Alma Mater Studiorum · Università di Bologna

Physics and Astronomy Department
PhD Thesis in Applied Physics

Implementation and optimization of algorithms in Biomedical Big Data Analytics

Supervisor:

Prof. Daniel Remondini

Correlator:

Prof. Gastone Castellani
Prof. Armando Bazzani

Presented by:

Nico Curti

Session 2019/2020

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 the association between data and classes.

In machine learning a key rule is played by Bayesian methods, i.e methods which use a Bayesian statistical approach to the analysis of data distributions. It can be proved that, if the underlying distributions are known, i.e a sufficient number of its moments are known with a sufficient precision, the Bayesian approach is the best possible method to face the classification problem (*Bayesian error rate*[11]).

Mathematical background

The exact knowledge of prior probabilities and conditional probabilities are generally hard to evaluate thus a parametric approach is often needed. A parametric approach aim to create reasonable hypothesis about the data distribution and its fundamental parameters (e.g mean, variance, $\cdots$). In the following discussion, we are going to focus only on normal distributions for mathematical convenience but the results could be easily generalized.

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 d-dimensional column vector, μ is the mean vector of the distribution, Σ is the covariance matrix ($d\times d$) and $|\Sigma|$ and Σ^{-1} the determinant and the inverse of Σ, respectively, we can notice the quadratic dependence of G depends 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 one when the covariance matrix is the identity matrix ($\mathbf{I}$).

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

Starting from a multi-variate Gaussian distribution[1], the Bayesian rule for classification problems can be rewritten as:

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

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

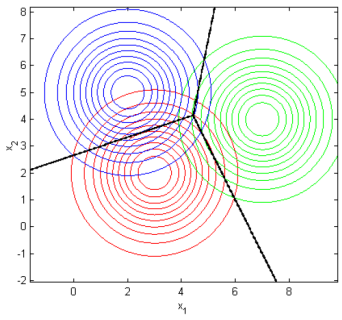
where, removing constant terms (π factors and the absolute probability density $p(\mathbf{x}) = \sum_{i=1}^{s} p(\mathbf{x}|w_i) \cdot P(w_i)$) and using the monotonicity of the function, we can extract the logarithmic relation:

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

which is called *Quadratic Discriminant function.*

The dependency by the covariance matrix allows 5 different cases:

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



This is the case in which features are completely independent, i.e they have equal variances for each class. This hypothesis allows us to simplify the discriminant function as:

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

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

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(-2{\mu_i}^T\mathbf{x} + {\mu_i}^T\mu_i) + \log P(w_i) = \mathbf{w_i}^\mathbf{T}\mathbf{x} + \mathbf{w_0}$$

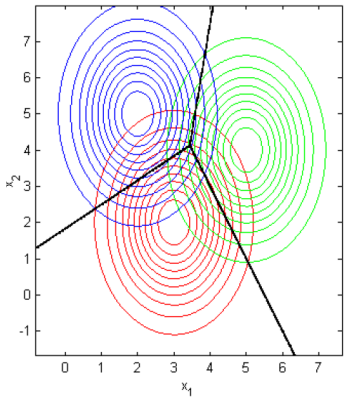
These simplifications create a linear discriminant function and the separation surfaces between classes are hyper-planes ($g_i(\mathbf{x}) = g_j(\mathbf{x})$).

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* and the equal-probability surfaces are hyper-spheres.

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

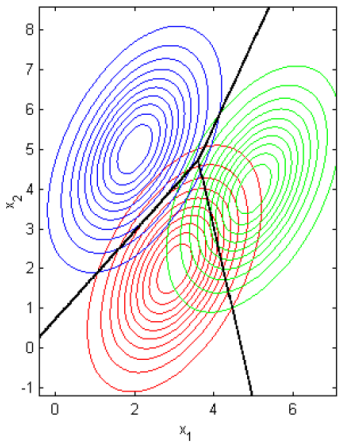


In this case the classes have same covariances but each feature has its own different variance. After the substitution of Σ in the equation, we obtain

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

where we can remove constant ${\mathbf{x_k}}^2$ terms (equal for each class) and obtain another time a linear discriminant function and discriminant surfaces given by hyper-planes and equal-probability boundaries given by hyper-ellipsoids. We remark that the only difference from the previous case is the normalization factor of each axis that in this case is given by 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

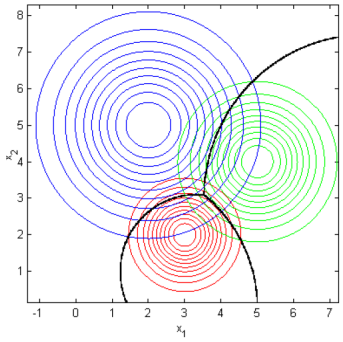
$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T\Sigma^{-1}(\mathbf{x} - \mu_i) - \frac{1}{2}\log|\Sigma| + \log P(w_i)$$

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

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

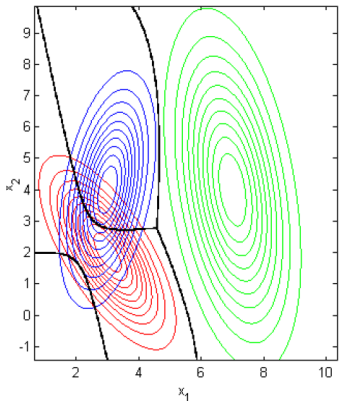


In this case we have a different covariance matrix for each class but they are all proportional to the identity matrix, i.e diagonal matrix. The discriminant function in this case becomes

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

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

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

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

The Guassian distribution hypothesis of data should be tested before using this classifiers. It can be evaluated using statistical tests as *Malkovich-Afifi* based on *Kolmogorov-Smirnov* index or using the empirical visualization of the data points.

Numerical Implementation

From a computational point-of-view we can exploit each mathematical information and assumption to simplify the computation and improve the numerical stability of our computation. We would remark that these considerations were taken into account in this

work only for the C++ algorithmic implementation, since these methods are already implemented in high-level programming languages as Python and Matlab[2].

In the previous section we highlighted the covariance matrix properties, i.e the covariance matrix is a positive semi-definite and symmetric matrix by definition and these properties allow the matrix inversion. The computation of the inverse-matrix is a well known complex computational step from a numerical point-of-view and in a general case can be classified as an $O(N^3)$ algorithm. Moreover, the usage of a Machine Learning classifier commonly matches the usage of a cross validation method, i.e multiple subdivision of the dataset into training and test sets. This involves the computation of multiple inverse matrices and it could represents the performance bottleneck in many real applications (the other computations are quite simple and their algorithmic complexity are certainly less than $O(N^3)$).

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

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

The algorithmic complexity is still 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 proved that general inverse matrix algorithms suffer of numerical instability issues compared to the output of 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, cross validation methods involve the data splitting 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 data superposition between folds is $N-1$ (where N is the size of the data). The statistical influence of the swapped data is quite low and the covariance matrix would be quite similar across folds (the inverse matrix would 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 modify it into the right k indexes at each cross-validation step ($O(N * k)$) that in the Leave-One-Out become a single editing case. This second optimization can also be performed in the Diag-Quadratic case substituting the covariance matrix with the simpler variance vector.

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

Both these two techniques have been used in the C++ implementation of the Quadratic Discriminant Analysis classifier and in the Diag-Quadratic Discriminant Analysis classifier used in the DNetPRO algorithm implementation (see **??**).

