Universitat Autònoma de Barcelona

MACHINE LEARNING TECHNIQUES IN BIOINFORMATICS: FROM DATA INTEGRATION TO THE DEVELOPMENT OF APPLICATION-ORIENTED TOOLS

Valentin Junet

Machine Learning techniques in bioinformatics:

From data integration to the development of application-oriented tools

Thesis for the degree of Doctor in Bioinformatics

Directors:
Prof. Dr. Xavier Daura
Dr. José Manuel Mas

Candidate:
Valentin Junet

2021
Programa de Doctorat en Bioinformàtica
Institut de Biotecnologia i de Biomedicina
Universitat Autònoma de Barcelona

A ma famille, merci pour votre soutien continu

Acknowledgement

I am grateful to my supervisors, Judith Farrés, José Manuel Mas and Xavier Daura and the Anaxomics team, in particular José-Luis Ruiz, for their help and advices during these years. I also would like to thank Doctor Rafael Morales for his contribution to the SICPAC study and Malu Calle for useful discussions during the preparation of CuBlock's manuscript.
This work was funded as part of the COSMIC project from the European Unions Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 765158. I am thankful for the funding that allowed me to conduct this research and to the COSMIC consortium for being part of an exciting innovation programme.

J'aimerais tout particulièrement remercier ma famille, mes parents, grand-parents, mon frère et mes proches, Vanja, Laura qui m'ont continuellement soutenu. Merci d'être là, dans ma vie.

Summary

Recent developments in Machine Learning aim at automatizing available methods, rendering them universal while requiring as little expert-knowledge as possible. In this thesis, we will take a step back. We will focus on the data, their specific needs and how to extract meaningful information out of them. This will be done through the presentation of different works highlighting various aspects to consider when developing Machine Learning techniques in bioinformatics.

There cannot be any models without the appropriate considerations on the data. Therefore, in the first part, we will put the models aside and focus on data integration. In more detail, we will present an algorithm for the normalization of gene-expression microarray data across different platforms. Microarray data are widely available in public repositories and such methods enable their subsequent downstream analysis.

In the next part, we will consider peptide sequence data and present a tool for the extraction of patterns in such sets. The model, based on convolutional neural networks, is open-source and can be used for peptide MHC-class II binding prediction among other applications.

The last part will be dedicated to the analysis of clinical data. We will present a retrospective cohort study on pancreatic cancer. For this study, a tool for the prediction of clinically relevant outcomes has been developed.

From data integration to the development of application-oriented tools, the three parts forming this thesis will be self-contained and will each address different challenges in the realm of data-driven approaches in bioinformatics.

Contents

Introduction

Machine Learning (ML) algorithms are data-driven approaches that have found numerous applications in the field of bioinformatics (Baldi and Brunak, 1998; Bhaskar *et al.*, 2006; Olson *et al.*, 2017; Shastry and Sanjay, 2020). How to deal and learn from an ever-growing amount of biomedical data remains a constant challenge in this field. The main objective of such algorithms is to find meaningful patterns and to make accurate predictions from the knowledge extracted from the data. It is therefore application-oriented and covers many domains in biology such as genomics, proteomics, systems biology, evolution and text mining (Larrañaga *et al.*, 2006).
We will start this introduction with a brief overview of some ML concepts and techniques. Then we will dive into the subject of this thesis, starting with a small discussion on automated ML to then highlight the needs that specific data sets might require. This will be followed by a short discussion on data repository and methods proposed in this thesis for particular types of data, namely sequence and clinical data.

0.1 Machine Learning: basic concepts and techniques

ML can be divided in 2 main categories:

- supervised learning: typically for classification/regression problems; the possible outcomes are known and a model is trained with labelled data.
- unsupervised learning: typically for clustering problems. The data are unlabelled and the algorithm looks for hidden patterns.

For the purpose of this thesis, we will focus on supervised learning. A typical pipeline for such algorithms is as follows:

- data preprocessing
- feature selection
- learning algorithm
- validation and evaluation

0.1.1 Data preprocessing

The data preprocessing contains any pre-steps such as gathering and organizing the data, cleaning them (handling missing values, removing redundant features, detecting outliers,...) and possibly performing feature engineering (note that some of these steps might also be referred to as data wrangling). The latter consists of generating features from raw data that will be used by the predictive model. For example, a categorical feature with values A, B and C cannot necessarily be used as such by an algorithm. If there is an order between the categories (for example A, B and C are grades), a possible solution would be to assign the numbers 1, 2 and 3 to the categorical values. A more general solution would be to divide the categorical feature into three binary features and set A=(1,0,0), B=(0,1,0) and C=(0,0,1).

0.1.2 Feature selection

Feature selection is important to improve the performance of the model and to extract information (Saeys *et al.*, 2007; Haury *et al.*, 2011). In some cases, the main objective is to identify biomarkers and the model itself is only relevant to select a good set of features. Feature selection methods are usually divided in two main categories (Bhaskar *et al.*, 2006): filter and wrapper methods. Filter methods are independent of the learning algorithm while wrapper methods select features based on the performance of the learning algorithm.

0.1.3 Learning algorithm

The learning algorithm is trained on the prepared data set. In the case of classification, it is optimized to predict the correct class based on the set of features. There exist numerous methods; we will present here a quick overview of 4 different types of algorithms.

Logistic regression

Logistic regression (Larrañaga *et al.*, 2006) is similar to linear regression, with the difference that the logistic function ($f(x) = 1/(1 + \exp(-x))$) is applied to the predicted outcome. The algorithm will optimize the search for parameters $\beta_0, \cdots, \beta_n$ such that the predictions best fit the training outcome. For a sample $x = (x_1, \cdots, x_n)$, the predicted outcome y will be

$$y = \frac{1}{1 + \exp(-(\beta_0 + \sum_{i=1}^{n} \beta_i x_i))}$$

and has values between 0 and 1. It is usually interpreted as a probability. The predictions are turned into a binary classification by choosing a threshold c (typically $c = 0.5$) and dividing them into two classes: $y < c$ and $y > c$. In Boughorbel *et al.* (2017), the authors propose a method to find an optimal cut-off c for classification.

Support vector machines

Support vector machines (Cortes and Vapnik, 1995) map the data into a high dimensional space to find a hyperplane separating the classes. In practice, to avoid computations in a high dimensional space, the kernel trick is used. The idea is that the data are represented by their pairwise similarity in the high dimensional transformed space rather than their exact positions. This similarity is determined by the kernel function.
The algorithm makes use of support vectors, which are training samples, to find the hyperplane with the largest margin, i.e. the largest distance from the hyperplane. For a new instance x, the predicted class y (either positive or negative) is determined by

$$y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i G(x, x_i) + \alpha_0\right)$$

where sign is the signature function (it is 1 for a positive value and -1 for a negative value), $G(a, b)$ is the kernel function, $x_1, \cdots, x_n$ are the training samples with their corresponding classes $y_1, \cdots, y_n$ and $\alpha_0, \cdots, \alpha_n$ are parameters determined during the optimization. The support vectors are those with non-vanishing α.

Neural network

Artificial neural networks are networks of neurons (Larrañaga *et al.*, 2006). The neurons are placed in layers that are connected to each other. The process of reading these layers is called the *feed-forward pass*. In its simplest case, called *perceptron*, there are two layers: the input and output layers. The neurons of the input layer are the features of the input data $x_1, \cdots, x_n$ and the output layer consists of one neuron (the output prediction y).

$$y = h\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

where $w_0, \cdots, w_n$ are the weights connecting the neurons that are determined during the optimization process and $h(x)$ is the activation function (usually scaling the outcome between 0 and 1 or -1 and 1).

Descision tree
Descision trees are rather intuitive. They consist of nodes with branches that correspond to a statement. If the statement is true, one follows the branch to the next node and so on until an end node is reached. The end nodes represent the prediction. Trees have the advantage of allowing categorical like continuous variables. Despite their simplicity, they can be very accurate when used in ensemble methods. Random forests (Breiman, 2001; Yang *et al.*, 2009) are ensemble of trees and the average of the outcome of all single trees is used for the prediction. In random forests, the trees are constructed independently using random methods. Another powerful family of algorithms using ensemble of trees are gradient boosting methods (Freund and Schapire, 1999; Friedman, 2001; Natekin and Knoll, 2013; Chen and Guestrin, 2016). As opposed to random forests, such methods construct trees based on previously constructed trees and each of them weight differently depending on the optimization process.

0.1.4 Validation and evaluation

A common issue with learning algorithms is overfitting the training data. This happens when the trained model gives excellent performances on the training data but poor ones on new, never-seen, data; in other words, the model doesn't generalize. The learning algorithm should be able to learn enough to fit training data but not too much so that it doesn't only fit the training data. Finding a good learning balance is a key issue in tuning an algorithm; this is known as the bias-variance dilemma. If an algorithm doesn't learn enough from the data and yet is still used to make prediction, it means that it relies on possibly erroneous assumptions, i.e. it is highly biased and might miss relevant information (underfitting). On the other hand, if the algorithm learns too much from the training data, it will be highly sensitive to minor fluctuations in the training data, i.e. it has a high variance (overfitting). For this reason, it is very important to validate a learning algorithm. There are several standard validation methods. We will briefly introduce the two most standard ones: k-fold cross-validation and bootstrapping. In any case, the important aspect is to perform the validation on a subset of data that hasn't been used for the training. It is therefore important for the feature selection step to be performed within the validation method. Indeed, if the features are selected using the whole data set, the algorithm has knowledge that these features are appropriate for all data, including the ones left out for validation. This can lead to overfitting.

Cross-validation
Cross-validation is a resampling method without replacement (Kohavi, 1995). The training data set is divided into k folds, one fold is left out for validation while the other (k-1) folds are used for the training. The procedure is repeated until each fold has been left out for validation.

Bootstrapping
Bootstrapping is a resampling method with replacement (Kohavi, 1995). The procedure consists of sampling with replacement a fixed amount of samples (typically the total number of samples), use the sampled data set for training and use the remaining samples for validation. This procedure is repeated a fixed number of times. The score of the validation with the whole training data is also considered. The final predictive score is a combination of the average score on the samples left out for validation and the training score. In more detail, if the number of samples is n, for b different samplings with replacement of n samples, the score sc_{boot} will be

$$sc_{\text{boot}} = 0.368 sc_{\text{train}} + \frac{0.632}{b} \sum_{i=1}^{b} sc_i$$

where sc_{train} is the score of the whole training set, sc_i the score of the i^{th} validation set and $0.368 \approx (1 - \frac{1}{n})^n$ is the probability for a sample not to be chosen for training (for large n).

These validation methods can also be used to select parameters and, more generally, any aspect of the final model -such as selecting the feature selection method, the learning algorithm, the data transformation and their optimal combination leading to the best validation results. This ”best model” selection bias can also lead to overfitting (Tsamardinos *et al.*, 2018). It is therefore important to also use, when possible, an external data set to perform an independent evaluation.

