



ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Physics and Astronomy Department
PhD Thesis in Applied Physics

Implementation and optimization of algorithms
in Biological Big Data Analytics

Supervisor:

Prof. Daniel Remondini

Correlator:

Prof. Gastone Castellani

Prof. Armando Bazzani

Presented by:

Nico Curti

Session 2019/2020

*"No one know nothing,
everyone know something,
but something is nothing to someone,
while
something is important to everybody"*

Daudi, Manyara

Abstract

Contents

1	Feature Selection	3
1.1	DNetPRO algorithm	5
1.2	Toy Model	6
1.3	DNetPRO Implementation	7
1.3.1	Pairs evaluation	7
1.3.2	Sorting	9
1.3.3	Network Signature	10
1.3.4	Python wrap	10
1.3.5	Pipeline	10
1.3.6	Time performances	10
1.3.7	Code availability	10
1.4	Synapse Dataset	10
1.5	Cytokinoma Dataset	10
1.6	Bovine Dataset	10
2	Deep Learning	11
2.1	NumPyNet	11
2.2	rFBP	11
2.3	Byron	11
2.4	Yolo	11
2.5	WDSR	11
2.6	UNet	12
3	Big Data	13
3.1	Web Scraping	13
3.2	CHIMeRA	13
3.3	CHIMeRA query	13

Introduction

in questo lavoro si affronteranno diverse tematiche relative alla Big Data Analytics e si propongono soluzioni inerenti ad ognuna di esse con esempi sviluppati ed applicati a dati reali. Partendo dalla curse of dimensionality e la feature extraction (dnet), passando per la visualizzazione dei dati con le NN fino alla eterogeneità dei dati (chimera)

definire feature come variable e dire che nel resto del testo verranno usati in maniera indistinta i due termini

Chapter 1

Feature Selection - DNetPRO algorithm

After the end of the Human Genome Project (HGP, 2003) [13] there has been growing interest on biological data and their analysis. At the same time, the availability of this type of data increased exponentially with the technological improvement of data extractors (High-Throughput technologies) [16] and with lower production costs. Lower costs and efficiency in time extraction are the main factors that allow us to go into the new scientific era of Big Data. Biological Big Data works with very large and complex datasets which are typically impossible to store, handle and analyze using standard computer and techniques [11]. Just think that we need around 140 Gb for the storage of the DNA of a single person and an Array Express, a compendium of public gene expression data, contains more than 1.3 million of genomes which have been collected in more than 45000 experiments [6]. Since the number of available data is getting greater, we need to design several storage databases to organize, classify and moreover to extract informations from them. The Bioinformatics European Institute (EBI) at Hinxton (UK), which is part of the European Laboratory of Biological Molecular and one of the biggest repositories of biological data, stores 20 petabytes of data and genomics and proteomics back-ups. The amount of the genomics data is only 2 petabytes, and it doubles every year: it is not worth to remark that these quantities represent about a tenth of data stored by CERN of Ginevra [12]. On the other hand, the ability of processing data and the computational techniques of analysis do not grow the same way. Therefore the gap between the great growth of the number of available data and our ability to work with them is getting bigger.

From a computational point of view, the Bioinformatics new-science is looking for new methods to analyze these large amount of data. The common Machine Learning methods, i.e computational algorithms able to identify significant patterns into large quantities of data, needs to be optimized and modified to increase their computational and statistical performances. To optimize the computational times we need to extend existing methods and algorithms and to develop new dimensionality reduction techniques. In Machine Learning, in fact, as the dimensionality of the data increases, the amount of data required to perform a reliable analysis grows exponentially¹. The dimensionality reduction techniques are methods able to identify the more significant variables of a given problem or a combination of them, where “significant” means that this smaller number of variables (or features) preserves the information about the problem as much as possible. So this huge amount of high-dimensional omics data (e.g. transcriptomics through microarray or NGS, epigenomics, SNP profiling, proteomics and metabolomics, but also metagenomics of gut microbiota) poses enormous challenges as how to extract useful information from them. One of the prominent problems is to extract low-dimensional sets of variables – sig-

¹ Often this phenomenon is called “curse of dimensionality”.

natures – for classification and diagnostic purposes, for example to better stratify patients for personalized intervention strategies based on their molecular profile [17, 2, 10, 1].