Appendix B - Venice Road Network

Tourist flows in historical cities are continuously growing in a globalized world and adequate governance processes, politics and tools are necessary in order to reduce impacts on the urban livability and to guarantee the preservation of cultural heritage. The ICTs offer the possibility of collecting large amount of data that can point out and quantify some statistical and dynamic properties of human mobility emerging from the individual behavior and referring to a whole road network. In this work we analyze a new dataset that has been collected by the Italian mobile phone company TIM, which contains the GPS positions of a relevant sample of mobile devices when they actively connected to the cell phone network. Our aim is to propose innovative tools allowing to study properties of pedestrian mobility on the whole road network. Venice is a paradigmatic example for the impact of tourist flows on the resident life quality and on the preservation of cultural heritage. The GPS data provide anonymized geo-referenced information on the displacements of the devices. After a filtering procedure, we develop specific algorithms able to reconstruct the daily mobility paths on the whole Venice road network. The statistical analysis of the mobility paths suggests the existence of a travel time budget for the mobility and points out the role of the rest times in the empirical relation between the mobility time and the corresponding path length. We succeed to highlight two connected mobility subnetworks extracted from the whole road network, that are able to explain the majority of the observed mobility. Our approach shows the existence of characteristic mobility paths in Venice for the tourists and for the residents. Moreover the data analysis highlights the different mobility features of the considered case studies and it allows to detect the mobility paths associated to different points of interest. Finally we have disaggregated the Italian and foreigner categories to study their different mobility behaviors.

The datasets

The dataset used in this study has been provided by the Italian mobile phone company TIM and contains geo-referenced positions of tens of thousands anonymous devices (e.g. mobile phones, tablets, etc. ...), whenever they performed an activity (e.g. a phone call or an Internet access) during eight days from 23/2/2017 up to 02/03/2017 (Carnival of Venice dataset), and from 14/7/2017 up to 16/7/2017 (*Festa del Redentore* dataset). According to statistical data, 66% of the whole Italian population has a smart-phone and TIM is one the greatest mobile phone company in Italy whose users are $\sim$ 30% of the whole smart-phone population. The datasets refer to a geographical region that includes an area of the Venice province, so that it is possible to distinguish commuters from sedentary people and the different transportation means used to reach Venice. Each valid record gives information about the GPS localization of the device, the recording time, the signal quality and also the roaming status, which in turns allow to distinguish between Italian and foreigners. The devices are fully anonymized and not reversible identification numbers (ID) are automatically provided by the system for mobile phones and calls within the scope of the trial; the ID is kept for a period of 24 hours. During each activity a sequence of GPS

data is recorded with a 2 sec. sampling rate and the collection stops when the activity ends. As matter of fact during an activity most of people reduce their mobility except if they are on a transportation mean, so that the dataset contains a lot of small trajectories that have to be joined to reconstruct the daily mobility. After a filtering procedure these data provide information on the mobility of a sample containing 3000 – 4000 devices per day. Since the presences during the considered events were of the order of 105 individuals per day, as reported by the local newspapers, we estimate an overall penetration of our sample of 3 – 4%. The filtering procedure and the other statistical information about the sample penetration are discussed in the original paper [14].

Mobility paths reconstruction on the road network

The procedure of mobility path reconstruction considers separately the land mobility and the water mobility since the two mobility networks have different features, so that it is necessary to check carefully the transitions from one network to the other. To create a mobility path, we connect two successive points left by the same device using a best path algorithm on the road network with a check on the estimated travel speed to avoid unphysical situations and discarding the paths whose velocity is clearly not consistent with the typical pedestrian velocity (or ferryboat velocity). To end a land path and to start a water path, we require that at least two successive points of the same device are attributed to a ferryboat line by the localization algorithm. In the case of a single point on a ferryboat line, we force the localization of this point on the nearest road on the land.

The reconstruction of the mobility paths also allows to study how people perform their mobility on the road network. We consider the problem of determining the most used subnetwork of the Venice road network. The existence of mobility subnetworks could be the consequence of the peculiarity of Venice road network, where it is quite easy to get lost if you do not have a map. Therefore people with a limited knowledge of the road network move according to paths suggested by Internet sites or following the signs on the roads. To point out a mobility subnetwork we rank the roads of Venice according to a weight proportional to the number of mobility paths passing through each road. Thus We define a relevant subnetwork as a connected subnetwork that explains a considerable fraction of the observed mobility. In this case each road (identified by two nodes in the poly-line format) represents the link of our weighted graph and we can apply the DNetPRO technique shown in **??** to identify the network core with only closed paths[3].

Starting from the previously evaluated daily flows for each road, we order in a decreasing way the roads according to the observed flows. The DNetPRO algorithm scrolls down the list adding the road to a temporary list. At every step the "pruning process" starts on the selected roads cutting the isolated roads in order to get a connected subnetwork[4]. Therefore the number of nodes of the subnetwork increases in a discontinuous way, when the adding of a new road in the list allows to connect several previously selected roads. After several parametric scans, we found that the best result for our purposes is achieved by choosing about the 10% of the nodes in the whole Venice road network. In Fig 1 we show four consecutive selected subnetworks in the case of Carnival dataset to illustrate how the algorithm operates.

[3] Pendant nodes are unphysical solutions in our model since we are interested on the pedestrian mobility paths that bring people from one location to an other.

[4] Since we are interested on the largest connected component the *merging* parameter is off.

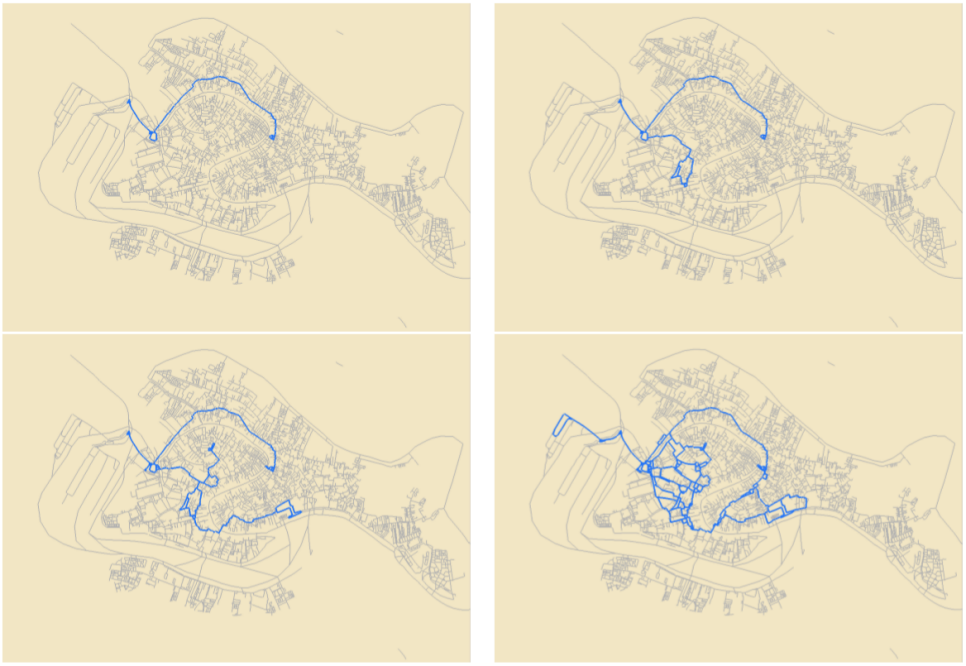


Figure 1: From top-left to right-bottom, we plot four mobility subnetworks with increasing number of roads, selected by the DNetPRO algorithm using the Carnival dataset.