0.2 Automated Machine Learning

An important objective in the deployment of application-oriented ML models in bioinformatics is the development of fully automated workflows, which can generalize on most data sets while requiring little to no technical knowledge. This approach is known as Automated Machine Learning (AutoML) and there exist already various implementations, for example the open-source package *auto-sklearn* (Feurer *et al.*, 2021), or services available in cloud platforms (e.g. Microsoft Azure, AWS, Google Cloud Platform, IBM Watson Studio). While AutoML has proven to work well in competitions (Feurer *et al.*, 2018), it cannot be applied in a fully automated way to all types of data sets. Many biological data sets have few samples and a lot of dimensions requiring some particular considerations. For example, if an AutoML approach assesses thousands of models, it is possible for the best validation result to overfit the training data (Tsamardinos *et al.*, 2018). To address bioinformatics needs, a bio-oriented AutoML platform called JADBio (`https://jadbio.com/`) has been created (Tsamardinos *et al.*, 2020). While these platforms can be of great help and save time and effort, they cannot be automatically applied on any data sets. For example, the application of JADBio to a speech classification problem (Simantiraki *et al.*, 2017) required expert knowledge for the transformation of the raw data into a structured set that could be used as the input of the automated system. Similar concerns arise when analysing sequence data; raw series of letters cannot be directly interpreted by such systems.

0.3 Objectives

The objectives of the present thesis on ML techniques in bioinformatics will revolve around the data themselves. How to retrieve and prepare data sets for subsequent analysis? How to engineer the features in order to apply ML algorithms? How to design a method or adapt existing ones for the specifics of a given data set?

Unlike AutoML techniques, the aim will not be to provide a self-contained tool that could address all of these questions but to show concrete applications through different projects. These questions won't find an eloquent universal answer in this thesis; they will however be addressed in light of the needs required by specific data sets.

0.4 Data repository

Data are the key to a good ML model. Public repositories, databases (Bateman, 2007; Wren and Bateman, 2008; Galperin *et al.*, 2016) contain valuable information that can be extracted through appropriate ML techniques. The web page `https://www.oxfordjournals.org/nar/database/c/` contains a summary of the databases from the journal *Nucleic Acids Research*.

In the next two chapters, we will introduce two databases: the Gene Expression Omnibus (GEO, `https://www.ncbi.nlm.nih.gov/geo/`) and the Immune Epitope Database (IEDB, `http://www.iedb.org/`).

The GEO database contains a large amount of gene-expression microarray data. These data are rich in gene-expression information, from which meaningful patterns can be extracted. A challenging aspect is the integration of the different sets deposited in GEO. To address this issue, we present in Chapter 1 a novel algorithm called CuBlock (Junet *et al.*, 2021) for the normalization of gene-expression microarray data across different platforms. CuBlock enables the integration of the data available in GEO (and other gene-expression microarray data repositories) with the aim of subsequently applying ML techniques to extract meaningful expression patterns. CuBlock has been published in *Bioinformatics* (Junet *et al.*, 2021); Chapter 1 is essentially a restructured version of the published article.

0.5 Sequence data

The IEDB repository contains experimental data on antibody and T-cell epitopes. The analysis of biological sequences has an important impact in immunoinformatics. The availability of such data is key to the development of ML predictive techniques in the field of reverse vaccinology (Sette and Rappuoli, 2010) and epitope-based vaccine design (Parvizpour *et al.*, 2020). In Chapter 2, we present a tool called CNN-PepPred for the discovery of patterns in peptide sets. We focus on the application of this algorithm for the *in-silico* prediction of MHC class II-restricted T-cell epitopes. The tool is based on an ensemble of convolutional neural networks which architecture was specifically designed for this type of data. CNN-PepPred is open-source, under the Apache-2.0 license and is available in the repository `https://github.com/ComputBiol-IBB/CNN-PepPred`.
Chapter 2's content is a mixture between an application note which has been accepted for publication in the journal Bioinformatics (Junet and Daura, 2021) and the User's guide which also contains an extension of the tool to allow training with transfer learning.

0.6 Clinical data

To this point, we will have been familiar with a specific solution to the problem of gathering and preparing gene-expression microarray data for downstream analysis and we will have seen how ML techniques can be adapted to the specific needs of peptide sets. In the last part, Chapter 3, we will dive in a different type of data, namely clinical data.
According to a roundtable on value & science-driven health care from the Institute of Medecine (US) (`https://www.ncbi.nlm.nih.gov/books/NBK54290/`), clinical data are any type of information which are *determinants of health*, *measures of health and health status* or *documentation of care delivery*.
Clinical data are relevant for *in-silico* clinical trials (ISCT). The biotech company Anaxomics (`https://www.anaxomics.com/`) uses so-called TPMS (Therapeutic Performance Mapping System) technology (Jorba *et al.*, 2020) for ISCT. This technology generates a protein activation network representing the state of a patient using knowledge extracted from clinical information, drug effectors and known relations between proteins.
In Chapter 3, we present the SICPAC study, a retrospective study in pan-

creatic cancer. Using the clinical information (analytical measures, adverse events, co-treatments) of a patient during multiple visits at the hospital, we present a tool for the prediction of several outcomes during the following visit. The different predicted outcomes are the concentration of leukocyte, monocyte, hemoglobin, red blood cell, eosinophil and platelet.
Decision trees were selected as the core learning algorithm of this method due to their interpretable nature and the easy inclusion of categorical variables alongside continuous ones.
To gain molecular information, patients were simulated using the TPMS technology with the available clinical information and the method was applied including the protein activation values.
The tool is available as a web-application: `http://sicpac.anaxomics.com:81`. Care must be taken when using such tool. The data used for training were specific to the context (patients from a unique hospital, under a fixed treatment, treated by the same physician), the sample size was small (20 patients) and the results don't justify a reliable predictive tool (as noted in Chapter 3). The following persons contributed to the SICPAC study: Pedro Matos-Filipe, J.M. Garca-Illarramendi, Esther Ramírez, Baldomero Oliva, Judith Farrés, Xavier Daura, José Manuel Mas and Rafael Morales.

Through the work presented in the three chapters forming the main body of this thesis, concrete, application-oriented problems are addressed with data-driven approaches, contributing to the development of ML techniques in bioinformatics.

Chapter 1

CuBlock: A cross-platform normalization method for gene-expression microarrays

1.1 Abstract

Motivation: Cross-(multi)platform normalization of gene-expression microarray data remains an unresolved issue. Despite the existence of several algorithms, they are either constrained by the need to normalize all samples of all platforms together, compromising scalability and reuse, by adherence to the platforms of a specific provider, or simply by poor performance. In addition, many of the methods presented in the literature have not been specifically tested against multi-platform data and/or other methods applicable in this context. Thus, we set out to develop a normalization algorithm appropriate for gene-expression studies based on multiple, potentially large microarray sets collected along multiple platforms and at different times, applicable in systematic studies aimed at extracting knowledge from the wealth of microarray data available in public repositories; for example, for the extraction of Real-World Data to complement data from Randomized Controlled Trials. Our main focus or criterion for performance was on the capacity of the algorithm to properly separate samples from different biological groups.
Results: We present CuBlock, an algorithm addressing this objective, together with a strategy to validate cross-platform normalization methods. To validate the algorithm and benchmark it against existing methods, we used two distinct data sets, one specifically generated for testing and standard-

ization purposes and one from an actual experimental study. Using these data sets, we benchmarked CuBlock against ComBat (Johnson *et al.*, 2007), UPC (Piccolo *et al.*, 2013), YuGene (Lê Cao *et al.*, 2014), DBNorm (Meng *et al.*, 2017), Shambhala (Borisov *et al.*, 2019) and a simple $\log_2$ transform as reference. We note that many other popular normalization methods are not applicable in this context. CuBlock was the only algorithm in this group that could always and clearly differentiate the underlying biological groups after mixing the data, from up to six different platforms in this study.
Availability: CuBlock can be downloaded from `https://www.mathworks.com/matlabcentral/fileexchange/77882-cublock`

1.2 Introduction

Since the first whole-genome microarray study of gene expression was published in 1997 (Schena *et al.*, 1995; Lashkari *et al.*, 1997), high-throughput gene-expression microarrays have been a standard in many experimental designs in biological and biomedical research. Although their use is being replaced by next-generation sequencing techniques such as RNA-Seq (Nagalakshmi *et al.*, 2008), the large amounts of microarray data relevant to an equally large variety of biological and biomedical problems and available in public databases constitutes a valuable resource that will remain in use for many years. The potentiality of resources such as the Gene Expression Omnibus (GEO) as sources of Real-World Data (RWD) —data derived from a number of sources, outside the context of Randomized Controlled Trials (RCTs), and associated with outcomes in an heterogeneous patient population (Berger *et al.*, 2017)– may in fact boost the use of the wealth of available microarray data in the near future. The importance of RWD as a complementary information source in drug-evaluation studies is based on the observation that data from RCTs does not always match results from observational studies (Trotta, 2012), mostly owing to the limited number of RCT patients, their over-monitoring and the limited follow-up time. Thus, certain adverse drug reactions or lack-of-efficacy problems are hidden until RWD studies are performed, and drug administrations have come to encourage the extraction of information from sources complementary to RCTs to increase evidence around treatments (Sherman *et al.*, 2017; Food and Drug Administration, 2018). A problem of RWD is that it tends to be highly heterogeneous, thus requiring careful analysis and statistical treatment (Berger *et al.*, 2017; Bartlett *et al.*, 2019). Translated to the context of this study, microarray data relevant to a particular problem will often originate from different laboratories and experiments, possibly using different microarray platforms (Bumgarner, 2013) and almost certainly obtained in different batches. In order to make a sensible use of such an heterogeneously sourced data, a data-normalization step is required before data analysis. Normalization can be relatively straightforward when dealing with different batches of a same experiment using the same biological samples, platform and operator, but gets increasingly complex as different operators, platforms and sample sources are introduced. This often leads to studies discarding part of the available data, which could otherwise be used to increase the chances of discovery of meaningful patterns or improve their statistics.

A main source for sample differences arising from systematic biases is the mixing of data from different microarray platforms. Unfortunately, most standard and widely used normalization methods are applicable to or have been developed for the single-microarray-platform context (Rudy and Valafar, 2011), making them generally inappropriate for the cross-study analysis of existing data sets. On the other hand, most existing cross-platform normalization methods, such as ComBat (Johnson *et al.*, 2007), XPN (Shabalin *et al.*, 2008) or DWD (Benito *et al.*, 2004), require the data from different platforms to be normalized together —XPN and DWD were in fact developed for pairwise cross-platform normalization. For large data sets, normalizing platforms together can be restrictive. In addition to involving the normalization of a large joined data set, the eventual addition of new microarray data requires global renormalization. This led to the more recent development of sample-wise, cross-platform normalization methods such as SCAN (Piccolo *et al.*, 2012) and UPC (Piccolo *et al.*, 2013), YuGene (Lê Cao *et al.*, 2014), DBNorm (Meng *et al.*, 2017) —which can operate sample or platform wise– and Shambhala (Borisov *et al.*, 2019). SCAN performs a sample-wise normalization assuming a double Gaussian mixture distribution. It was, however, specifically designed for Affymetrix and two-channel Agilent platforms, thereby restricting its general use. Developed by the same authors, the Universal exPression Code (UPC) builds on SCAN to generate standardized estimates of expression that have a consistent interpretation across platforms, measuring how much the expression of the gene deviates from model-estimated background levels within the sample. Although, the derivation of UPCs is platform specific and, to our knowledge, it is currently available only for Affymetrix and Agilent microarrays through the package

SCAN/UPC, the program offers also a generic UPC function applicable to any microarray platform, by making assumptions on the background and background-plus-signal distributions. The other distribution-based normalization method, DBNorm, scales the data distributions from the individual microarrays to a common form, which does not need to be predetermined (e.g. the distribution from a reference microarray). As a downside, it is very slow. On the other end, YuGene uses a simple transform that assigns a modified cumulative proportion value to each measurement, making the normalization very fast. Finally, Shambhala uses a harmonization method that transforms each profile so that it approaches the output of a chosen golden-standard platform.