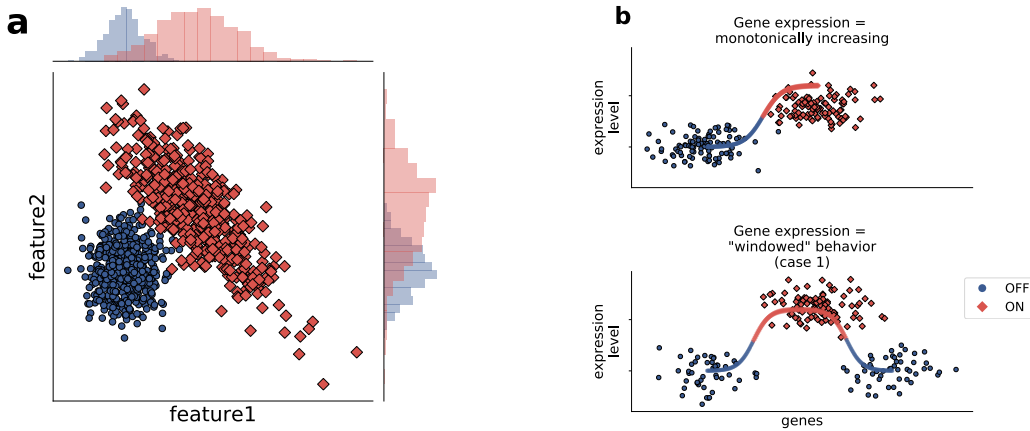


Figure 1.1: (a) An example in which single-parameter classification fails in predicting higher-dimension classification performance. Both parameters (*feature1* and *feature2*) badly classify in 1-D, but have a very good performance in 2D. Moreover, classification can be easily interpreted in terms of relative higher/lower expression of both probes. (b) Activity of a biological feature (e.g. a gene) as a function of its expression level: top) monotonically increasing, often also discretized to an on/off state; center, bottom) “windowed” behavior, in which there are two or more activity states that do not depend monotonically on expression level. X axis: expression level, Y axis, biological state (arbitrary scales).

Many approaches are used for these classification purposes [7], such as Elastic Net [9], Support Vector Machine, K-nearest Neighbor, Neural networks and Random Forest [14]. Some methods select signature variables by means of single-variable scoring methods [5, 8] (e.g. Student’s t test for a two-class comparison), while others search for projections in variable space, and then perform a dimensionality reduction by thresholding the projection weights, but these approaches could fail even in simple two-dimensional situations (Fig. 1.1).

Methods that select variables for multi-dimensional signatures based on single-variable performance can have limits in predicting higher-dimensional signature performance. As shown in Fig. 1.1(a), in which both variables taken singularly perform poorly, but their performance becomes optimal in a 2-dimensional combination, in terms of linear separation of the two classes.

It is known that complex separation surfaces characterize classification tasks associated to image and speech recognition, for which Deep Networks are used successfully in recent times, but in many cases biological data, such as gene or protein expression, are more likely characterized by a up/down-regulation behavior (as shown in Fig. 1.1(b) top), while more complex behaviors (e.g. a “windowed” optimal range of activity, Fig. 1.1(b) bottom) are much less likely. Thus, discriminant-based methods (and logistic regression methods alike) can very likely provide good classification performances in these cases (as demonstrated by our results with DNetPRO) if applied in at least two-dimensional spaces. Moreover, the “linearity” of these methods (that generate very simple class separation surfaces, i.e. linear or quadratic) guarantee that a “buildup” of a signature based on lower-dimensional signatures is feasible.

This consideration are relevant in particular for microarray data where we face on a small number of samples compared to a huge amount of variables (gene probes). This kind of problem, often called “large N , small S ” problem (where N is the number of features,

i.e variables, and S is the number of samples), tend to be prone to overfitting² and they are classified to ill-posed. The difficulty on the feature extraction can also increase due to noisy variables that can drastically affect the machine learning algorithms. Often is difficult to discriminate between noise and significant variables and even more as the number of variables rises.

In this thesis I propose a new method of features selection - DNetPRO, *Discriminant Analysis with Network PROCESSing* - developed to outperform the mentioned above problems. The method is particularly designed to gene-expression data analysis and it was tested against the most common feature selection techniques. The method was already applied on gene-expression datasets but my work focused on the benchmark of it and on its optimization for Big Data applications. The pipeline algorithm is made by many different steps and only a part of it was designed to biological application: this allow me to apply (part of) the same techniques also in different kind of problems with good results (see next sections).

1.1 DNetPRO algorithm

The DNetPRO algorithm generates multivariate signatures starting from all couples of variables tested with Discriminant Analysis. For this reason it can be classified as a combinatorial method and the computational time for variable space exploration is proportional to the square of the number of available variables (ranging from 10^3 to 10^5 in a typical high-throughput omics study). This behavior allows it to overcome some of the limits of single-feature selection methods, and provides a hard-thresholding approach at difference with projection-based variable selection methods. Certainly the combination evaluation is the most time expensive step of the algorithm and it needs accurate algorithmic implementation for Big Data applications (see the next section for further informations about the algorithm implementation strategy). The algorithm can be summarize as shown in 1.