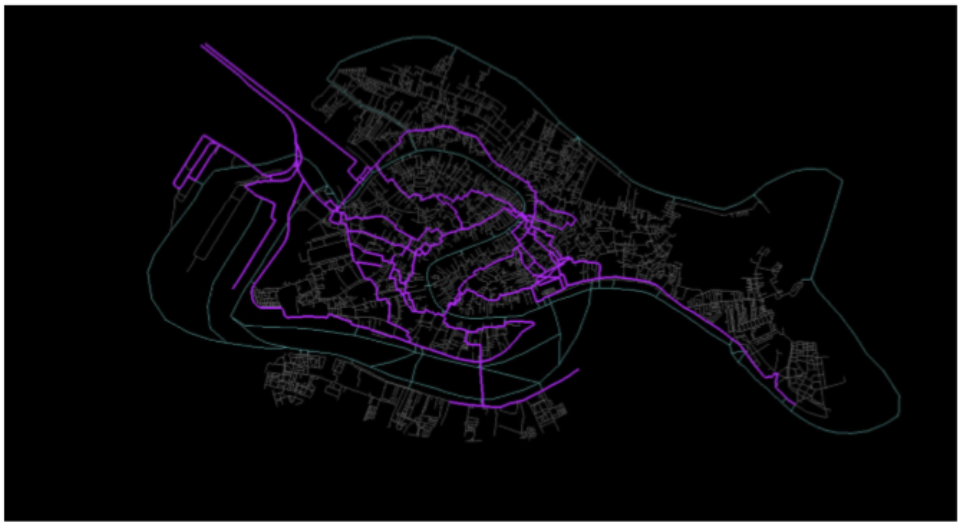
Using the DNetPRO algorithm we are able to extract a subnetwork which explains the 64% of the observed mobility using 13% of the total road network length for the case of the Carnival dataset and 15% of the total length in the case of the *Festa del Redentore* dataset.

The selected road subnetworks are plotted in Fig 2 for both the datasets. As a matter of fact, many of the highlighted paths are also suggested by Internet sites. However, we remark some differences that can be related by the different nature of the considered events. During the Carnival of Venice the mobility seems to highlight three main directions connecting the railway station and the *Piazzale Roma* (top-left in the map), which are the main access points to the Venice historical centre, with the area around San Marco square, where many activities where planned during 26/02/2017. In the case of the *Festa del Redentore* the structure is more complex due to the appearance of several paths connecting the station and *Piazzale Roma* with the *Dorsoduro* district in front of the *Giudecca* island.

This geometrical structure could have a double explanation: on one hand the *Festa del Redentore* introduces an attractive area near the *Giudecca* island, where the fireworks take place in the evening; on the other hand the *Festa del Redentore* is a festivity very much felt by the local population, that knows the Venice road network and performs alternative paths.

On these subnetwork we also map the mobility of Italians and foreigns separately. The results of this application are deeply discussed in the paper.

(a)



(b)

Figure 2: Picture (a): selected subnetworks (highlighted in purple) from the road network of the Venice historical centre (in the background), that explain 64% of the recorded mobility in the datasets. The top picture refers to the Carnival mobility during 26/02/2017 and corresponds to 13% of the total length of the Venice road network. The picture (b) refers to the *Festa del Redentore* mobility during 15/07/2017 and corresponds to 15% of the total length of the Venice road network.

Appendix C - BlendNet

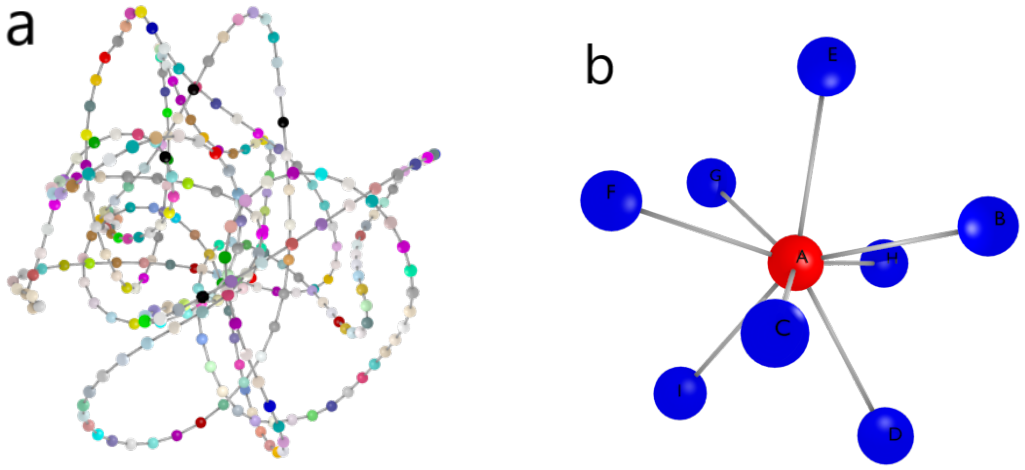


Figure 3: (**a**) Chain graph rendered by BlendNet software. Node colors are randomly generated by the tool. (**b**) Star graph rendered by BlendNet software. Node colors and labels are given as extra columns in node-list file.

Graph visualization is still an open problem in many applications. The problem is commonly related to large graph visualization in which problems arise from the rendering of a large number of nodes and a greater number of links between them (a graph with N nodes could have (NxN) possible links). An other open problem concern the multi-dimensional visualization of graphs. Despite common graph tools compute the node coordinates in any space dimensions (and clearly the maximum number of possible dimension for a visualization is only 3) the real visualization is often allowed only in 2D spaces. The counterpart of these problems concern a pretty visualization of the graphs that it is often ignored by many tools which prefer focusing on simple renderings.

In this section we introduce a new custom graph viewer developed for pretty small-network visualization in 2D and 3D, called BlendNet [4] (*Blender Network viewer*). BlendNet is an open-source project and it is released on Github under GPL license. All the small-graphs showed in this work are made using this tool and in particular the feature-signatures generated by the DNetPRO algorithm.

BlendNet is written in Python with the help of Blender API. Blender is now a standard for 3D rendering and it is commonly used in a wide range of graphical applications, starting from the simpler 3D dynamics to video-game applications. Blender is certainly more than a simple graphical viewer but it provide an easy Python interface and a wide on-line documentation which make it a useful tool for graphical representation of 3D structures.

We are forced to use the Python version provided by Blender to use its APIs and any extra-package required by our application have to be installed with the appropriated pip. We use the networkx Python library for node coordinates computation and thus we have to update our Python-Blender with the appropriated packages. Moreover, since the code can

be difficult to manage for non-expert users we have written an easy command-line interface to set the whole set of parameters required by the graph viewer that can be piloted via Makefile rules. The list of nodes and edges can be passed via command-line with relative filenames, in the same format of the concurrent graph viewers (e.g *Gephi* software, the other graph viewer used in this work to generate the larger network structures of the CHIMeRA project).

The software project is a single script file and it includes a full list of possible examples and usages. Some of this examples are shown in Fig. 3. A full list of installation instructions is also provided for any operative system (Unix, MacOS and Windows). These instructions cover a full installation of Blender, Python and BlendNet package for administrator and no-root users (ref. Shut project [6]). With slight code editing we can obtain different node coordinates and shapes. Node colors, sizes and positions can also be given using the node-list file as independent columns.

Appendix D - Multi-Class Performances

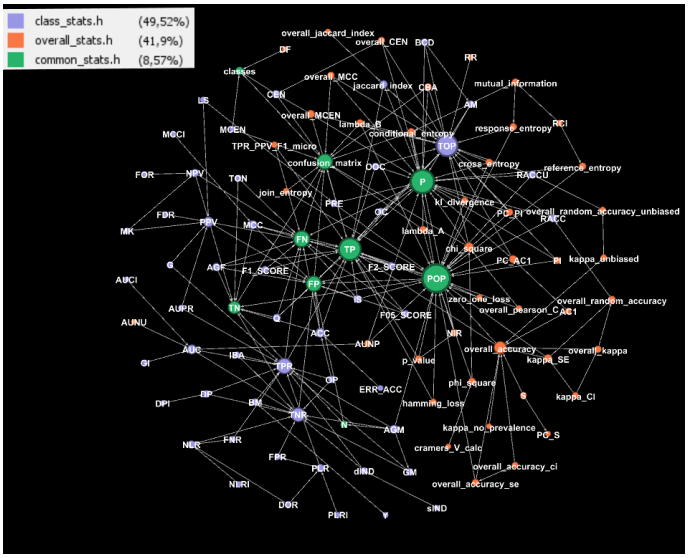


Figure 4: Multi class score interaction graph. Each node identifies a different performance evaluator and links are given by the interactions between mathematical formulations of each quantity. The graph has more than 100 nodes and more than 200 links. The node colors are given by the classes identified in the work of Sepand et al. [12].