Some methods like UPC, Shambhala and MatchMixeR (Zhang *et al.*, 2020) have specifically included in their design the possibility to integrate data from both microarray and RNA-seq sources. As a matter of fact, any methods that can be applied at the gene level could be adapted for such studies. It should be kept in mind, however, that this requires source-specific preprocessing steps taking into account the fundamental differences between these two types of data. In fact, the conceptual differences between microarray and RNA-seq gene-expression measurements are so significant that they may require distinct normalization procedures (Rapaport *et al.*, 2013). In this study, we will solely focus on the application of normalization methods for the integration of gene-expression microarrays.

One should also note that the methods mentioned above were not necessarily developed with a same purpose. For example, ComBat was developed for the adjustment of batch effects and is often used in combination with other methods in cross-platform normalization procedures; although it does not normalize platforms separately, we introduced it in this study because of its broad use. Thus, the SCAN/UPC package offers the possibility to apply ComBat after SCAN normalization and summarization at gene-level —and before transformation to UPCs if so chosen. Thus, methods like ComBat are often called *integration* methods, in so that they integrate previously normalized data. Nevertheless, ComBat has become, on its own, a popular first choice for cross-platform normalization and a frequent benchmark standard for other methods (Walsh *et al.*, 2015; Irigoyen *et al.*, 2018). Shambhala, on the other hand, is classified as a *harmonization* method because it uses a golden standard as reference, while YuGene is a *transformation* and XPN, DBNorm and SCAN are referred to as *normalization* methods. In this study, like in (Rudy and Valafar, 2011), we will refer to all these methods as *cross-*

platform normalization methods so far as they are being used in the literature to make data across different platforms comparable for the purpose of analysis.

Although the number of normalization methods proposed in the literature is large, to our knowledge there are no other major cross-platform normalization methods that can be applied to gene-expression microarrays in a platform agnostic way and that have been tested and validated as such. To enable systematic studies involving the download of microarray data from databases (possibly at different times) and its normalization and storage for later retrieval, allowing a non-linear use of the data —for example, in successive analyses incorporating different amounts of data as available or necessary, it is essential that a downloaded microarray set need not be normalized more than once. Here, we introduce a novel cross-platform normalization method fulfilling all these conditions. The algorithm is called CuBlock, which stands for Cubic approximation by Block. We validate its performance using various metrics and compare it to six methods that can be used in a cross-platform context, namely, the $\log_2$ transform of raw data, ComBat, YuGene, DBNorm, Shambhala and UPC. Overall, CuBlock shows the best performance in this group.

1.3 Methods

In this section we introduce the data sets used for the validation of CuBlock and describe the data preprocessing approach and the methods used for benchmarking and validation.

1.3.1 The data sets

We selected three data sets, including two for benchmark analysis which were previously used in similar studies (Rudy and Valafar, 2011; Borisov *et al.*, 2019). The first data set (here called the single-platform data set) originates from (Maire *et al.*, 2013a,b; Maubant *et al.*, 2015). It contains samples from a unique platform and was used to show CuBlock's consistent results in this context. The second set (here called the reference data set) originates from projects MAQC (MAQC-I) (MAQC Consortium, 2006) and SEQC/MAQC-III (SEQC/MAQC-III Consortium, 2014), which made use of reference RNA samples to assess repeatability of gene-expression microarray data within a specific site, reproducibility across multiple sites and comparability across multiple platforms. The third set (here called the experimental data set) originates from a study trying to assess profile differences of human spermatozoal transcripts from fertile and teratozoospermic males (Platts *et al.*, 2007). The use of these two data sets allows us to assess, independently, effects from technical replicates (same biosample, analysed in different labs with repetition) and biological replicates (different biosamples corresponding to a same condition).

1.3.1.1 The single-platform data set

The data are available in GEO with accession number GSE65212 and contain breast cancer samples including 5 biological groups, TNBC, Her2, LuminalA, LuminalB and Healthy. The platform is the Affymetrix Human Genome U133 Plus 2.0 Array (GPL14877) and the CEL files were read with the CDF file *GPL14877_HGU133Plus2_Hs_ENTREZG.cdf*. In order to focus on patient samples, the cell lines contained in this data set were omitted.

1.3.1.2 The reference data set

The data of this set are accessible in GEO with accession numbers GSE5350 (MAQC-I) and GSE56457 (MAQC-III), respectively. The set contains microarray gene-expression data corresponding to four titration pools from two distinct reference RNA samples: (A) Stratagene's Universal Human Reference RNA pool; (B) Ambion's Human Brain Reference RNA pool; (C) pool with an A:B ratio of 3:1; (D) pool with an A:B ratio of 1:3. These biosamples had been analysed using different platforms and in different sites, as described (MAQC Consortium, 2006; SEQC/MAQC-III Consortium, 2014). Following the work from Rudy and Valafar (2011) and Borisov *et al.* (2019), we selected data from six of the platforms (between parentheses, data-set identifier in this study, GEO platform ID and project of origin):

- Affymetrix Human Genome U133 Plus 2.0 Array (AFX, GPL570, MAQC-I): 3 experiments (sites) (AFX_1 to AFX_3), with 4 biosamples (A-D) per experiment and 5 replicates per biosample (60 samples)

- Agilent-012391 Whole Human Genome Oligo Microarray G4112A (AG1, GPL1708, MAQC-I): 3 experiments (AG1_1 to AG1_3), with 4 biosamples (A-D) per experiment and 5 replicates per biosample (60 samples)

- Illumina Sentrix Human-6 Expression BeadChip (ILM, GPL2507, MAQC-I): 3 experiments (ILM_1 to ILM_3), with 4 biosamples (*A*-*D*) per experiment and 5 replicates per biosample (59 valid samples)

- Illumina HumanHT-12 V4.0 Expression Beadchip (HT12, GPL10558, MAQC-III): 2 experiments (ILM_COH and ILM_UTS), with 4 biosamples (*A*-*D*) per experiment and 3 replicates per biosample (24 samples)

- GeneChip® PrimeView™ Human Gene Expression Array (PRV, GPL16043, MAQC-III): 1 experiment (AFX_USF_PRV), with 4 biosamples (*A*-*D*) and 4 replicates per biosample (16 samples)

- Affymetrix Human Gene 2.0 ST Array (HUG, GPL17930, MAQC-III): 1 experiment (AFX_USF_HUG), with 4 biosamples (*A*-*D*) and 4 replicates per biosample (16 samples)

Note that in the MAQC-I study the following microarrays from AG1 were discarded as outliers after the Agilent's Feature Extraction QC Report: AG1_1_A1, AG1_2_A3, AG1_2_D2, AG1_3_B3. Since the data for these microarrays is nevertheless deposited and we wanted our analysis to be as independent as possible of platform-dependent data-preprocessing steps, we considered also their inclusion. To this end, we evaluated the correlation of the data between all AG1 samples and observed that the "outliers" are highly correlated to the non-outliers of the same experiment and of the other two experiments (about 0.97 in both cases). A dimension reduction of the raw data showed also no outliers. We therefore decided to include these four microarrays in the data set.

1.3.1.3 The experimental data set

This data set contains spermatozoal RNA samples from normally fertile (N) and heterogeneously teratozoospermic (T) subjects and is accessible in GEO with accession number GSE6969. The samples had been analysed on three different platforms (between parentheses, data-set identifier in this study and GEO platform ID):

- Affymetrix Human Genome U133 Plus 2.0 Array (AFF, GPL570): 13 independent biosamples of type N and 8 of type T

- Illumina Sentrix Human-6 Expression BeadChip (ILL1, GPL2507): 5 independent biosamples of type N and 8 of type T. All ILL1 biosamples are replicates of AFF biosamples.

- Illumina Sentrix HumanRef-8 Expression BeadChip (ILL2, GPL2700): 4 independent biosamples of type N and 6 of type T

1.3.2 Data processing

To make the analysis as platform agnostic as possible, we took the image-processed raw intensities for all non-control probes and disregarded any platform-dependent background-signal correction such as that provided by mismatch probes in Affymetrix platforms. CEL files for Affymetrix and txt files for the other platforms were used. Probes with invalid intensities (NaN) in data sets HT12 and HUG were ignored. For Affymetrix microarrays, the intensities of probes constituting a probe set were averaged. We note that in the context of this study preprocessing of Affymetrix probe sets by simple averaging performed just as well as more complex treatments including background correction and RMA median polish (Irizarry *et al.*, 2003), which was not completely unexpected —for example, Hubbell *et al.* (2002) found that simple averaging performed comparably to more robust approaches from the Affymetrix Micro Array Suite under low-noise conditions. From this point onward, the Affymetrix probe-set-average intensities were treated the same way as the raw probe intensities from the other platforms (the number of Affymetrix probe sets per gene being on the same order as the number of probes per gene in other platforms). Note that we could have skipped the probe-set-averaging step and worked directly with all Affymetrix probes. While we tried this, it increased significantly the computational cost of the normalization procedure (due to the several-fold increase in number of probes) at a marginal gain. At this point, probe-set average intensities (Affymetrix) and raw probe intensities (other platforms) were $\log_2$ transformed —we also tried applying quantile normalization in the preprocessing without it improving significantly the results. CuBlock normalization was then applied for each platform separately. Since probes vary among the different microarray platforms, the normalized data sets were then transformed from the probe level to the protein level by mapping probes to UniProtKB accession numbers (ACs) and keeping only those probes that map to an AC present in all platforms. The mapping to proteins was performed by taking the gene identifiers from the GEO tables containing the microarray data. Each selected AC was then assigned an intensity equal to the average of the normalized intensities of associated probes in the given microarray. The choice of UniProt ACs, rather than gene identifiers, was made to facilitate streamlining with protein-level post-normalization analysis in studies where microarray data is used to infer protein expression (i.e. probes matching CDS regions). We note, however, that CuBlock delivers normalized data at the probe level, meaning that the user is free to summarize the data at the gene level if appropriate, and that potential information regarding, for example, probes matching non-coding exon regions, will remain available.