Data: Data matrix (N, S)

Result: List of putative signatures

Divide the data into training and test by an Hold-Out method;

for *couple* \leftarrow (*feature_1*, *feature_2*) \in *Couples* **do**

 | Leave-One-Out cross validation;

 | Score estimation using a Classifier;

end

Sorting of the couples in ascending order according to their score;

Threshold over the couples score (K best couples);

for *component* \in *connected_components* **do**

 | **if** *reduction* **then**

 | Iteratively pendant node remotion;

 | **else**

 | S

 | **end**

 | signature evaluation using a Classifier;

end

Algorithm 1: DNetPRO algorithm for Feature Selection.

So, given an initial dataset, consisting in S *samples* (e.g. cells or patients) with N observations each (our *variables*, e.g. gene or protein expression profiles), the signature identification procedure can be summarized with the following pipeline:

² A solution to a problem is classified as “overfitted” if small fluctuations on the data variance produce classification errors.

- separation of available data into a training and a test set (e.g. 33/66, or 20/80);
- estimation of the classification performance on the training set of all $S(S - 1)/2$ variable couples through a computationally fast and reproducible cross-validation procedure (leave-one-out cross validation was chosen);
- selection of top-performing couples through a hard-thresholding procedure. The performance of each couple constitutes a *weighted link* of a network in which nodes are the variables connected at least through one link;
- every *connected component* in which the network is divided into constitutes an identified classification signature.
- (optional) in order to reduce the size of an identified signature, the pendant nodes of the network (i.e. nodes with degree equal to one) can be removed, in a single step or recursively until the core network (i.e. a network with all nodes with at least two links) is reached.
- all signatures are applied onto the test set to estimate their performance.
- a further cross validation step is performed (with a further dataset splitting into test and validation sets) to identify the best performing signature.

I would stress that this method is completely independent to the choice of the classification algorithm but from a biological point of view a simple one is preferred to preserve an easy interpretability of the results. The geometrical simplicity of the resulting class-separation surfaces, in fact, allows an easier interpretation of the results, as compared with very powerful but black-box methods like nonlinear-kernel SVM or Neural Networks. Moreover the network interaction of variables can keep an internal ranking score of features importance or possible features cooperation. These are the reasons that move us to use very simple classifier methods in our biological application as diag-quadratic Discriminant Analysis or Quadratic Discriminant Analysis (Appendix A for more informations about the mathematical background and implementation in the different languages). Both these methods allow fast computation and easy interpretation of the results. This linear separation might not be common in some classification problems (e.g. image classification) but it is very plausible in biological systems, in which many responses to perturbation consist in increase or decrease of variable values (e.g. expression of genes or proteins, see Fig. 1.1(b)).

In a general classification problem (e.g. image analysis) this could not be the case, since complex non linear separating surfaces may exist among the classes, but we hypothesize (and our results seem to confirm so) that in classification problems based on biological data such as gene expression these situations are not so common. This assumption is very plausible for biological data, since genes are in general up- or down-regulated in order to modify their activity, and protein and metabolites most of the times respond consequently.

A second direct gain by the couples evaluation is related to the network structure: the DNetPRO network signatures allow a hierarchical ranking of the features according to their centrality compared to possible Kbest signatures. This underlying network structure of the signature could suggest further methods for signature dimensionality reduction based on network topological properties to fit real application needs and it could help to evaluate the cooperation of the variables for the class identification.

1.2 Synthetic dataset benchmark

We firstly tested the DNetPRO method with synthetic data, consisting in a small set of discriminating variables together with a large number of “noisy” variables. Fixing the

number of informative features and classes we test the DNetPRO efficiency on the features extraction, compared to the results obtained by individually single features ranking (*Kbest* feature selection).

To simulate a synthetic “gene expression dataset”, with a large number of variables and a much smaller number of samples, we use the toy [model generator](#) provided by the *scikit-learn* [15] python package. This model generator allows to set a precise number of classes and it distinguishes between *informative features*, i.e. features which easily separate the class populations, and *redundant features*, i.e. features which represent noise in our problem. The number of informative features should be realistically small compared to the noise, so in our simulations we chose to introduce at least a 10% of informative features in the whole dataset.

1.3 Algorithm implementation

The DNetPRO algorithm is made by a sequence of different steps which have to be performed sequentially for a signature extraction. For this purpose each step can be optimized independently using the full set of available computational resources³. In this section will be analyzed each part of the pipeline focusing on the optimization strategies used for the algorithm implementation.