The performances evaluation is a crucial task in any Machine Learning application. Given a set of patterns and its corresponding (true) labels we can evaluate the efficiency of a given model with a comparison between labels and model outputs, i.e the predicted labels. There are a lot of different score functions that can be computed and any of them evaluates some aspects of the model efficiency. Any paper author choses the score that better highlights the advantages of its model and it is difficult to move around this large zoo of indicators. Moreover, (it is quite a constant in scientific research) when a paper is send to a peer-review, in many cases the reviewers suggest to check if other performance indicators are good enough for the showed results. This means that a lot of large simulations should be performed again and the appropriated variables recomputed to obtain the requested scores.

At this point the main question is: are these scores totally independent one from each other? The brief answer is simply no. In a very interesting work of Sepand et al. [12] the authors show how we can compute a wide range of these scores starting from the evaluation of the simple confusion matrix[5], providing a full mathematical documentation

[5] The confusion matrix is a square matrix of shapes (N, N), with N the total number of classes in the

and references about their numerical evaluation.

Despite the Python code provided by Sepand et al. explains these links between the mathematical quantities they stop their analyses on the score evaluations without any interest on the optimization of these computations. Starting from their work we analyzed the inter-connections between these mathematical formulas and we extracted the dependencies between the involved variables. In particular, a score function can be interpreted as a node and its connections could be given by the variables needed to evaluate it. This type of graphs are commonly called *factor graphs*. In a mathematical formulation of *factor graphs* there are different kinds of nodes (variables and factors, or equations). The focus of our analysis was not on mathematical formalism of these kind of graphs but we aimed to a visualization of function interactions and an analysis of the numerical improvements derived from it.

In the work of Sepand et al. the authors identify three function classes: common statistics, class statistics and overall stats. In Fig. 4 the interaction graph of these three classes is showed. The figure shows deep interactions between the three function classes and it highlights the dependencies of the different quantities involved. We can also use this kind of visualization to formulate computational considerations about the order in which these quantities could be evaluated. Since the graph is a direct graph by definition, we can start from the root node (the node without links which bring to it) and cross the network until the leaf nodes (nodes without link which go out from them) like in a tree-graph. At each step of the percolation, the incoming nodes identify totally independent quantities. This independence means that the nodes-quantities can be potentially computed in parallel. To clarify this consideration we can reorganize the graph visualization minimizing the link lengths and obtaining a stratified graph in which each level identifies a potential parallel section. A graph with these properties was obtained using the dot visualization and it is showed in Fig. 5. As can be seen in figure we can identify 7 levels in the graph and thus 7 potential parallel regions for the computation of the full set of functions.

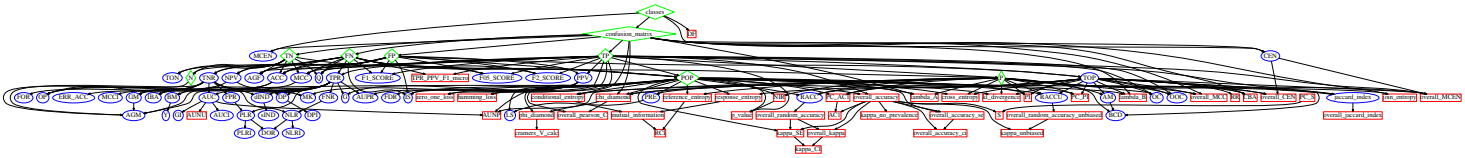


Figure 5: Re-organization of the graph in Fig. 4. The rendering was obtained using the dot visualization, i.e the minimization of the link lengths. The direct graph identifies the tree of dependencies and each level of the tree represents a set of independent functions that can be potentially computed in parallel. This graph is used as parallel scheme for the Scorer library.

These considerations allow us to create an optimized version of the code of Sepand et al., the Scorer library [7]. The Scorer library is the C++ porting of the PyCM library of Sepand et al. with a Cython wrap for the Python compatibility. Following the above told graph the computation of score quantities are performed in parallel according to the 7 levels found. The parallelization strategy chosen uses the section keywords of OpenMP library to perform no-wait tasks that are computed by each thread of the parallel region.

current problem, whose entries are the number of right and false classifications. In particular, each entry of the matrix represents the predicted instances in a given class. If the class is the right one we call it as true positive item. As counterpart we have a false positive item.

The extracted graph includes more than 100 different quantities so writing the full set of parallel sections becomes an hard (boring) work in C++. Moreover, update the graph with new quantities brings to a consequential update of the full code and also of the parallelization strategy. Each function was written as an anonymous-struct, i.e a functor, with an appropriate operator overloading. Each functor has a name given by a pre-determined regex (get_{function}) and the list of arguments follows the same nomenclature[6]. With these expedients we created a fully automated C++ script which parses the list of functors, it computes the dependency graph and the parallelization levels and it gives back a compilable C++ script with the desired characteristics. In this way we can guarantee an easy way to update the library and moreover we overcome the boring writing of a long code. The automatic creation script is provided in the Scorer library and it should be used at each pull request or version update.

For a pretty/useful visualization of the computed quantities we rendered the interaction graph in an HTML framework. In this way we can insert with a CSS table the computed values in each node that can be discovered passing the mouse over the figure. An example of this rendering is given in the on-line version of the library [7].

In conclusion the developed Scorer library is a very powerful tool for Machine Learning performances evaluation which can be used either in C++ either in Python codes through the Cython wrap. The code is automatically generated at each update and automatically tested using continuous integration for any platform using Travis CI and Appveyor CI[7]. The code can be compiled using CMakefile or Makefile and a setup.py is provided for the Python version. So when you write a new paper on Machine Learning and you do not know what could be the most appropriate indicator to show in your research or you are afraid that a referee could ask you to compute an other one there is only one solution: compute them all using Scorer.

[6] If the functor receives in input the variable A and B we have to ensures that two functors named get_A and get_B will be provided. The only exception is given by the root functor.

[7] We perform tests for Unix and Windows environments. We check more than 15 combinations of environments and compilers.

Appendix E - Neural Network as Service

One of the final goals of Machine Learning is certainly the process automation. We develop everyday complex models to perform tasks that should be automatically executed by a computer without human supervision. Neural Networks are classical mathematical tools used for these purposes and we have widely discussed about them in Chapter **??** of this work. Beyond Neural Network structures and purposes for which they are made there is still an uncovered topic to discuss: the automation of these kind of algorithms into a computer device. In this section we are going to discuss an implementation of these algorithms as service in a computer server. In particular we will talk about the implementation of the *FiloBlu* service which is part of a project developed in collaboration with the Sapienza University (Rome) and the INFN Data Center CNAF of Bologna. This work is still in progress and its purpose goes beyond the current topic, so we will focus only on the implementation of the service without any reference on the Machine Learning algorithm used. This is a further proof that the developed techniques are totally independent by the final application purpose.

A service is a software that is executed in background in a machine. In Unix machines it is often call daemons, while in Windows machine is called *Windows service*. A service starts only with administrator privileges and it goes on without any user presence. An other important requirement is the ability to restart when some troubles occur in the machine functionality and/or at the boot of the machine.

A Machine Learning service could be used for applications in which we have to manage an asynchronous stream of data for long time intervals. An example could be the case in which the data provider is identified by an App or a video-camera. These data should be stored inside a central database that can be located in a different device or in the same computer in which the service run. Since the service runs in background the only communication channel with the user is given by log files. A log file is a simple readable file in which are saved the base information about the current status of the service. Thus, it is crucial to set appropriated check-points into the service script and chose the minimum quantity of information that the service should write to make user-understandable its status.