To benchmark CuBlock against established normalization methods applicable in a generic cross-platform context, we compared it to a simple $\log_2$ transform and to the methods ComBat (Johnson *et al.*, 2007), YuGene (Lê Cao *et al.*, 2014), DBNorm (Meng *et al.*, 2017), Shambhala (Borisov *et al.*, 2019) and UPC (Piccolo *et al.*, 2013). YuGene, DBNorm and UPC were applied following the same procedure used for CuBlock, i.e., normalization of the $\log_2$ transform of the probe intensities and successive mapping to ACs. ComBat requires all microarrays to be normalized together, which implies their merging before normalization. Therefore, in this case the mapping to UniProtKB ACs and selection of ACs present in the different platforms was performed after $\log_2$ transform and before ComBat normalization. We note that DBNorm allows normalization per sample and per platform. We performed both, but show only the results obtained with sample-wise normalization since they are better. Comparison to Shambhala was done only for the data sets AFX, AG1 and ILM from the reference data set, since Shambhala-normalized data for these sets has been already reported by the authors as supplementary data to Borisov *et al.* (2019). DBNorm was only used on the experimental data set, as the calculations turned out to be forbiddingly slow. To perform the calculations we used the R package *sva* for ComBat (`https://bioconductor.org/packages/release/bioc/html/sva.html`) and the packages provided by YuGene (`https://cran.r-project.org/web/packages/YuGene/index.html`) and DBNorm (`https://github.com/mengqinxue/dbnorm`) authors in the respective papers. Calculations with these programs were performed with default settings. For DBNorm, in order to reproduce the general case (e.g. this study), in which a reference microarray cannot be straightforwardly selected, we used the option of normalization into a normal distribution. For UPC (`https://www.bioconductor.org/packages/release/bioc/html/SCAN.UPC.html`), we used its generic function for expression set, which takes expresion×samples data, with default parameters from the package.

1.3.3 Comparison and validation methods

To validate and compare the cross-platform normalization methods evaluated in this study we used the methodology described below. The objective was to increase the sensitivity, i.e. the identification of true biological differences, while minimizing platform and various kinds of replica effects. All validation methods were applied on a subset of 500 proteins that best distinguish two given biological groups.

To select the 500 proteins we first performed a differential analysis on the normalized data, for all platforms in the reference or experimental data set. To this end, we performed Welch's t-test to evaluate, for each protein, the difference between the associated mean intensities in units of uncertainty (the t-statistic) in two biological groups, A and B from the reference data set (total of 16624 proteins) or N and T from the experimental data set (total of 16937 proteins). Note that we deliberately avoid considerations on whether the data sets meet the requirements of the t-test, since we used the test simply to identify the 500 proteins with largest separation of group means per uncertainty unit, that is, with lowest associated p-values, irrespective of the error in the p-value and, therefore, of its valid interpretation as a probability. Although for such purpose we do not require the calculation of FDR-adjusted p-values (q-values) (Storey, 2002), since they conserve p-value ranking, we did obtain them and show corresponding ROC-like curves (the cumulative distribution function of the q-values) in Figure 1.1. We decided to select a fixed number of proteins, rather than proteins with a p- or q-value below a given arbitrary threshold, to enable the comparison of methods using data sets of equal and reasonably large dimensionality. We note, nevertheless, that the 500-protein cut corresponds to an FDR well below 10^{-2} (Figure 1.1). The differential analysis was performed with the MATLAB function *mattest*.

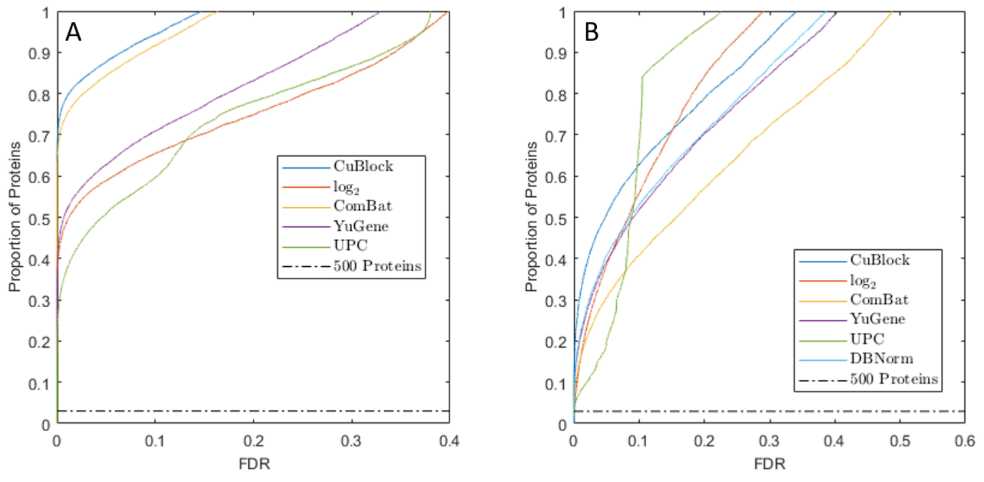


Figure 1.1: ROC-like curves. The cumulative distribution function of the FDR-adjusted p-values (also called q-values) plotted for the reference and the experimental data set after normalization with CuBlock, $\log_2$, ComBat, YuGene, DBNorm and UPC. The dotted horizontal line represents the 500-protein cut. Plot A corresponds to the reference data set. Plot B corresponds to the experimental data set.

1.3.3.1 Silhouette plot

Silhouette plots are graphical displays of data partitions (Rousseeuw, 1987), where clusters are represented by so-called *silhouettes* generated by comparison of cluster tightness and separation. The method assigns a silhouette value between -1 and 1 to each element of a cluster, indicating if the element is well clustered (value close to 1), lies between two or more clusters (close to 0) or is likely misclassified (close to -1). The silhouette plot is then generated by representing the values for all elements as bars, for the different cluster partitions. We computed three silhouette plots for the reference data set: one identifying clusters with platforms, one where the data was assigned to groups $A \cup C$ and $B \cup D$ and one where the partitioning was represented by sets A, B, C and D. For the experimental data set, silhouette plots based on platform partitioning and T *vs.* N partitioning were computed. The MATLAB function *silhouette* was used to compute the silhouette plots.

1.3.3.2 *t*-SNE dimension reduction

t-SNE (van der Maaten and Hinton, 2008) is a stochastic dimension-reduction method aimed to preserve the local structure of data (keeping the low-dimensional representation of very similar data points close together) while retaining essential traits of global structure. It analyses the neighbourhood of the data points by calculating pairwise conditional probabilities representing their similarity. The method then tries to find a low-dimensional representation that minimizes the difference between the high-dimensional and low-dimensional conditional probabilities. The parameter controlling the number of neighbours is called perplexity, and is typically given values between 5 and 50. Due to its stochastic nature and the dependence on the chosen perplexity parameter, the algorithm may converge to irrelevant solutions. We thus performed 10 runs for each of a number of perplexity values and selected the one producing the most consistent biological partitioning according to the average silhouette values. For the reference data set we used perplexity values from 5 to 50, in increments of 5, and selected the representation giving the best clustering relative to sets A, B, C and D. For the experimental data set, we used perplexity values 5, 10 and 15 and selected the representation giving the best clustering relative to sets T and N. The MATLAB function *tsne* was used to perform the t-SNE dimension reduction.

1.3.3.3 Dendrogram

We performed a hierarchical clustering analysis using the Euclidean distance as metric and the arithmetic mean as linkage criterion, and represented the resulting cluster hierarchy as a dendrogram. To assess the significance of the clusters, we applied multiscale bootstrap resampling as provided in the R package *pvclust* (Suzuki and Shimodaira, 2006). By default, this package considers 10 relative bootstrap sample sizes (bootstrap sample size divided by total sample size), from 0.5 to 1.4, with 1000 resamplings per sample size, leading to a total of 10000 bootstrap resamples. The package provides two statistics to estimate the significance of the obtained clusters: the bootstrap probability (BP) or frequency (expressed as percentage) of observation of a given cluster in the bootstrap resamples, and the approximately unbiased p-value (AU), an unbiased version of BP. More details on multiscale bootstrap resampling can be found in Shimodaira (2004). We plot the dendrograms using the R package *dendextend*.

1.3.3.4 SVM classification

The goal of this analysis was to assert whether relevant patterns can be found using the data from only one platform. Support vector machines (SVM) (Cortes and Vapnik, 1995) are binary classifiers applicable to problems that are reducible to a binary outcome, such as the T and N phenotypes in our experimental data set. We trained a linear support vector machine model for each platform using the following approach. To reduce feature-vector dimensionality, where dimensions are proteins (more specifically their microarray-derived intensities), while retaining the capacity to asses how well the 500 proteins separate the T and N populations, the training was performed six times, starting with dimension 5 and increasing it up to dimension 10. For each of 1000 runs with a given dimensionality, we selected randomly from the 500 protein set as many proteins as dimensions, extracted the corresponding data from sets T and N, trained a linear SVM model for each platform, separately, and tested it on the other two platforms. This led to a total of 6000 models per platform. For each platform, we calculated the mean and standard deviation of different classification scores over the 6000 models, namely, Accuracy, Matthews Correlation Coefficient (MCC) (Boughorbel *et al.*, 2017), Balanced Accuracy and Area Under the ROC Curve (AUC) (Fawcett, 2006). The MATLAB function *svm* was used to train the SVM models.

1.4 Algorithm

The CuBlock algorithm relies on the simple and widely used assumption that most genes are neither over- nor under-expressed (Yang *et al.*, 2002). Thus, a transformation following the cubic polynomial x^3 will leave most of the genes around 0 and slowly differentiate the extremes, i.e. the under- and over-expressed genes. The complementary idea in CuBlock is the use of a block-wise transformation, which had been already implemented successfully in XPN (Shabalin *et al.*, 2008)). To this end, CuBlock partitions probes into clusters and, for each sample and probe cluster (i.e. for each data block), transforms the data by a procedure that involves its mapping to objective values between -1 and 1 (with density increasing toward 0) and the fitting of a cubic polynomial to the resulting distribution (see below). By using data blocks, different profiles present (mixed) in the full data set are considered, and as many different cubic polynomials are fitted to them, underlying different shapes contained within the original distribution. A pseudo code of the CuBlock algorithm is described in Figure 1.2. It calls two additional algorithms with pseudo codes provided in Figures A.1 and A.2.

```
**CuBlock**($X, k, N$)
$X$ is the log2 transform of micro array, columns are samples and rows are probes.
$k$ is the number of clusters.
$N$ is the number of repetitions.
The algorithm calls 2 other algorithms: *GetTargetValues* returns the target values
for a polynomial fitting. *ModPol* modifies the fitted polynomial to make it increasing.
*mean*, resp. *std*, returns the mean value, resp. standard deviation, of an array.
$[a: b]$ is the set of natural numbers between $a$ and $b$.
$X[\cdot,\cdot]$ access the element(s) in bracket.

    $\hat{X} \leftarrow NaN$ of same size as $X$
    count $\leftarrow$ 0 of same size as $X$
    $m \leftarrow$ number of samples
    REPEAT
      $C \leftarrow k$-means clustering partition of the probes of $X$ into $k$ clusters
      FOR1 $i \in [1: m]$
        FOR2 $j \in [1: k]$
          IF there are more than 100 probes in cluster $j$
            $B \leftarrow X[j^{\text{th}}$ Cluster of $C$, $i^{\text{th}}$ Sample$]$
            $B \leftarrow \frac{B - \text{mean}(B)}{\text{std}(B)}$
            $BS \leftarrow$ sort $B$ in ascending order
            $D \leftarrow$ GetTargetValues($BS$)
            $P \leftarrow$ get the coefficients for a cubical polynomial
                   such that its evaluation at $BS$ best fits the target values $D$
            $\hat{B} \leftarrow$ ModPol($B, P$)
            $\hat{X}[j^{\text{th}}$ Cluster of C, $i^{\text{th}}$ Sample$] \leftarrow \hat{X}[j^{\text{th}}$ Cluster of C, $i^{\text{th}}$ Sample$] + \hat{B}$
            count$[j^{\text{th}}$ Cluster of C, $i^{\text{th}}$ Sample$] \leftarrow$ count$[j^{\text{th}}$ Cluster of C, $i^{\text{th}}$ Sample$] + 1$
          END IF
        END FOR2
      END FOR1
    UNTIL $N$ repetitions
    $\hat{X} \leftarrow$ elementwise division of $\hat{X}$ by count
    END FUNCTION
```

Figure 1.2: Pseudocode describing the CuBlock algorithm (see description in Section 1.4).