1.3.1 Combinatorial algorithm

The most computational time expensive step of the algorithm is certainly the couples evaluation. From a computation point-of-view this step requires ($O(N^2)$) operations for the full set of combination. Since we want to perform also an internal Leave-One-Out cross validation for the couple performances estimation we have to add a ($O(S - 1)$) to the algorithmic complexity. Lets focused on some preliminary considerations before the implementation discussion:

- **Performances:** we aim to apply our method on large datasets since we have to focused on time performances of the code and particularly on this step (identified as bottleneck). To reduce as much as possible the call stack inside our code we should perform the entire code with the small number of functions as possible and possibly inside a unique main. Moreover we can simplify the for loop and take care of the automatic code vectorization performed by the optimizer at compile time (SIMD, *Single Instruction Multiple Data*). A further optimization step to take in count is related to the cache accesses: the use of custom objects inside the code should benefit from cache accesses (AoS vs SoA, *Array of Structure* vs *Structure of Arrays*).
- **Interdependence:** the variable couple performances evaluation is a completely independent computational process and can be faced on as N^2 separately tasks. Thus it can be easily parallelizable to increase speed performance.
- **Simplify:** the use of simple classifier for performance evaluation simplify the computation and the storage of the relevant statistical quantities. In the discussed implementation we focused on a Diag-Quadratic classifier (see Appendix A for further informations) and only means and variances of the data plays a role in its evaluation.
- **Cross Validation:** the use of Leave-One-Out cross validation allows to perform substantially optimizations in the statistical quantities evaluations across the folds (see discussion in Appendix A - Numerical Implementation).

³ Further optimization can be performed in a cross validation environment and they will be discussed later in this section.

- **Numerical stability:** we have also to take in care the numerical stability of the statistics since we are working in the assumption of a reasonable small number of samples compared to the amount of variables. This behavior particularly affects the variance estimation: the chose of a numerical stable formula for this quantity play a crucial role for the computation because the classifier score has to be normalized by it.

With these idea in mind we can write a C++ code able to optimize this step of computation in a multi-threading environment with the purpose of testing its scalability over multi-core machines.

Starting from the first discussed point we chose to implement the full code inside a unique main function with the help of only a single SoA custom object and one external function (*sorting algorithm* discussed in the next section). This allows us to implement the code inside a single parallel section reducing the time of thread spawns. We chose to import the data from file in sequential mode since the I/O is not affected by parallel optimizations.

Following the instructions suggested in Appendix A - Numerical Implementation we compute the statistic quantities on the full set of data before starting the couples evaluation. Taking a look to the variance equation

$$\sigma^2 = \frac{\sum_{i=1}^S (x_i - \mu)^2}{S - 1} = \frac{\sum_{i=1}^S (x_i^2)}{S - 1} - \mu^2$$

we can see that the first equation involve the computation of the mean as a simple sum of the elements but a large number of subtractions from it that are numerical unstable for data outliers (moreover because they are elevated to square). The better choice in this case is given by the second formulation that allows us to compute the both quantities in the formula inside a single parallel loop⁴. At each cross validation we will use the two pre-calculated sums of variables removing the only data point excluded by the Leave-One-Out. Another precaution to take in care is to add a small epsilon to the variance before its use at denominator inside the classifier function to prevent numerical underflow.

The main role is still given by the couples loop. The set of pair variables can be obtained only by two nested for loops in C++ and naive optimization can be obtained by simply reduce the number of iterations following the triangular indexes of the full matrix (by definition the score of the couple (i, j) is equal to the score of (j, i)). This precaution easily allows the parallelization of the external loop and drastically reduce the number of iteration but it also creates a link between the two iteration variables. The new release of OpenMP libraries [4]⁵ (from OpenMP 4.5) introduce a new *keyword* of the language that allows the collapsing of nested for loops in a single one (whose number of iterations is given by the product of the single dimensions) in the only exception of completely independences of iteration variables. So the best strategy to use in this case is to perform the full set of N^2 iterations with a single `collapse` clause in the external loop⁶.

Listing 1.1: Python parallel couples evaluation algorithm

```
1 import pandas as pd
2 import itertools
3 import multiprocessing
```

⁴ To facilitate the SIMD optimization the code is written using only float (single precision) and integer variables. This precaution takes in care the register alignment inside the loops and facilitate the compile time optimizer.

⁵ The OpenMP library is the most common non-standard library for C++ multi-threading applications.

⁶ Obviously the iteration where the inner loop variable is lower than the outer one will be skipped by an if condition.

```

4
5 from sklearn.naive_bayes import GaussianNB
6 from sklearn.model_selection import LeaveOneOut, cross_val_score
7
8 def couple_evaluation (pair):
9     f1, f2 = pair
10    samples = data.iloc[[f1, f2]]
11    score = cross_val_score(GaussianNB(), samples.T, labels,
12                           cv=LeaveOneOut(), n_jobs=1).mean() # nested
13    parallel loops are not allowed
14
15    return (f1, f2, score)
16
17 def read_data (filename):
18     data = pd.read_csv(filename, sep='\t', header=0)
19     labels = data.columns.astype('float').astype('int')
20     data.columns = labels
21
22     return (data, labels)
23
24 if __name__ == '__main__':
25     filename = 'data.txt'
26
27     global data, labels
28     data, labels = read_data(filename)
29
30     Nfeature, Nsample = data.shape
31
32     couples = itertools.combinations(range(0, Nfeature), 2)
33
34     nth = multiprocessing.cpu_count()
35
36     with multiprocessing.Pool(nth) as pool:
37         score = zip(*pool.map(couple_evaluation, couples))

```

In this section we also provide an “equivalent” Python implementation with the use of common machine learning libraries and parallel settings (ref. 1.1). In the next sections we will discuss the computational performances of this naive implementation with C++ one discussed above.