FiloBlu Service

In the *FiloBlu* project we have a stream of data provided by an external App that are stored in a central database server. The Machine Learning service has to read the information stored into a database, it process them and finally it write the results into the same database. All these operations have to be performed with high frequency since the output results have to be shown in a real-time application. This frequency would be the clock-time of the process function, i.e at each time interval (as small as we like) the process task will be called and we will have the desired results in output. At the same time we have to take

care about the time required by our Machine Learning algorithm: not all the algorithms can process data in real time and the frequency of process function has to be less than the time required by the algorithm or we can lose some information.

We obtain the best efficiency from a service splitting as much as possible the required functionality in small-and-easy tasks. Small task can be evaluated as independent functions with an associated frequency that in this case can be reduced as much as possible. The *FiloBlu* required functionality can be reviewed as a sequence of 3 fundamental steps and other 2 optional ones: 1) read the data from the database, 2) process the data with the Machine Learning algorithm and 3) write the obtained results into the database; 4) update the Machine Learning model and 5) clear old log files are optional steps. To further improve the service efficiency we give each (independent) step to a different thread. The whole set of tasks are piloted by a master thread given by the service itself. In this way the service is computational efficient and moreover it does not weight on the computer performances. We have to take in mind that the computer which host the service has to be effected by the daemon process as less as possible either in memory either on computational efficiency. The last step is the synchronization of the previous tasks with appropriated clock frequencies.

Let's start from the data reading function. Since our data are assumed to be stored into a database, this function has to perform a simple query and extract the latest data inserted. Obviously the efficiency of the step is based on the efficiency of the chosen query. The data extracted are saved in a common container shared between the list of threads and thus it belongs to the master. The choice of an appropriated container is a second point to carefully take in mind. This container should be light an thread-safe to avoid thread concurrency. While the second request is implementation dependent, the first one can be faced on using a FIFO container[8]. In this way we can ensure that the application will save a fixed amount of data and it will not occupy large portion of memory (RAM).

The second task is identified by the Machine Learning function which processes the data. The algorithm takes the data from the FIFO container of the previous step (if there is) and it saves the results into a second FIFO container for the next step. The time frequency of the step is given by the time required by the Machine Learning algorithm.

The third step keeps the data from the second FIFO container (if there is) and it performs a second query (a writing one in this case) to the database. Also in this case the frequency is given by the efficiency of the chosen query.

The last two steps can be executed without press time requirements and they are useful only on a large time scale.

Each step performs its independent logging on a single shared file. If an error occurs the service logs an appropriated message and it saves the current log-file in a different location to prevent possible log-cleaning (optional step). Then the service restarts.

We implemented this type of service in pure Python and the code is public available on Github [5]. The developed service was customized according to the server requirements of the project[9]. We chose the Python language either for its simplicity in the code writing either for its thread native module which ensures a total thread-safety of each variable. Using a set of function decorators we are able to run each function (callback) in a separated-detached thread as required by the previous instructions. The project includes a documentation about its use (also for general applications) and it can be easily installed via setup.py. In the *FiloBlu* project we used a Neural Network algorithm written in Tensorflow as Machine Learning model. Tensorflow does not allow to run background process directly so the problem was overcame using a direct call to a Python script which performed the

[8] FIFO container, i.e *First-In-First-Out*, is a special data structure in which the first element added will be processed as first and then automatically removed from it.

[9] The FiloBlu service is a Windows service and it can not run on Unix machines. Moreover, the database used in the project is a MySQL one so the queries and the libraries used are compatible only with this kind of database.

full list of steps into an infinite loop. In this way the service could be restarted also if the process-service was killed. The service can be driven using a simple Powershell script provided in the project.

Data Transmission

In the above configuration we focused on the pipeline which process the stream of data ignoring any problem about the communication between the external device and the machine which host the service. The *FiloBlu* project uses an external App to send data to the main server, so we have two systems which have to communicate between them automatically via Internet connection. In general, we could manage sensitive data that could be vulnerable using an Internet communication. To face on this problem we developed a simple TCP/IP client-server package which also supports a RSA cryptography, the CryptoSocket package [8].

The communication security could be an important point in many research applications and a valid cryptography procedure is essential. The RSA cryptography is considered one of the most secure cryptography algorithm for data transmission and it is quite easy to implement. In the CryptoSocket package we implemented a simple wrap around the socket Python library to perform a serialization of our data which are (optionally) processed by our custom RSA algorithm. In this way different kind of data could be sent by the client at the same time. The client script could be adapted with slight modifications for any user need and also complex Python structures could be transmitted between two machines (to the server). The cryptography module was written in pure C++ for computational efficiency and a Cython wrap was provided for pure-Python applications. CryptoSocket has only demonstrative purpose and so it works only for a 1-by-1 data transmission (1 server and 1 client).

Since this second implementation could be used also for other applications it was treated as a separated project and it has its own open-source code. The CryptoSocket package can be installed via CMake in any platform and operative system and a full list of installation instructions is provided in the project repository. The continuous integration of the project is guaranteed by testing the package installation across multiple C++ compilers and platforms via Travis CI and Appveyor CI.

Appendix F - Bioinformatics Pipeline Profiling

In this work many times we have talked about the performances evaluation of a scripts in terms of time performances and other system statistics. The importance in the understanding the state of our infrastructure is essential not only for ensuring the reliability and stability of a software but also for a more efficiency use of the available resources. In particular about what concern the memory, CPUs and diskIO management is useful to know the required amount of each step of our software to perform the better parallelization strategy. Metrics represent the raw measurements of resource usage that are used by a software or a collection of them. These might be low-level usage summaries provided by the operating system, or they can be higher-level types of data tied to the specific functionality or work of a component. These kind of data could be collected and aggregated by a monitoring system like *Telegraf*[10]. In general, the difference between metrics and monitoring mirrors the difference between data and information. Monitoring takes metrics data, aggregates it, and presents it in various ways that allow humans to extract insights from the collection of individual pieces.

In this section we focused on the importance of software monitoring. In particular we will talk about a work conducted in collaboration with INFN-CNAF of Bologna about the monitoring and the performance evaluation of a bioinformatics pipeline across various computational environments [9].

In this work a previously published bioinformatics pipeline was reimplemented across various computational platforms, and the performances of its steps evaluated. The tested environments were: I) dedicated bioinformatics-specific server II) low-power single node III) HPC single node IV) virtual machine. The pipeline was tested on a use case of the analysis of a single patient to assess single-use performances, using the same configuration of the pipeline to be able to perform meaningful comparison and search the optimal environment/hybrid system configuration for biomedical analysis. Performances were evaluated in terms of execution wall time, memory usage and energy consumption per patient.

GATK-LODn pipeline

The pipeline used in this work, GATK-LODn, has been developed in 2016 by Do Valle et al. [10], and codifies a new approach aimed to Single Nucleotype Polimorphism (SNP) identification in tumors from Whole Exome Sequencing data (WES). WES is a type of "next generation sequencing" data [17, 1, 15], focused on the part of the genome that actually codifies proteins (the exome). Albeit known that non-transcriptional parts of the genome can affect the dynamic of gene expression, the majority of cancers inducing mutations are known to be on the exome, thus WES data allow to focus the computational effort on the most interesting part of the genome. Being the exome in human approximately 1% of the

[10] An automatic installation guide for Telegraf is provided in the Shut [6] project for any OS and also for no-root users.

	Coverage	No. of Reads	Read Length	BAM file size	NGS size
Whole genome	37.7x	975,000,000	115	82 GB	104 GB
Whole genome	38.4x	3,200,000,000	36	138 GB	193 GB
Exome	40x	110,000,000	75	5.7 GB	7.1 GB

Table 1: Typical dataset size for a single patient of different types of next generation sequencing. BAM file size refers to the size of the binary file containing the reads from the machine.

total genome, this approach helps significantly in reducing the number of false positives detected by the pipeline. The different sizes of next generation sequencing dataset are shown in Tab 1.