The input to CuBlock is a matrix X containing the $\log_2$ transform of the gene-expression microarray intensities, where columns are samples and rows are probes. As discussed in section 1.3.2 it is up to the user to decide any level of preprocessing of the input log_2-transformed intensities, for example, probe-set summarization for Affymetrix microarrays.

CuBlock makes use of the k-means clustering algorithm (Lloyd, 1982) to partition probes in the space defined by the samples —a probe data point is a vector of probe intensities of dimension equal to the number of samples– and is applied per platform, i.e. the k-means clustering is performed for all samples of a given platform. k-means is an iterative algorithm that tries to partition the data into a predefined number k of non-overlapping clusters, starting from a random initialization of their centroids. Because of the random initialization, clusters from different runs may differ, and the core part of the CuBlock algorithm is repeated several times for different solutions of k-means. Thus, input parameters k and N in Figure 1.2 refer to the chosen number of k-means clusters and repetitions, which in this work took values of 5 and 30, respectively. The number of clusters was chosen to be low enough that the k-means algorithm will not, for some partitions, converge always to the same solution and high enough that blocks with different distributions will be obtained. The CuBlock algorithm finds first a probe-cluster partition in the space defined by the samples and then applies its normalization scheme to data blocks defined as those ($\log_2$) probe-intensity values from a sample that belong to a given cluster. Therefore, for k clusters and m samples we have a total of $k \cdot m$ blocks. The advantage of the normalization by block is that it decomposes the distribution of probe intensities of a sample into its different block distributions, according to similarities between probes found by the clustering algorithm in the space of all samples. These different distributions will enable the emergence of different patterns present in the data. Instead, if the blocks were selected at random or the whole sample was used, the normalization method would estimate parameters based on a unique distribution, masking these different patterns. Although we initially determine the probe clusters using all samples, we then normalize sample by sample to reduce the dependence of the normalization on the full sample collection. The strategy of normalization by block is similar to that used by the cross-platform normalization method XPN (Shabalin *et al.*, 2008). However, XPN defines probe clusters and sample clusters with two independent application of k-means (one on the input matrix and the other on its transpose) and blocks are then constituted by all possible combinations of one sample cluster with one probe cluster.

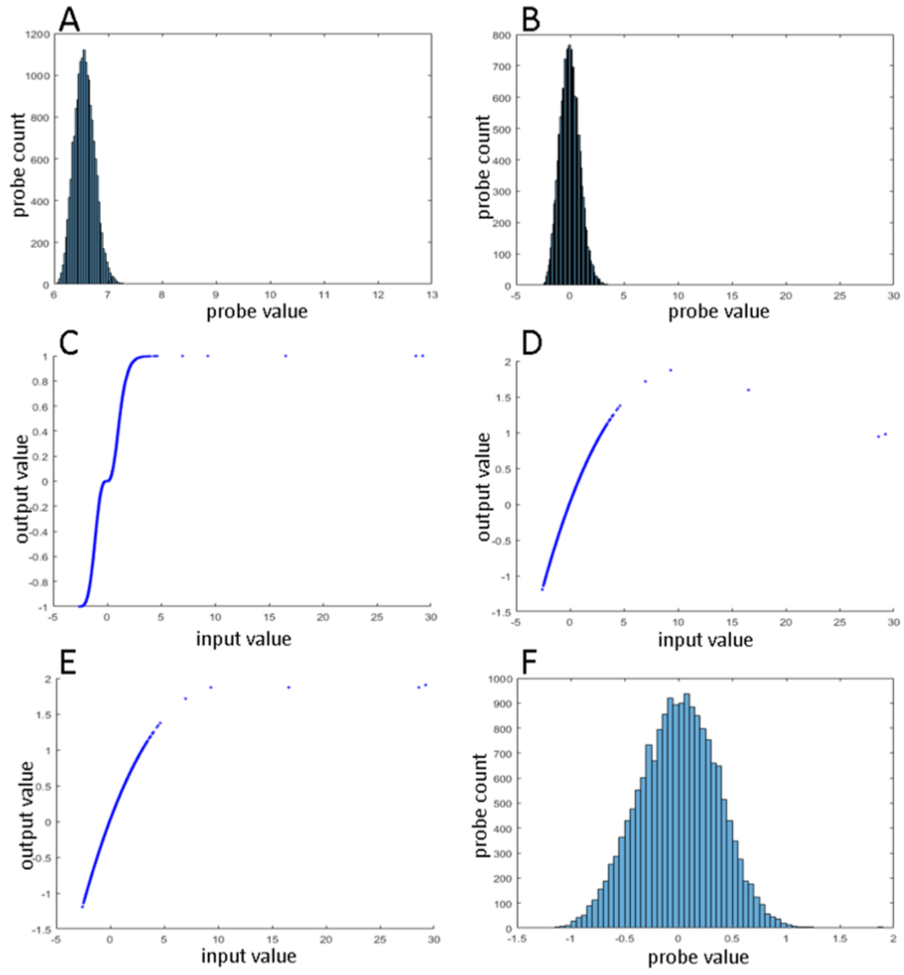


Figure 1.3: The different steps of the CuBlock algorithm for an example block. A: histogram of the untransformed block. B: histogram of the z-transformed block. C: target values (output values) of the sorted z-transformed block (input values) as obtained after the GetTargetValues algorithm. D: values after evaluation of the fitted polynomial (output value) on the sorted z-transformed block (input value); the cubic polynomial's coefficients are such that these values minimize the root mean square error with the target values plotted in C. E: modified values (output value) as obtained after the ModPol algorithm; the decreasing values in D are corrected in order to preserve data sorting upon normalization. F: histogram of the normalized block.
The example block contains a few positive outliers that are identified by the fitted cubic polynomial in the decreasing part; this part is corrected by equating the decreasing values to the last increasing point and letting the final increasing part continue its growth from this point.

For each block, and each of the N repetitions of the k-means clustering, CuBlock fits a cubic polynomial to a mapped set of points symmetrically distributed between -1 and 1 with density increasing toward zero. This is performed in four steps, as shown in Figure 1.2. First, the block data is linearly transformed to z-scores (zero mean and unit standard deviation). These are then used as input values of a mapping function whose output values will be used to fit the cubic polynomial, as described in the pseudocode shown in Figure A.1. The mapping associates the sorted values present in the block to an equal number of equidistant points between -1 and 1, and takes these new points to an uneven power in order to have their distance decrease as they approach zero from either side (Figure A.1.3). The exact uneven power will determine how slow is the growth of the points around zero, and is selected such that, on average, the values of the block that are within standard deviation, i.e. the block values between -1 and 1, are mapped to a value smaller than 0.1 (Figure A.1.4). The algorithm tries uneven powers between 3 and 21 and the first one that fulfills the criterion is selected. Next, the algorithm finds the coefficients of a cubic polynomial that, when evaluated on the sorted block data (input values), best fits the output values from the mapping function (Figure A.1.3). We chose to fit a cubic polynomial instead of a higher degree one to avoid overfitting. Polynomial coefficients were obtained with the MATLAB function *polyfit*, with degree 3.
If the block data is not symmetric or contains many outliers, a cubic polynomial will produce a poor fit. Thus, the polynomial will increase along the symmetric part of the block and decrease as it reaches the outliers (Figure A.1.5). Despite leading to a poor fit, this feature can be used to identify asymmetry issues and outliers. When this is the case and decreasing values are identified after evaluating the polynomial on the block data, the decreasing values are corrected in order to preserve data sorting upon normalization. Roughly, the correction equates the decreasing values to the last increasing value (after an increasing section) or to the last decreasing value (before an increasing section). The precise corrections are described in Figure A.1.5, and the cases where the cubic polynomial might decrease are considered in Figure A.2.

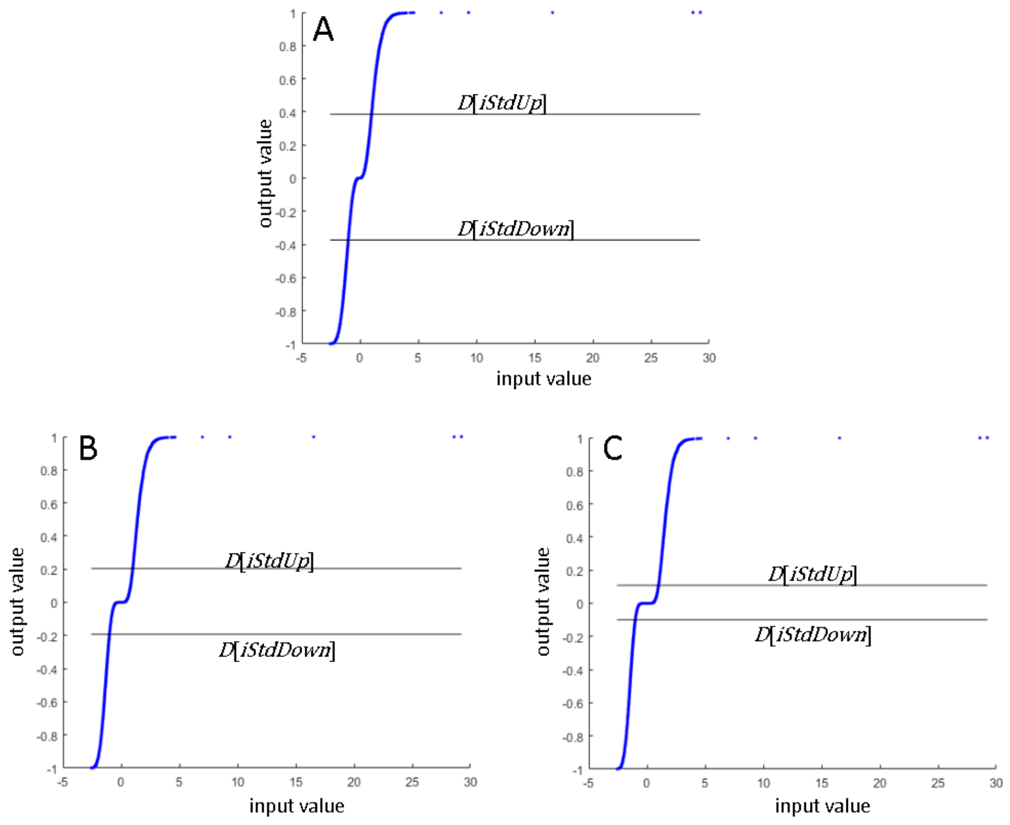


Figure 1.4: Illustration of possible target values (output value) with respect to the sorted z-transformed block (input value) as described in the GetTargetValues algorithm. D is a vector containing the target values and equals L^p where p is an uneven power and L is a vector of equidistant points between -1 and 1 (the length of L is the number of points in the block). $[iStdDown : iStdUp]$ represents the set of indices corresponding to the points in the sorted z-transformed block that are between -1 and 1, i.e. the points within standard deviation; mean$(D[iStdDown : iStdUp])$ is the average of the output values contained between the two horizontal lines in the plots. The target values are $D = L^p$ where p is the smallest uneven power such that mean$(D[iStdDown : iStdUp])$ is smaller than 0.1.
A: target values for $p = 3$; mean$(D[iStdDown : iStdUp]) = 0.095$. B: target values for $p = 5$; mean$(D[iStdDown : iStdUp]) = 0.033$. D: target values for $p = 7$; mean$(D[iStdDown : iStdUp]) = 0.013$.
In this example block, the chosen p is 3.