1.3.2 Pair sort

The sorting algorithm starts at the end of variable couple evaluation and re-order the pairs in ascending order to ease the next steps of signature identification. This step is performed in the same code (and same parallel section) of the before section but it deserves an own topic for a better focus on the parallelization strategy chosen. Moreover there are many common parallel implementation of sorting algorithm and to reach the best performances we have to chose the appropriated one.

The sorting algorithm are already implemented in serial version in the major part of the languages (Python and C++ included). The naive version of the algorithms are also quite optimized and they perform the computation with complexity ($O(N\log(N))$)⁷. In this case we have not to re-invent any sorting technique but only insert as well as possible these algorithms inside a parallel sections and use the variable format chosen for couple performances storage. Since we are working with SoA objects we need to re-order all the structure arrays in the same way. So we can not use the a simple sort function but can compute the set of indexes that allow the re-order of the arrays, the so called **argsort**

⁷ We are considering only un-stable sort in which the preserving order of equivalent elements in the array is not guaranteed.

method. To rearrange the indexes according to a given array of values we can use the templates in C++.

As parallelization strategy we can yet invoke the new *keywords* of OpenMP libraries and apply a *divide-and-conquer* architecture using a tree of independent *tasks*⁸. Using the maximum power of two of the available threads we split the computation in equal size sub-arrays and perform independent *argsorts*. Then, going backwards to the subdivisions at each step we merge the sub-arrays two-by-two until the root.

1.3.3 Network signature

1.3.4 DNetPRO in Python

1.3.5 DNetPRO in Snakemake

1.3.6 Time performances

1.3.7 Code availability

The full code is open source and available at [3]. The code installation is automatically tested using *travis* (for Linux and MacOS environments) and *appveyor* (for Windows environments) at every commit. The installation can be performed using *CMake* or *Makefile* and a full set of installation instructions can be found in the on-line project documentation.

1.4 Synapse dataset

Description of the synapse datasets. Application of the DNetPRO on the Synapse dataset (mRNA, miRNA, RPPA) of Yuan et al. with two different pipelines. Discussion on obtained performances compared to the most common machine learning methods. Discussion on the ranking. Discussion on the extracted signature.

1.5 Cytokinoma dataset

Description of the cytokinoma dataset with statistics. Application of the DNetPRO on the Cytokine dataset. Discussion on the obtained signature and biological interpretation of the Alzheimer disease.

1.6 Bovine Paratuberculosis

Description of the bovine dataset with biological background. Application of the DNetPRO on the Bovine dataset with the description of the two singatures extracted. Discussion on biological interpretation of the genes.

⁸ Tasks in OpenMP are code blocks that the compiler wraps up and makes available to be executed in parallel.

Chapter 2

Deep Learning - Neural Network algorithms

Description of the modern deep neural networks. Computational problems and potential applications

2.1 Neural Network laboratory - NumPyNet

Description of the Neural Network laboratory developed in pure numpy. Study of the neural network functionality. Testing of the code against tensorflow.

2.2 Replicated Focusing Belief Propagation

Description of the rFBP library as optimization of the Julia code. Pure c++ implementation with Python wrap (sklearn compatibility). Scorer library as performance evaluation tool with parallel evaluation of scorers.

2.3 Build Your Own Neural network - Byron library

Limits of the most common neural network frameworks. Neural Network library for parallel computing developed in C++. Pyron as python wrap of the library. Description of the algorithms used to optimize the computation (ex. im2col vs winograd).

2.4 Object Detection - Yolo architecture

Introduction on the image classification and detection with Yolo architecture. Implementation in Byron with description of performances against darknet (original implementation). Focus on performances (time, memory, cpu).

2.5 Super Resolution - WDSR architecture

Introduction on Super Resolution problem with focus on state-of-art neural network architecture. Description of the Byron implementation and application on NMR data with the most common measurements. Super-resolution allows better detection!

2.6 Image Segmentation - UNet architecture

Introduction on Image Segmentation problem. Creation of the datasets with common image-processing methods Application of Unet (Byron implementation) on femur images.