The GATK-LODn pipeline is designed to combine results of two different SNP-calling softwares, GATK [13] and MuTect [3]. These two softwares employ different statistical approaches for the SNP calling: GATK examines the healthy tissue and the cancerous tissue independently, and identifies the suspect SNPs by comparing them; Mutect compares healthy and cancerous tissues at the same time and has a more strict threshold of selection. In identifying more SNPs, GATK has a higher true positive calling than Mutect, but also an higher number of false positives. On the other end Mutect has few false positives, but often does not recognize known SNPs. The two programs also call different set of SNPs, even when the set size is similar. The pipeline therefore uses a combination of the two sets of chosen SNPs to select a single one, averaging the strictness of Mutect with the recognition of known variants of GATK.

The pipeline work-flow includes a series of common steps in bioinformatics analysis and in the common bioinformatics pipelines. It includes also a sufficient representative sample of tools for the performances statistical analysis. In this way the results extracted from the single steps analysis could be easily generalized to other standard bioinformatics pipelines.

With the increasing demand of resources from ever-growing datasets, it is not favorable to focus on single server execution, and is better to distribute the computation over cluster of less powerful nodes. The computational pipeline also has to manage a high number of subjects, and several steps of the analyses are not trivial to be done in a highly parallel way. Thus, the importance of system statistics management as the efficiency usage of available resources are crucial to reach a compromise between computational execution time and energy cost. For these reasons our main focus is on the performance evaluation of a single subject without using all the available resources, as these could be more efficiently allocated to concurrently execute several subjects at the same time. Due to the nature of the employed algorithms, not all steps can exploit the available cores in a highly efficient way: some scales sub-linearly with the number of cores, some have resource access bottleneck. Other tools are simply not implemented with parallelism in mind, often because they are the result of the effort of small teams that prefer to focus their attention on the scientific development side rather than the computational one.

Moreover in order to obtain an optimal execution of bioinformatics pipelines, each analysis step might need very different resources. This means that any suboptimal component of a server could act as a bottleneck, requiring bleeding edge technology if all the steps are to be performed on a single machine. Hybrid systems could be a possible solution to these issues, but designing them requires detailed information about how to partition the different steps of the pipeline.

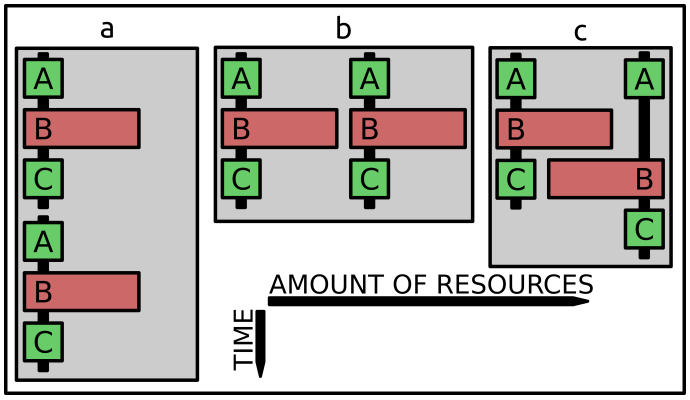


Figure 6: Examples of concurrency work-flow of two processes. The first case (a) represents a simple (naive) sequential work-flow; the second (b) highlights a brute force parallelization; the third (c) is the case of a perfect match between the available resources and the requested resources. Often brute force parallelization of pipelines done as in the image b ends up overlapping the most computationally intensive steps. Measuring the minimum viable requirements for the execution allow to better allocate resources as seen in the image c.

Computational Environments

There are two main optimization strategies: the first is to improve the efficiency of a single run on a single patient and the second is to employ massive parallelization on various samples. In both cases we have to know the necessary resources of the pipeline (and in a fine grain the resources of each step) and the optimal concurrency strategy to be applied to our work-flow (see Fig. 6). In the analyses we want to highlight limits and efficiencies of the most common computational environments used in big data analytics, without any optimization strategy of the codes or systems.

We also focused on a single patient analysis, the base case study to design a possible parallelization strategy. This is especially relevant for the multi-sample parallelization, that is the most promising of the two optimization strategies, as it does not rely on specific implementations of the softwares employed in the pipeline.

The pipeline was implemented on 5 computational environments: 1 server grade machine (Xeon E52640), 1 HPC node (Xeon E52683), 2 low power machines (Xeon D and Pentium J) and one virtual machine built on an AMD Opteron hypervisor. The characteristics of each node are presented in Tab. 2.

The server - grade node is a typical node used for bioinformatics computation, and as such features hundreds of GB of memory with multiple cores per motherboard: for these reasons we chose it as reference machine and the following results are expressed in relation to it.

The two low - power machines are designed to have a good cost - to - performance ratio, especially for the running cost[11]. These machines have been proven to be a viable solution for high performance computations [2]. Their low starting and running cost mean that a cluster of these machines would be more accessible for research groups looking forward to increase their computational power.

The last node is a virtual machine, designed to be operated in a cloud environment.

The monitoring tool used is *Telegraf*, which is an agent written in Go for collecting, processing, aggregating, and writing metrics. Each section of the pipeline sends messages to the *Telegraf* daemon independently.

[11] Running cost is evaluated as the energy consumption that the node requires per subject, assuming that the consumption scales linearly with the number of cores used in the individual step.

CLASS	**server grade machines**		**low power machines**		**virtual machine**
CPU	Intel Xeon	Intel Xeon	Intel Pentium	Intel Xeon	AMD Opteron
version	E5-2683v3	E5-2640v2	J4205	D-1540	6386 SE
Microarchitecture	Haswell	Ivy Bridge EP	Apollo Lake	Broadwell	Piledriver
Launch Date	Q3'14	Q3'13	Q4'16	Q1'15	Q3'12
Lithography	22 nm	22 nm	14 nm	14 nm	32 nm
Cores/threads	14/28	8/16	4/4	8/16	16
Base/Max Freq	2.00/3.00	2.00/2.50	1.50/2.60	2.00/2.60	2.80/3.50
L2 Cache	35 MB	20 MB	2 MB	12 MB	16 MB
TDP	120 W	95 W	10 W	45 W	115 W
Total CPUs	2	2	1	1	1
total cores/threads	28/56	16/32	4/4	8/16	16
Total Memory	256 GB	252 GB	8 GB	32 GB	60 GB
System power	240 + 60 W	190 + 60 W	10 + 2 W	45 + 10 W	115 + 10 W
Electrical costs	650 €/year	550 €/year	26 €/year	120 €/year	273€ /year
System price	4000-6000 €	3000-5000 €	100-130 €	900-1200 €	2000-3000€

Table 2: Characteristics of the tested computational environments. Electrical costs are estimated as 0.25 €/kWh; CPU frequencies are reported in GHz; TDP: Thermal Design Power, an estimation indicator of maximum amount of heat generated by a computer chip when a "real application" runs.

Regardless of the number of cores of each machine we restrict the number of cores used to only two to compare the statistics: this restriction certainly penalize the environment with multiple cores but with a view of maximizing the parallelizations and minimize the energy cost it is the playground to compare all the available environments. Another restriction is applied to the chosen architectures: since available low - power machines provides only x86 - architectures also the other environments are forced to work in x86 to allow the statistics comparison.

Pipeline steps

The pipeline steps that have been examined are a subset of all the possible steps: we only focus on those whose computational requirements are higher and thus require the most computational power. These steps are:

1. **mapping:** takes all the reads of the subjects and maps them on the reference genome;
2. **sort:** sorts the sequences based on the alignment, to improve the reconstruction steps;
3. **markduplicates:** checks for read duplicates (that could be imperfections in the experimental procedures and would skew the results);
4. **buildbamindex:** indexes the dataset for faster sorting;
5. **indexrealigner:** realigns the created data index to the reference genome;
6. **BQSR:** base quality score recalibration of the reads, to improve SNPs detection;
7. **haplotypecaller:** determines the SNPs of the subject;
8. **hardfilter:** removes the least significant SNPs.