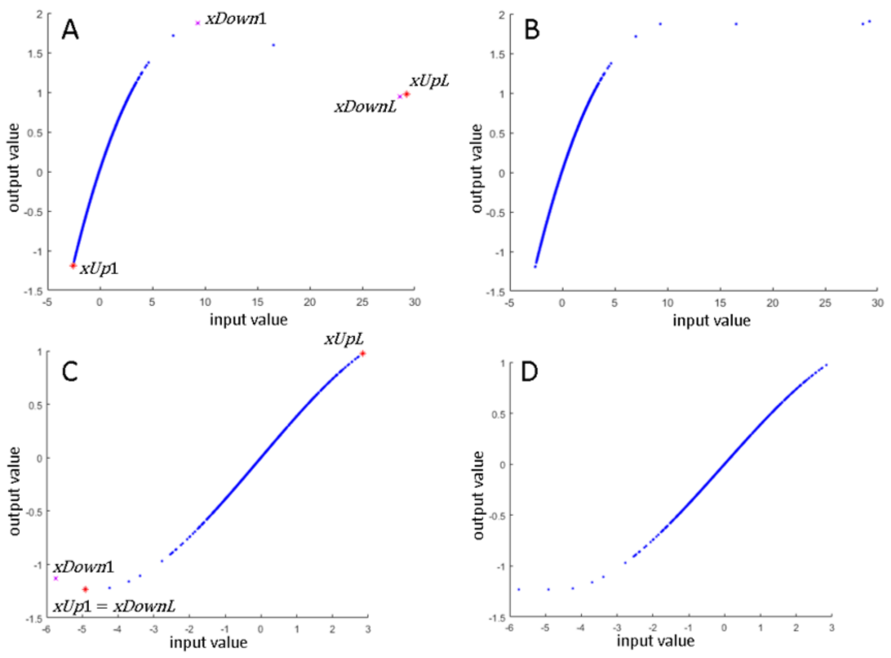


Figure 1.5: Illustration of example modifications to correct for the decreasing values in the evaluation of the fitted polynomial (output value) on the z-transformed sorted block (input value) as described in the ModPol algorithm. A-B: example on one block with the output values before (A) and after (B) the correction. C-D: example on another block with the output values before (C) and after (D) the correction.
$xDown1$ and $xDownL$ are the first and last decreasing values, respectively, and $xUp1$ and $xUpL$ are the first and last increasing values, respectively. These values are used to identify the decreasing part(s) which will be modified to be equal to the closest point in the main increasing part.

The output of CuBlock is a matrix of normalized gene-expression values, where columns are samples and rows are probes. As discussed in section 1.3.2 it is up to the user to decide at which level and using which database codes for the mapping to that level, the probe values should be summarized.

Figure 1.6 shows the histograms of different samples before and after normalization. While before normalization the samples follow clearly different distributions, after normalization the distributions are much more homogeneous. We note that before normalization the distributions are clearly platform dependent (compare A and C, which correspond to the same biosample but different platforms, and B and C, which correspond to different biosamples and the same platform). This effect is remarkably corrected after normalization.

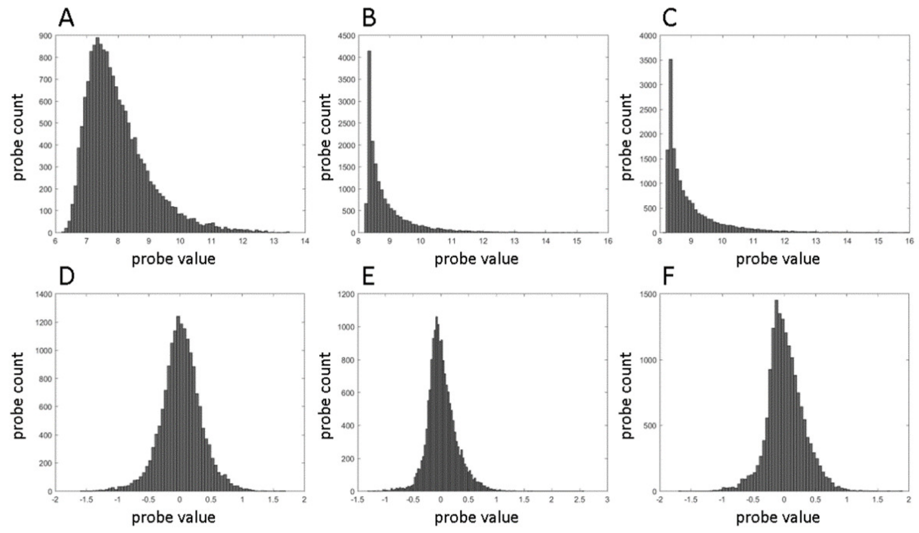


Figure 1.6: Example histograms for samples from different platforms. A-C: histograms before CuBlock normalization and after $\log_2$ transform. D-F: histograms after CuBlock normalization. A, D: biosample *A*, platform AFX; B, E: biosample *B*, platform AG1; C, F: biosample *A*, platform AG1.

1.5 Results and discussion

The algorithm described in the previous section was applied to the data introduced in Section 1.3.1 after preprocessing (see Section 1.3.2), and the results were compared to those obtained with other normalization methods as explained in Section 1.3.3. In line with the objectives stated in the introduction, the discussion of the results evolves around the ability of the methods to highlight biological patterns in a multi-platform context.

1.5.1 Single-platform data set

Although CuBlock was not developed for single-platform normalization —this being already well covered by other methods, it can be also used for this purpose. Thus, the single-platform data set was used to demonstrate that results from CuBlock are very consistent with those from RMA normalization.
The RMA normalization consists of a background correction step, a quantile normalization and a sumarization of the probes constituting a probe set

with log_2 transformation. It was performed using the MATLAB functions *rmabackadj*, *quantilenorm* and *rmasummary* with default settings. The preprocessing step for CuBlock was the same as presented in the section 1.3.2. Since only one platform is used, the analysis was performed using all probe sets and no summarization at the gene or protein level was performed, as opposed to the analysis with the other two sets.

Figure 1.7 shows the dendrogram analysis (performed as indicated in Section 1.3.3.3 but with 100 resamplings instead of 1000 due to the larger size of the data set), where the color bars under the dendrograms indicate the biological group, and the t-SNE dimension reduction plot (see Section 1.3.3.2) with the biological groups identified also by color. It can be observed that the results from CuBlock in the identification of the 5 biological groups (TNBC, Her2, LuminalA, LuminalB and Healthy) is consistent with those from the RMA normalization.

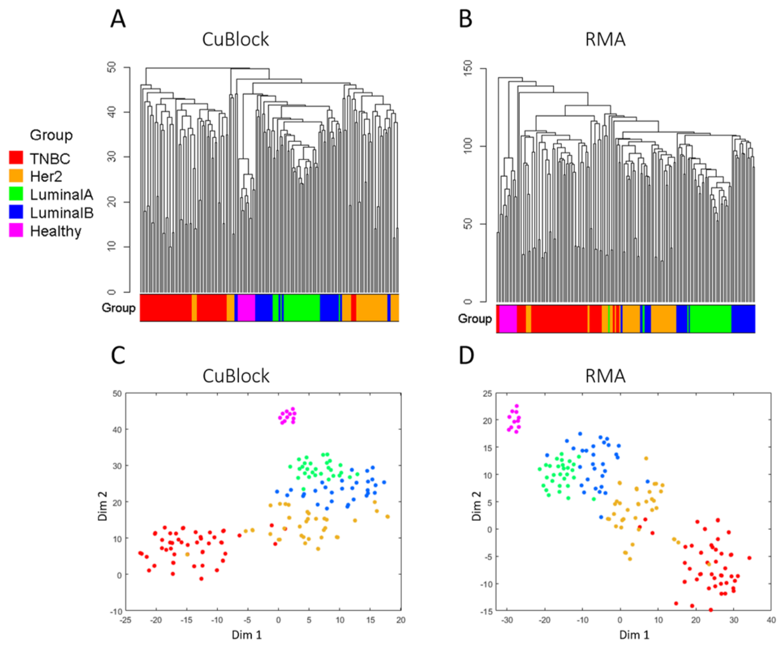


Figure 1.7: Single-platform normalization by CuBlock and RMA. A: dendrogram for CuBlock normalized data. B: dendrogram for RMA normalized data. C: t-SNE for CuBlock normalized data; perplexity (Prp) and mean silhouette index (SI) values (see Section 1.3.3.2): Prp=15, SI=0.58. D: t-SNE for RMA normalized data; Prp=15, SI=0.63.

1.5.2 Reference data set

Three analyses were performed with the reference data set. The first one contains a benchmark analysis between CuBlock, $\log_2$, ComBat, YuGene and UPC with the data from the six platforms. The second one contains a benchmark analysis between CuBlock and Shambhala with the data from three of the six platforms. Finally, the third one consists of an analysis of the reference data set with the removal of two biological groups.

1.5.2.1 Six platforms

Figure 1.8 and Figures A.3-A.6 show the results obtained with the different normalization methods using the dendrogram, silhouette and t-SNE analyses, respectively. The three validation methods show that CuBlock and ComBat separate very clearly the biological groups A, B, C and D (except for a couple of A points in ComBat's case). ComBat tends to produce tighter but less cleanly separated clusters for these four groups, as illustrated by both the t-SNE (Figure A.4C) and silhouette (Figure A.3C) plots. CuBlock is the only method that clusters the biological groups A and C, and B and D together in the dendrogram plot (Figure 1.8A), and this is also underlined in the corresponding silhouette plot in Figure A.5A, showing high and homogeneous silhouette values. On the contrary, $\log_2$, ComBat, YuGene and to a smaller extent UPC tend to cluster C with D (Figure 1.8B-E). In fact, $\log_2$ and YuGene have difficulties to separate these two groups at all, while UPC has serious difficulties to separate the groups C and D from the parent groups (A and B; see also Figure A.4). Figures A.5 and A.6 show silhouette plots using the groups $A \cup C$ and $B \cup D$ and the platforms as given clusters, respectively. We note that even though CuBlock is shown to emphasize the biological differences and Figure A.6A indicates weak platform clusters, both the t-SNE (Figure A.4A) and dendrogram (Figure 1.10A) plots show that, within each of the A, B, C, D clusters, the samples are subclustered by platform. As can be seen in these Figures, ComBat mixes the data from the different platforms best, while YuGene, UPC and $\log_2$ are, approximately in this order, worst at mixing platform data.