Chapter 3

Biological Big Data - CHIMeRA project

Many public datasets available. Description of the database used in chimera. Problems about the intersections and partial informations (single db).

3.1 Data extraction - Web scraping

Description of the web scraping techniques used to obtain the "no-public" datasets. Reference to the github project.

3.2 The CHIMeRA project

What is CHIMeRA project and which is its potentiality. Description of the database created and of the query implemented to obtain the results

3.3 CHIMeRA query

Some query examples like leukemia subnetwork and PRNP subnetwork. Description of the information extracted by these subnetworks.

Conclusions

Appendix A - Discriminant Analysis

The classification problems aim to associate a set of *pattern* to one or more *classes*. With *pattern* we identify a multidimensional array of data labeled by a pre-determined tag. In this case we talk about *supervised learning*, i.e the full set of data is already annotated and we have prior knowledge about data association to the belonging classes. Since in this work only supervised learning algorithms have been analyzed we do not cite other different learning methods.

In machine learning a key rule is assumed by Bayesian methods, i.e methods which use a Bayesian statistical approach to the analysis of data distributions. It can be proof that if the distributions under analysis are known, i.e a sufficient number of moments of it is known with a sufficient precision, the Bayesian approach is the best possible method to face on the classification problem.

Mathematical formulation

Since the exact knowledge of the prior probabilities and conditional probabilities is possible only on theory a parametric approach is often needed. A parametric approach aim to create reasonable hypothesis about the distribution under analysis and its fundamental parameters (e.g mean and variance). In the next of this discussion we focused only on normal distributions for convenience.

Given the multi-dimensional form of Gauss distribution:

$$G(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \cdot |\Sigma|^{1/2}} \cdot \exp \left[-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu) \right]$$

where \mathbf{x} is a column d -dimensional vector, μ the mean vector of the distribution, Σ the covariance matrix ($d \times d$), $|\Sigma|$ and Σ^{-1} the determinant and the inverse of Σ , respectively, we can notice the G depends quadratically by \mathbf{x} ,

$$\Delta^2 = (\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu)$$

where the exponent (Δ^2) is called Mahalanobis distance of vector \mathbf{x} from its mean. This distance can be reduced to the Euclidean distance when the covariance matrix is the identity \mathbf{I} .

The covariance matrix is always symmetric and positive semi-definite (useful information for next algorithmic strategies) so it has an inverse. If the covariance matrix has only diagonal terms the multidimensional distribution can be express as simple product of d mono-dimensional normal distributions. In this case the main axes are parallel to the Cartesian axes.

Starting from the multi-variate Gaussian distribution expression¹, the Bayesian rule for classification problems can be rewrite as:

¹ In Machine Learning it will correspond to the conditional probability density.

$$g_i(\mathbf{x}) = P(w_i|\mathbf{x}) = \frac{p(\mathbf{x}|w_i)P(w_i)}{p(\mathbf{x})} = \frac{1}{(2\pi)^{d/2} \cdot |\Sigma_i|^{1/2}} \cdot \exp \left[-\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) \right] \frac{P(w_i)}{p(\mathbf{x})}$$

where, removing constant terms (π factors and absolute probability density $p(\mathbf{x}) = \sum_{i=1}^s p(\mathbf{x}|w_i) \cdot P(w_i)$) and using the monotonicity of the function, we can extract the logarithmic relation:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) - \frac{1}{2} \log |\Sigma_i| + \log P(w_i)$$

which is called Quadratic Discriminant function.

The function dependency by the covariance matrix allows 5 different cases:

- $\Sigma_i = \sigma^2 I$ - **DiagLinear Classifier**

This is the case of completely independence of features, where they have equal variance for each class. This hypothesis allow us to simplify the discriminant function as:

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(\mathbf{x}^T \mathbf{x} - 2\mu_i^T \mathbf{x} + \mu_i^T \mu_i) + \log P(w_i)$$

and removing all the $\mathbf{x}^T \mathbf{x}$ constant terms for each class

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(-2\mu_i^T \mathbf{x} + \mu_i^T \mu_i) + \log P(w_i) = \mathbf{w}_i^T \mathbf{x} + \mathbf{w}_0$$

This simplifications create a linear discriminant function where the separation surfaces between classes are hyper-planes ($g_i(\mathbf{x}) = g_j(\mathbf{x})$).

With equal prior probability the function can be rewritten as

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(\mathbf{x} - \mu_i)^T (\mathbf{x} - \mu_i)$$

which is called *nearest mean classifier* where the equal-probability surfaces are hyper-spheres.