The following statistics were evaluated:

1. **memory per function:** estimate percentage of the total memory available to the node used for each individual step of the pipeline;
2. **energy consumption:** estimated as the time taken by the step, multiplied by the number of cores used in the step and the power consumption per core (TDP divided by the available cores). As mentioned before this normalization unavoidably penalize the multi-core machines but give us a term of comparison between the different environment;
3. **elapsed time:** wall time of each step.

The pipeline was tested on the patient data from the 1000 genome project with access code NA12878, sample SRR1611178. It is referred as a Gold Standard reference dataset [16]. It is generated with an Illumina HiSeq2000 platform, SeqCap EZ Human Exome Lib v3.0 library and have a 80x coverage. As Gold Standard reference it is commonly used as benchmark of new algorithm and for our purpose can be used as valid prototype of genome.

Results

Memory occupation is one of the major drawbacks of the bioinformatics pipelines, and one of the greater limits to the possibility of parallel computation of multiple subjects at the same time. As it can be seen in Fig. 7, the memory occupation is comprised between 10% and 30% on all the nodes. This is due to the default behavior of the GATK libraries to reserve a fixed percentage of the total memory of the node. The authors could not find any solution to prevent this behavior from happening. As it can be noticed, in the node with the greatest amount of total memory (both Xeon E5 and the virtual machine) the requested memory is approximately stable, as is always sufficient for the required task. The memory allocation is less stable in the nodes with a limited memory (Xeon D and Pentium J), as GATK might requires more memory than what initially allocated to perform the calculation. The exception to this behavior is the *mapping* step, that uses a fixed amount of memory independently from the available one (between 5 and 7 GB). This is due to the necessity of loading the whole human reference genome (version hg19GRCh37) to align each individual read to it. All the other steps do not require the human reference genome but can work on the individual reads, allowing greater flexibility in memory allocation.

As can be seen in Fig. 8 and Fig. 9, this increase of memory consumption does not correspond to a proportional improvement of the time elapsed in the computation.

The elapsed time for each step and for the whole pipeline can be seen in Fig. 8. It can be seen that there is a non consistent trend in the behavior of the different environments. Aside from the most extreme low power machine, the pentium J, the elapsed times are on average higher for the low power and slightly higher for the cloud node, but the time for the individual rule can vary. In the sorting step, Pentium J is 20 times slower than the reference. This is probably due to the limited cache and memory size of the pentium J, that are both important factors determining the execution time of a sorting algorithm and are both at least four to six times smaller than the other machines. The HPC machine, the Xeon E52683, is consistently faster than the reference node.

The energy consumption per step can be seen in Fig. 9. The low power machines are consistently less than half the baseline consumption. Even considering the peak of consumption due to the long time required to perform the sorting, the most efficient low power machine, the pentium J, consumes 40% of the reference, and the Xeon D consumes

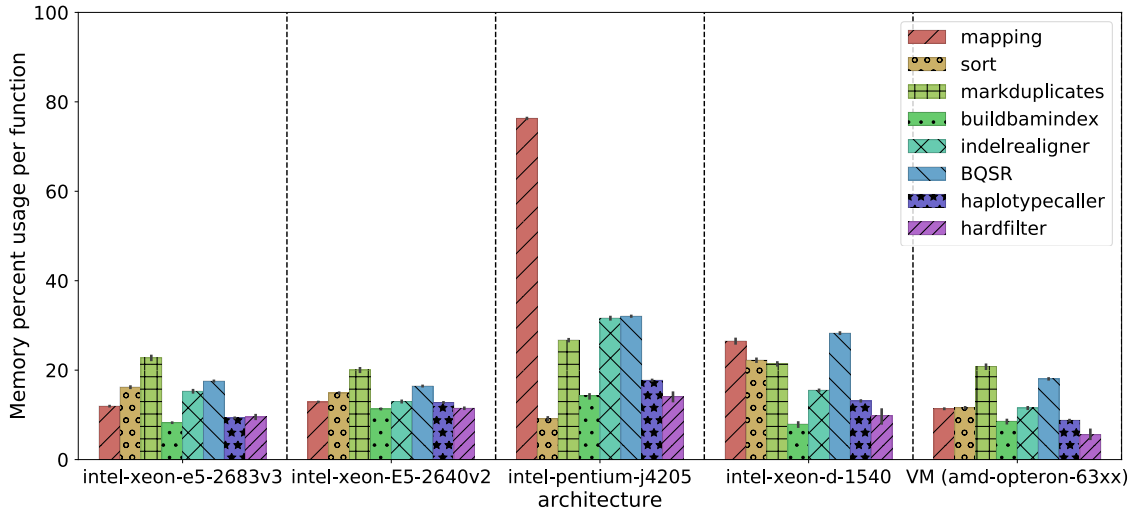


Figure 7: Memory used for each step of the pipeline. Due to the GATK memory allocation strategy, all steps use a baseline amount of memory proportional to the available memory. Smaller nodes, like the low power ones, require more memory as the baseline allocated memory is not sufficient to perform the calculation.

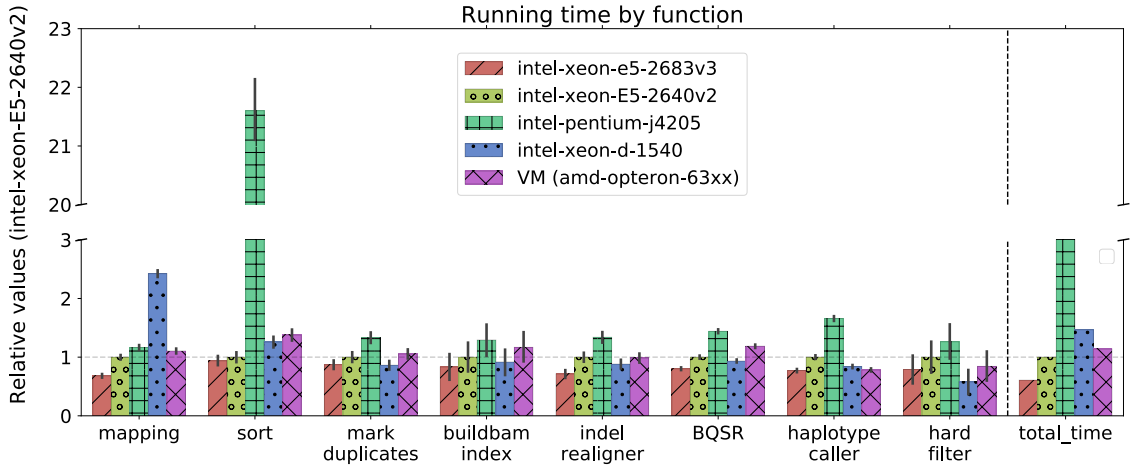


Figure 8: Time elapsed per step of the pipeline, and total elapsed time. In the sorting step, Pentium J is 20 times slower than the reference, probably due to the limited cache size.

60% of the reference. The HPC machine, the Xeon E52683, have consumption close to the low power nodes, balancing out the higher energy consumption with a faster execution speed. The virtual machine has the highest consumption despite the fact that the execution time of the whole pipeline is comparable to the reference due to the high TDP compared to its execution time.

Conclusions

Bioinformatics pipelines are one of the most important uses of biomedical big data and, at the same time, one of the hardest to optimize, both for their extreme requisites and the constant change of the specification, both in input-output data format and program API.