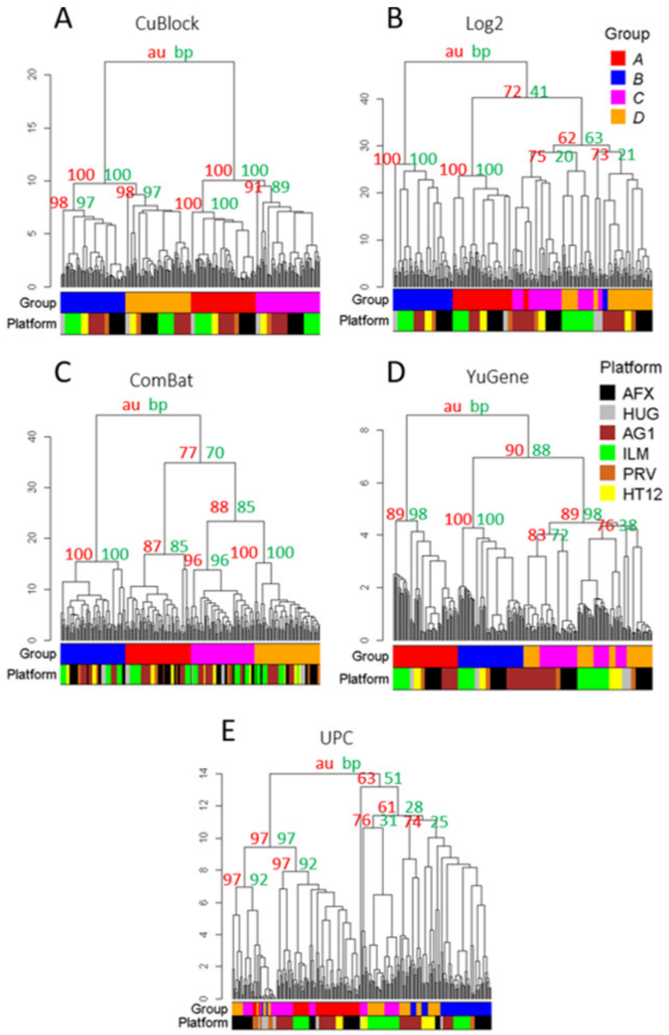


Figure 1.8: Dendrogram analysis of the reference data set (six platforms) after normalization with CuBlock (A), $\log_2$ (B), ComBat (C), YuGene (D) and UPC (E). Color bars under the dendrograms indicate the biological group and platform corresponding to each leaf; the BP (green) and AU (red) values (see Section 1.3.3.3) for some selected clusters are indicated at the origin of the branches.

In Figure 1.8 and throughout this study, the log_2 transform plays the role of control method. As a second potential control, we also used a common approach consisting in platform-specific normalization of the samples followed by a centering transformation. Figure 1.9 shows the dendrogram plot of the RMA method (background correction, quantile normalization, summarization at the probe-set level for Affymetrix platforms and log_2 transformation) followed by Z-score transformation (subtracting the sample's mean and dividing by its standard deviation). It can be observed that the results are only slightly better than those from a log_2 transformation, for which reason we kept the latter as the simplest approach.

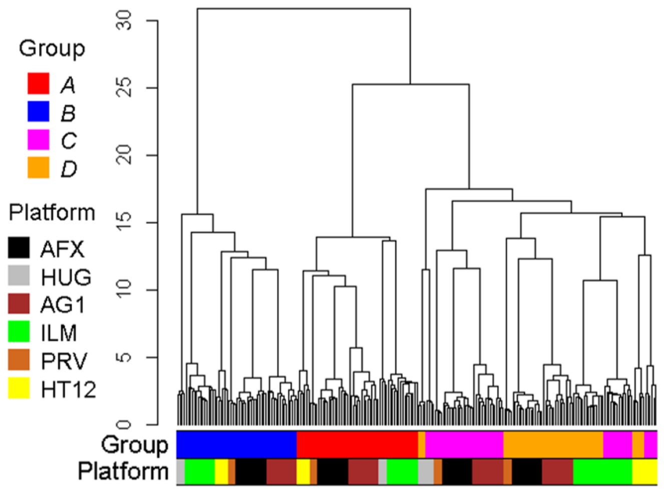


Figure 1.9: Dendrogram analysis of the reference data set after RMA normalization and centering by Z-score.

1.5.2.2 Three platforms

To compare the results from CuBlock and Shambhala (the latter reported by Borisov *et al.* (2019) for the same data set), we also performed the analysis for the three-platform subset used by the authors of Shambhala, namely AFX, ILM and AG1. They had concluded that Shambhala separates well $A \cup C$ from $B \cup D$ but not A from C or B from D. Using our selection of 500 proteins that best distinguish A from B, when looking at the results for Shambhala in Figure 1.10B,D we observe that, while $A \cup C$ forms a relatively clear cluster, all the $B \cup D$ points from AG1 samples are clustered with $A \cup C$, making $B \cup D$ a well defined cluster only for AFX and ILM. As illustrated by the t-SNE and dendrogram plots and by the negative silhouette values in Figure 1.10F, Shambhala does also not distinguish A, B, C and D from each other well. The results for CuBlock in Figure 1.10A,C,E show the same features already discussed in Section 1.5.2.1 using the data for six platforms.

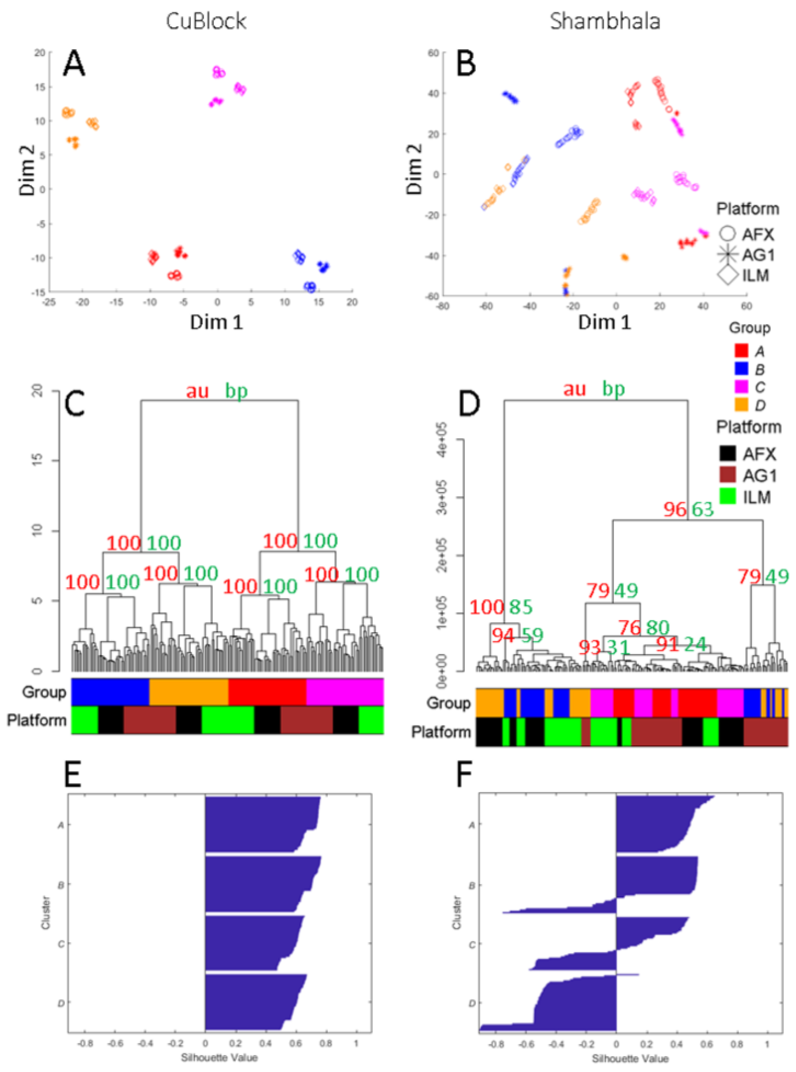


Figure 1.10: *t*-SNE dimension reduction, dendrogram and silhouette plots for the reference data set (three platforms) after normalization with CuBlock and Shambhala. A: *t*-SNE for CuBlock normalized data; point color and shape indicate biological group and platform, respectively (right-hand legend); perplexity (Prp) and mean silhouette index (SI) values (see Section 1.3.3.1): Prp = 25, SI = 0.97. B: corresponding analysis for Shambhala-normalized data; Prp = 5, SI = 0.28. C, D: dendrograms for CuBlock (C) and Shambhala (D) normalized data; color bars below the dendrograms indicate the biological group and platform corresponding to each leaf; the BP (green) and AU (red) values (see Section 1.3.3.3) for some selected clusters are indicated at the origin of the branches. E: silhouette plot for CuBlock-normalized data, using the groups A, B, C and D as given clusters; SI values: 0.70 (A), 0.69 (B), 0.58 (C) and 0.60 (D). F: silhouette plot for Shambhala-normalized data; SI values: 0.49 (A), 0.27 (B), 0.03 (C) and -0.52 (D).

1.5.2.3 Missing biological groups

CuBlock is a method that works with the actual distribution of the data, without making any assumption on its shape. It is, in that sense, dependent on the actual differences found in the input data at the platform level —which are in turn highlighted by the block treatment enabling the uncovering of the different distributions present in the data. To test this dependence, we analysed again the reference data set using CuBlock and ComBat while removing the groups B and D from the platforms HUG and AG1. In other words, these two platforms were normalized only with A and C samples. The biological difference between A and C is that 25% of C is made of B RNA samples. The other four platforms were normalized with all four biological groups. As it can be seen in Figure 1.11A,C, CuBlock results in the clustering of the C samples of HUG and AG1 separately and closer to the D cluster of the other platforms than to their C cluster. However, the A cluster remains a well-defined cluster for all platforms. This suggests that, in the two platforms with missing groups, CuBlock emphasizes the difference between the available data, as predicted above. For HUG and AG1, this means emphasizing the differences between A and C (the only groups it sees), thus bringing C closer to the D cluster formed by the other platforms, since, as C itself, D is also a mixture of A and B. ComBat does however do similarly in this regard (Figure 1.11B,D), with the C samples of HUG and AG1 being even more mixed with the D samples, and the A samples of the two platforms getting closer (see t-SNE plot) to the C samples of the four other platforms, some of them ending up clustered (see dendrogram) in this group.