- $\Sigma_i = \Sigma$ (**diagonal matrix**) - **Linear Classifier**

In this case the classes have same covariances but each feature has its own different variance. After the Σ substitution in the equation, we obtain

$$g_i(\mathbf{x}) = -\frac{1}{2} \sum_{k=1}^s \frac{(\mathbf{x}_k - \mu_{i,k})^2}{\sigma_k^2} - \frac{1}{2} \log \prod_{k=1}^s \sigma_k^2 + \log P(w_i)$$

where we can remove constant \mathbf{x}_k^2 terms (equals for each class) and obtain another time a linear discriminant function where the discriminant surfaces are hyper-planes and equal-probability boundaries given by hyper-ellipsoids. Note that the only difference from the previous case is the normalization factor of each axes that in this case is given by the its variance.

- $\Sigma_i = \Sigma$ (**non-diagonal matrix**) - **Mahalanobis Classifier**

In this case we assume that each class has the same covariance matrix but they are non-diagonal ones. The discriminant function becomes

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma^{-1}(\mathbf{x} - \mu_i) - \frac{1}{2} \log |\Sigma| + \log P(w_i)$$

where we can remove the $\log |\Sigma|$ term because it is constant for all the classes and we can assume equal prior probability. In this case we obtain

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma^{-1}(\mathbf{x} - \mu_i)$$

where the quadratic term is the Mahalanobis distance, i.e a normalization of the distance according to the inverse of their covariance matrix. We can proof that expanding the scalar product and removing the constant term $\mathbf{x}^T \Sigma^{-1} \mathbf{x}$, we obtain yet a linear discriminant function with the same properties of the previous case. In this case the hyper-ellipsoids have axes aligned according to the eigenvectors of the Σ matrix.

- $\Sigma_i = \sigma_i^2 I$ - **DiagQuadratic Classifier**

In this case we have different covariance matrix for each class but they are proportional to the identity matrix, i.e diagonal matrix. The discriminant function in this case becomes

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \sigma_i^{-2}(\mathbf{x} - \mu_i) - \frac{1}{2} \log |\sigma_i^2| + \log P(w_i)$$

where this expression can be further reduced obtaining a quadratic discriminant function. In this case the equal-probability boundaries are hyper-spheres aligned according to the feature axes.

- $\Sigma_i \neq \Sigma_j$ (**general case**) - **Quadratic Classifier**

Starting from the more general discriminant function we can relabel the variables and highlight its quadratic form as

$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{W}_{2,i} \mathbf{x} + \mathbf{w}_{1,i}^T \mathbf{x} + \mathbf{w}_{0,i} \quad \text{with} \quad \begin{cases} \mathbf{W}_{2,i} = -\frac{1}{2} \Sigma_i^{-1} \\ \mathbf{w}_{1,i} = \Sigma_i^{-1} \mu_i \\ \mathbf{w}_{0,i} = -\frac{1}{2} \mu_i^T \Sigma_i^{-1} \mu_i - \frac{1}{2} \log |\Sigma_i| + \log P(w_i) \end{cases}$$

In this case each class has its own covariance matrix Σ_i and the equal-probability boundaries are hyper-ellipsoids oriented according to the eigenvectors of the covariance matrix of each class.

The Gaussianity of dataset distribution should be tested before using this classifiers. It can be performed using statistical tests as *Malkovic-Afifi* based on *Kolmogorov-Smirnov* index or just simpler with the empirical visualization of the data points.

Numerical Implementation

From a numeric point of view we can exploit each mathematical information and assumption to simplify the computation and improve the numerical stability of our computation. I would remark that this consideration were taken into account in this work only for the C++ algorithmic implementation since these methods are already implemented in the high-level programming languages as *Python* and *Matlab*².

In the previous section we highlight that the covariance matrix is a positive semi-definite and symmetric matrix by definition and this properties allows the matrix inversion. The computation of the inverse-matrix is a well known complex computation step from a numerical point-of-view and in a general case can be classified as an $O(N^3)$ algorithm. Moreover the use of a Machine Learning classifier commonly match the use of a cross validation method, i.e multiple subdivision of the dataset in a training and test sets. This involves the computation of multiple inverse matrix and it could represent the performance bottleneck in many cases (the other computations are quite simple and the algorithm complexity is certainly less than $O(N^3)$).

