]</td>
</tr>
<tr>
<td>[Rameswamy et al., 2001]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>High Sezary CTCL</strong></th>
<th><strong>INT-IFOM Sarcoma</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>30 (18 disease + 12 control) samples</td>
<td>35 (23 tumor type one +12 tumor type two ) samples</td>
</tr>
<tr>
<td>6660 genes</td>
<td>7143 genes</td>
</tr>
<tr>
<td>cDNA filter array platform</td>
<td>cDNA filter array platform</td>
</tr>
<tr>
<td>[Kari et al., 2003]</td>
<td>Unpublished data from INT-IFOM</td>
</tr>
</tbody>
</table>

Figure 4.1: Datasets
4.2 Tests on Egrid

4.2.1 Colon Cancer

An interesting test on grid was performed with colon cancer data. The experimental setup is the following:

- $B = 50$ fixed size cross validation;
- $K = 3$ fold cross validation;
- linear kernel for SVM, $C=100$;
- E-RFE feature ranking method.

The used machines are Intel Xeon @ 2.80GHz. The elaboration time was estimated repeating the experiment using an increasing number of CPUs. It is interesting to see that the scalability curve has a minimum using 4 CPUs. This trend is due to the increase of data quantity to transfer when the number of machines increases.

Figure 4.2: Scalability test of colon cancer dataset on egrid testbed

The test details (with 4 CPUs) are shown in the Table 4.1.
### Table 4.1: Test details with 4 CPUs (see the Egrid Tutorial at [http://www.egrid.it/doc/utenti/egrid-tutorial.pdf](http://www.egrid.it/doc/utenti/egrid-tutorial.pdf) for more details)

<table>
<thead>
<tr>
<th>Object</th>
<th>Elapsed (s)</th>
<th>System (s)</th>
<th>User (s)</th>
<th>CPU use (%)</th>
<th>Time (s)</th>
<th>Replicates</th>
<th>WN→MWN (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE→MWN</td>
<td>329</td>
<td>0.05</td>
<td>2.00</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MWN→WNs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mpirun</td>
<td>168.94</td>
<td>1.48</td>
<td>145.45</td>
<td>86</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run0</td>
<td>148.83</td>
<td>0.7</td>
<td>141.27</td>
<td>95</td>
<td>13</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Run1</td>
<td>150.05</td>
<td>1.12</td>
<td>142.18</td>
<td>95</td>
<td>13</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Run2</td>
<td>143.59</td>
<td>0.69</td>
<td>130.03</td>
<td>91</td>
<td>12</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Run3</td>
<td>144.0</td>
<td>0.90</td>
<td>135.19</td>
<td>94</td>
<td>12</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>MWN→SE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduled to Run</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>Run to Done</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>176</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>441</td>
</tr>
</tbody>
</table>
4.2.2 INT-IFOM Data

Another interesting test was performed on INT-IFOM data. The experimental setup is the following:

- \( B = 400 \) fixed size cross validation;
- \( K = 3 \) fold cross validation;
- linear kernel for SVM, \( C=100 \);
- E-RFE feature ranking method.

The scalability curve is shown in the Figure 4.2.

Table 4.2: Scalability test on egrid of INT-IFOM dataset

4.3 Tests on workstation and on Linux openMosix Clusters

Several tests have been performed on single machines and on openMosix cluster at ITC-irst. A complete comparison of BioDCV with the previous system (\texttt{pr} package + scripts) was possible only with the High Sezary CTCL data. Indeed,
this dataset is the unique on which the elaboration times of an entire experiment have been analyzed with the previous system. From this comparison, we can see that BioDCV is ten times faster (655 hours on a single Intel Pentium 3 @ 1 GHz machine with the old system; 3 hours and 40 minutes on a single Intel Pentium 4 @ 2.80 Ghz with BioDCV).
Bibliography


[Simon et al., 2004] Simon, R., McShane, L., Wright, G., Korn, E., and Ram- 

Springer.