This makes the task of pipeline optimization a daunting one, especially for the final target of the results; physicians and biologists could lack the technical expertise (and time)

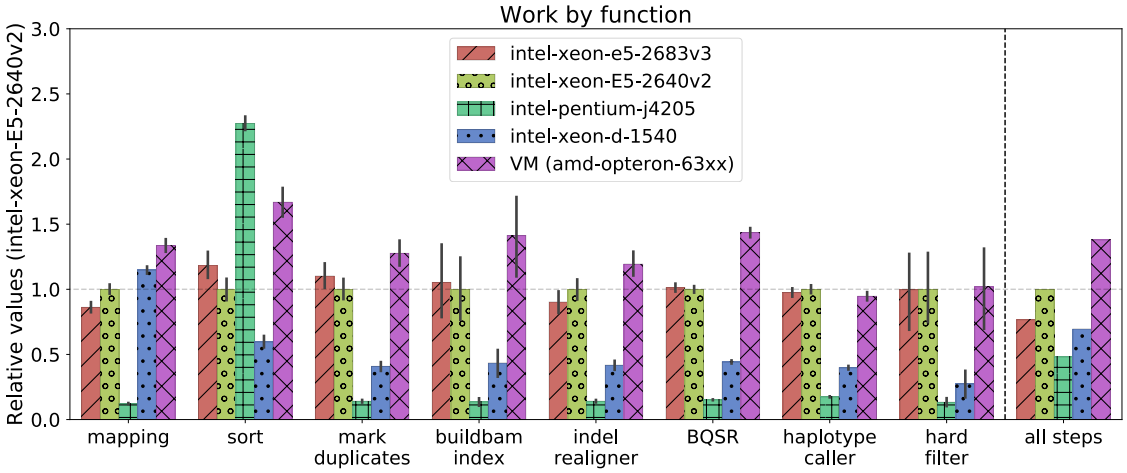


Figure 9: Energy consumption per pipeline step and on the whole pipeline. Energy consumption is estimated as the time taken by the step, multiplied by the number of cores used in the step and the power consumption per core (TDP divided by the available cores).

required to optimize each new version of the various softwares of the pipelines. Moreover, in a verified pipeline updating the software included without a long and detailed cross-validation with the previous one is often considered a bad practice: this means that often these pipelines are running with under-performing versions of each software.

Clinical use of these pipelines is growing, in particular with the rise of the concept of *personalized medicine*, where the therapy plan is designed on the specific genotype and phenotype of the individual patient rather than on the characteristic of the overall population. This would increase the precision of the therapy and thus increase its efficacy, while cutting considerably the trial and error process required to identify promising target of therapy. This requires the pipelines to be evaluated in real time, for multiple subjects at the same time (and potentially with multiple samples per subject). To perform this task no single node is powerful enough, and thus it is necessary to use clusters. This brings the need to evaluate which is the most cost and time efficient node that can be employed.

In the cost assessment there are several factors that need to be considered aside of the initial setup cost, namely cost for running the server and opportunity cost for obsolescence. Scaled on medium sized facilities, such the one that could be required for a hospital, this cost could quickly overcome the setup cost. This cost does also include not only the direct power consumption of the nodes, but also the required power for air conditioning to maintain them in the working temperature range. Opportunity costs are more complex, but do represent the loss of possibility of using the most advanced technologies due to the cost of the individual node of the cluster. Higher end nodes require a significant investment, and thus can not be replaced often.

With this perspective in mind, we surmise that energy efficient nodes present an interesting opportunity for the implementation of these pipelines. As shown in this work, these nodes have a low cost per subject, paired with a low setup cost. This makes them an interesting alternative to traditional nodes as a workhorse node for a cluster, as a greater number of cores can be bought and maintained for the same cost.

Given the high variability of the performances in the various steps, in particular with the sorting and mapping steps, it might be more efficient to employ a hybrid environment, where few high power nodes are used for specific tasks, while the bulk of the computation is done by the energy efficient nodes. This is true even for those steps that can be massively parallelized, such as the mapping, as they benefit mainly from a high number of processors rather than few powerful ones. In this work we focused only on CPUs computation,

but another possibility could be an hybrid-parallelization approach in which the use of a single GPU accelerator can improve the parallelization of the slower steps. Each pipeline work-flow requires its own analyses and tuning to reach the best performances and the right parallelization strategy based on the use which it is intended but a low energy node approach is emerging as a good alternative to the more expensive and common solutions.

Bibliography

[1] S. Behjati and P. S. Tarpey. What is next generation sequencing? *Archives of disease in childhood - Education & practice edition*, 98(6):236–238, 2013.

[2] D. Cesini, E. Corni, A. Falabella, A. Ferraro, L. Morganti, E. Calore, S. F. Schifano, M. Michelotto, R. Alfieri, R. De Pietri, T. Boccali, A. Biagioni, F. Lo Cicero, A. Lonardo, M. Martinelli, P. S. Paolucci, E. Pastorelli, and P. Vicini. Power-efficient computing: Experiences from the cosa project. *Scientific Programming*, 2017, 2017.

[3] K. Cibulskis, M. S. Lawrence, S. L. Carter, A. Sivachenko, D. Jaffe, C. Sougnez, S. Gabriel, M. Meyerson, E. S. Lander, and G. Getz. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature Biotechnology*, 31(3):213–219, 2013.

[4] N. Curti. BlendNet: Blender network viewer. https://github.com/Nico-Curti/BlendNet, 2019.

[5] N. Curti. FiloBlu: Machine learning as service. https://github.com/Nico-Curti/FiloBlue, 2019.

[6] N. Curti. Shut: Shell utilities for no-root users. https://github.com/Nico-Curti/Shut, 2019.

[7] N. Curti and D. Dall'Olio. Scorer: Machine learning scorer library. https://github.com/Nico-Curti/Scorer, 2019.

[8] N. Curti and A. Fabbri. Cryptosocket - tcp/ip client server with rsa cryptography. https://github.com/Nico-Curti/CryptoSocket, 2019.

[9] N. Curti, E. Giampieri, A. Ferraro, C. Vistoli, E. Ronchieri, D. Cesini, B. Martelli, C. Duma Doina, and G. Castellani. Cross-environment comparison of a bioinformatics pipeline: Perspectives for hybrid computations. *Springer, Cham, Euro-Par 2018: Parallel Processing Workshops*, 11339, 2019.

[10] Í. F. do Valle, E. Giampieri, G. Simonetti, A. Padella, M. Manfrini, A. Ferrari, C. Papayannidis, I. Zironi, M. Garonzi, S. Bernardi, M. Delledonne, G. Martinelli, D. Remondini, and G. Castellani. Optimized pipeline of MuTect and GATK tools to improve the detection of somatic single nucleotide polymorphisms in whole-exome sequencing data. *BMC Bioinformatics*, 17(S12):341, 2016.

[11] K. Fukunaga. *Introduction to Statistical Pattern Recognition (2Nd Ed.)*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[12] S. Haghighi, M. Jasemi, S. Hessabi, and A. Zolanvari. PyCM: Multiclass confusion matrix library in python. *Journal of Open Source Software*, 3(25):729, may 2018.

[13] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–1303, 2010.

[14] C. Mizzi, A. Fabbri, S. Rambaldi, F. Bertini, N. Curti, S. Sinigardi, R. Luzi, G. Venturi, M. Davide, G. Muratore, A. Vannelli, and A. Bazzani. Unraveling pedestrian mobility on a road network using icts data during great tourist events. *EPJ Data Science*, 7(1):44, Oct 2018.

[15] J. Shendure and H. Ji. Next-generation DNA sequencing. *Nature Biotechnology*, 26(10):1135–1145, 2008.

[16] J. M. Zook, B. Chapman, J. Wang, D. Mittelman, O. Hofmann, W. Hide, and M. Salit. Integrating human sequence data sets provides a resource of benchmark snp and indel genotype calls. *Nature Biotechnology*, 32, 2014.

[17] M. Zwolak and M. Di Ventra. Colloquium: Physical approaches to DNA sequencing and detection. *Reviews of Modern Physics*, 80(1):141–165, 2008.