Using the information about the covariance matrix we can find the best mathematical solution for the inverse matrix computation that in this case is given by the Cholesky decomposition algorithm. The Cholesky decomposition or Cholesky factorization allows to re-write a positive-definite matrix into the product of two triangular matrix (the first is the conjugate transpose of the second)

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T = \mathbf{U}^T\mathbf{U}$$

The complexity of the algorithm is the same but the inverse estimation is simpler using a triangular matrix and the entire inversion can be performed in-place. It can also be proof that general inverse matrix algorithms have numerical instability problems compared to the Cholesky decomposition. In this case the original inverse matrix can be computed by the multiplication of the two inverses as

$$\mathbf{A}^{-1} = (\mathbf{L}^{-1})^T(\mathbf{L}^{-1}) = (\mathbf{U}^{-1})(\mathbf{U}^{-1})^T$$

As second bonus, the cross validation methods involve the subdivision of the data in multiple non-independent chunks of the original data. The extreme case of this algorithm is given by the Leave-One-Out cross validation in which the superposition of the data between folds are $N - 1$ (where N is the size of the data). The statistical influence of the swapped data is quite low and the covariance matrix will be quite similar between one fold to the other (the inverse matrix will be drastically affected from each slight modification of the original matrix instead). A second step of optimization can be performed computing the original full-covariance matrix of the whole set of data ($O(N^2)$) and at each cross-validation

² For completeness we have to highlight that for the Matlab case classification functions, i.e *classify*, is already included in the base packages of the software, i.e no external Toolbox are needed, while for the Python case the most common package which implements these techniques are given by the *scikit-learn* library. Matlab allows to set the classifier type as input parameter in the function using a simple string which follows the same nomenclature previously proposed. Python has a different import for each classifier type: in this case we find correspondence between our nomenclature and the Python one only in *quadratic* and *linear* cases, while the *Mahalanobis* is not considered a putative classifier. The *diagquadratic* classifier is called *GaussianNB* (*Naive Bayes Classifier*) instead. The last important discrepancy between the two language implementation is in the computation of the variance (and the corresponding covariance matrix): Matlab proposes the variance estimation only in relation to the mean so the normalization coefficient is given by the number of sample except by one ($N - 1$), while Python compute the variance with a simple normalization by N .

step evaluate the right set of k indexes needed to modify the matrix entrances ($O(N * k)$) that in the Leave-One-Out case are just one. This second optimization consideration can also be performed in the Diag-Quadratic case substituting the covariance matrix with the simpler variance vector.

Both these two techniques were used in the custom C++ implementation of the Quadratic Discriminant Analysis classifier and in the Diag-Quadratic Discriminant Analysis classifier for the DNetPRO algorithm implementation (see 1.1).

Appendix N - Bioinformatic Pipeline Profiling

Bibliography

- [1] J. S. Beckmann and D. A. Lew. Reconciling evidence-based medicine and precision medicine in the era of big data: challenges and opportunities. In *Genome Medicine*, 2016.
- [2] I. S. Chan and G. S. Ginsburg. Personalized medicine: Progress and promise. *Annual Review of Genomics and Human Genetics*, 12(1):217–244, 2011. PMID: 21721939.
- [3] N. Curti. QDANetPRO pipeline: Implementation of the qdanetpro pipeline for tcga datasets. <https://github.com/Nico-Curti/DNetPRO>, 2017.
- [4] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.
- [5] L. Eckhard. A universal selection method in linear regression models. *Open Journal of Statistics*, 2, 2012.
- [6] C. Greene, J. Tan, M. Ung, J. Moore, and C. Cheng. Big data bioinformatics. *Journal of cellular physiology*, 229(12), 2014.
- [7] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Mach. Learn.*, 2002.
- [8] R. R. Hocking. A biometrics invited paper. the analysis and selection of variables in linear regression. *Biometrics*, 32(1):1–49, 1976.
- [9] J. J. Hughey and A. J. Butte. Robust meta-analysis of gene expression using the elastic net. *Nucleic Acids Research*, 2015.
- [10] T. M. Johnson. Perspective on precision medicine in oncology. *Pharmacotherapy: The Journal of Human Pharmacology and Drug Therapy*, 37(9):988–989, 2017.
- [11] D. Kumari and R. Kumar. Impact of biological big data in bioinformatics. *International Journal of Computer Applications*, 101(11):22–24, 2014.
- [12] V. Marx. The big challenges of big data. *Nature Reviews*, 498(255), 2013.
- [13] M. McKinney. Human genome project information. *Reference Reviews*, 26(3):38–39, 2012.
- [14] H. Pang, S. L. George, K. Hui, and T. Tong. Gene selection using iterative feature elimination random forests for survival outcomes. *IEEE/ACM Transactions on Computational Biology and Bioinformatics / IEEE*, 2012.
- [15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

BIBLIOGRAPHY

- [16] J. A. Reuter, D. V. Spacek, and M. P. Snyder. High-throughput sequencing technologies. *Molecular Cell*, 2015.
- [17] K. Scotlandi, D. Remondini, G. Castellani, M. C. Manara, F. Nardi, L. Cantiani, M. Francesconi, M. Mercuri, A. M. Caccuri, M. Serra, S. Knuutila, and P. Picci. Overcoming resistance to conventional drugs in ewing sarcoma and identification of molecular predictors of outcome. *Journal of Clinical Oncology*, 27(13):2209–2216, 2009. PMID: 19307502.

Acknowledgment