1.5.3 Experimental data set

1.5.3.1 Three platforms

Figure 1.12 and Figures A.7-A.9 show the results obtained for the human sperm data set, after normalization with CuBlock, $\log_2$, ComBat, YuGene, DBNorm and UPC. CuBlock is the only normalization method that significantly distinguishes the two biological groups, T and N. The dendrogram plots in Figure 1.12 and t-SNE plots in Figure A.8 show that ComBat has troubles to establish a clear boundary between the two groups, particularly for the ILL2 platform, while the other methods tend to misclassify the ILL2 samples corresponding to the N group. Similarly to the results for the refer-

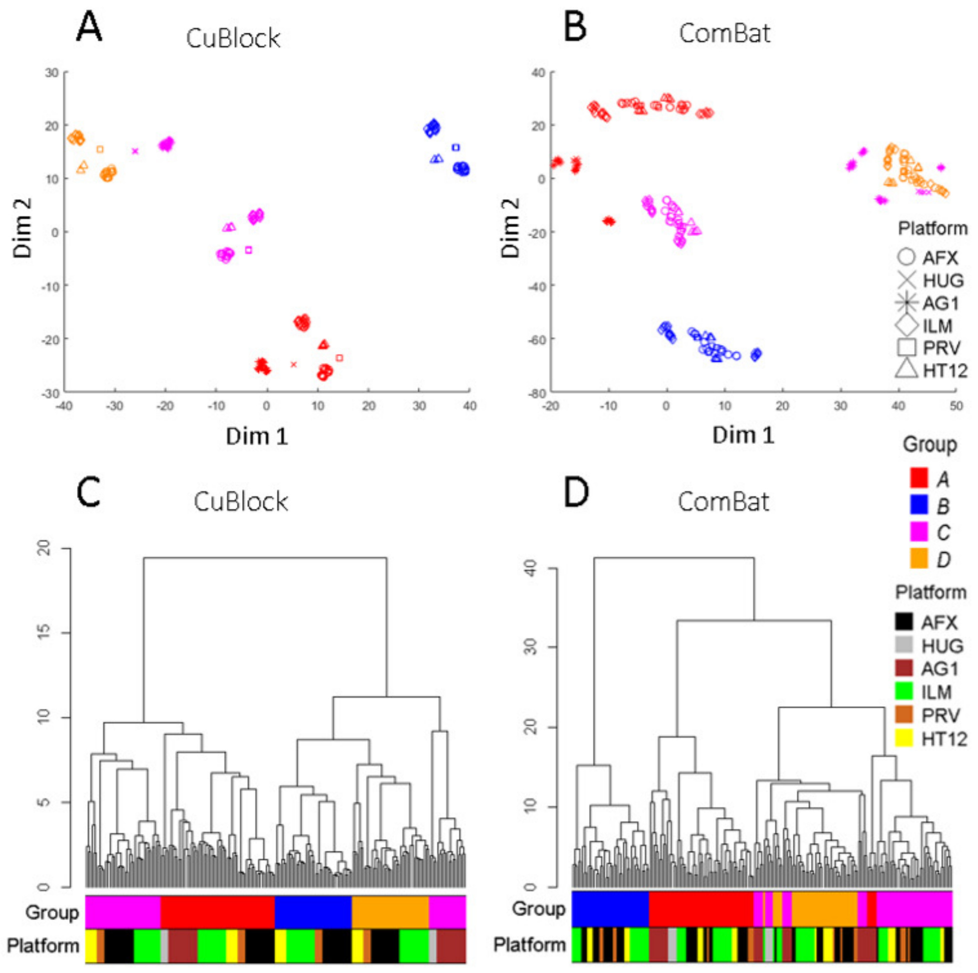


Figure 1.11: *t*-SNE dimension reduction and dendrogram plots for the reference data set, with exclusion of the *B* and *D* samples from platforms HUG and AG1, after normalization with CuBlock and ComBat. Point color and shape indicate biological group and platform, respectively (right-hand legend). A: *t*-SNE for CuBlock normalized data; perplexity (Prp) and mean silhouette index (SI) values (see Section 1.3.3.2): Prp = 15, SI = 0.80. B: corresponding analysis for ComBat-normalized data; Prp = 10, SI = 0.67. C, D: Dendrograms for CuBlock (C) and ComBat (D) normalized data; color bars under the dendrograms indicate the biological group and platform corresponding to each leaf.

ence data set (Section 1.5.2.1), CuBlock tends to sort the samples by platform within the clusters T and N (except for one T sample from ILL2). To investigate whether patterns that are found using one platform can be extrapolated to the other platforms, we performed a SVM classification test as described in Section 1.3.3.4. The results are shown in Table 1.1. In all cases, CuBlock outperforms the other methods. It is also worth noting that no matter which platform is used for the training based on CuBlock, the results are always very similar. To a lesser extent, this is also true for DBNorm. However, using $\log_2$, ComBat, YuGene and UPC, training with ILL2 gives worse results than with the other platforms, probably due to the fact that this platform constitutes a better defined cluster, as shown in the silhouette plots in Figure A.9.

1.5.3.2 Correlation analysis

In Figure 1.12, it can be seen that ComBat clusters more strongly AFF and ILL1 (which contains replicates of AFF) than CuBlock does. As a recall, the experimental data set contains three platforms AFF, ILL1 and ILL2 and two biological groups T and N. The samples from the platform ILL1 are replicates of a subset of samples of AFF. To discuss the results on replicates, we computed the average pairwise Kendall correlation between different groups of samples from this data set, as reported in Table 1.2. The actual groups of samples in this table are:

- 1st column: the replicate samples shared by AFF and ILL1.
- 2nd column: the samples from one group (T or N) of AFF with the opposite group of ILL1, i.e. AFF/T vs ILL1/N and AFF/N vs ILL1/T.
- 3rd column: same as the 2nd column for AFF and ILL2.
- 4th column: same as the 2nd column for ILL1 and ILL2.

If one would only look at the first column, ComBat would clearly be the superior method in correlating replicates, and CuBlock the worst. However, looking at the second column we can see that the average correlation between different samples from different biological groups (same two platforms) is also

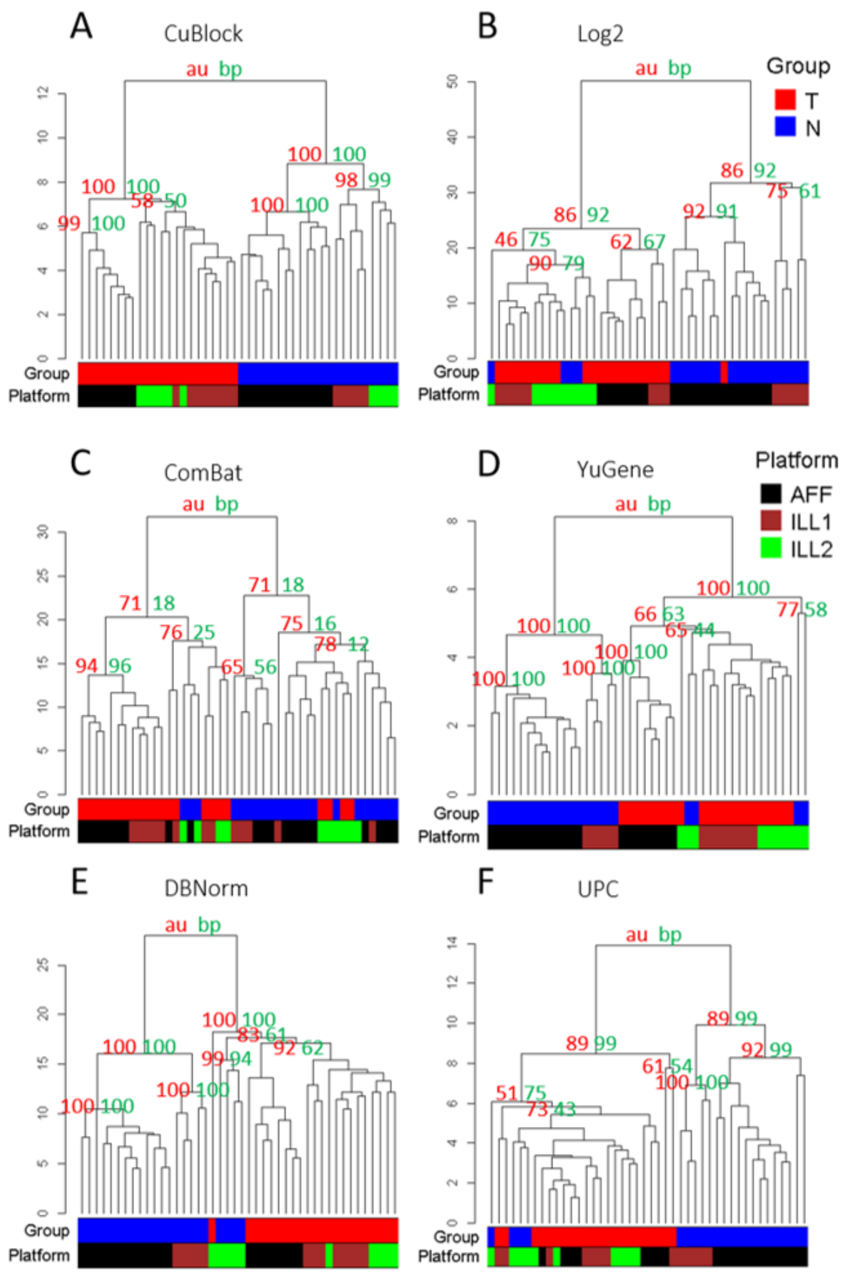


Figure 1.12: Dendrogram analysis of the experimental data set after normalization with CuBlock (A), $\log_2$ (B), ComBat (C), YuGene (D), DBNorm (E) and UPC (F). Color bars under the dendrograms indicate the biological group and platform corresponding to each leaf; the BP (green) and AU (red) values (see Section 1.3.3.3) for some selected clusters are indicated at the origin of the branches.

SVM classification scores (see Section 1.3.3.4)

Training platform	Accuracy	MCC	Balanced Accuracy	AUC
		CuBlock		
AFF	0.88 ± 0.08	0.76 ± 0.17	0.86 ± 0.10	0.98 ± 0.03
ILL1	0.92 ± 0.06	0.85 ± 0.12	0.91 ± 0.07	0.99 ± 0.02
ILL2	0.86 ± 0.16	0.74 ± 0.32	0.86 ± 0.17	0.99 ± 0.04
		$\log_2$		
AFF	0.77 ± 0.07	0.52 ± 0.16	0.71 ± 0.08	0.70 ± 0.11
ILL1	0.80 ± 0.06	0.64 ± 0.11	0.81 ± 0.06	0.81 ± 0.04
ILL2	0.44 ± 0.16	-0.13 ± 0.35	0.46 ± 0.16	0.24 ± 0.31
		ComBat		
AFF	0.73 ± 0.11	0.53 ± 0.20	0.76 ± 0.10	0.90 ± 0.08
ILL1	0.76 ± 0.08	0.57 ± 0.15	0.77 ± 0.08	0.90 ± 0.06
ILL2	0.61 ± 0.30	0.22 ± 0.61	0.61 ± 0.30	0.63 ± 0.36
		YuGene		
AFF	0.81 ± 0.07	0.63 ± 0.15	0.77 ± 0.08	0.91 ± 0.06
ILL1	0.87 ± 0.08	0.74 ± 0.16	0.87 ± 0.09	0.96 ± 0.04
ILL2	0.55 ± 0.07	0.07 ± 0.17	0.53 ± 0.07	0.97 ± 0.11
		DBNorm		
AFF	0.81 ± 0.07	0.62 ± 0.16	0.77 ± 0.09	0.91 ± 0.07
ILL1	0.86 ± 0.06	0.76 ± 0.09	0.87 ± 0.05	0.95 ± 0.04
ILL2	0.79 ± 0.13	0.61 ± 0.24	0.77 ± 0.13	0.96 ± 0.11
		UPC		
AFF	0.74 ± 0.08	0.45 ± 0.22	0.68 ± 0.10	0.82 ± 0.08
ILL1	0.82 ± 0.06	0.68 ± 0.09	0.83 ± 0.05	0.90 ± 0.05
ILL2	0.57 ± 0.10	0.10 ± 0.24	0.54 ± 0.11	0.75 ± 0.34

Table 1.1: Mean and standard deviation over the 6000 models per platform and method.

very high with ComBat. That is, the preservation of the ranking for the ca. 17000 proteins when the samples are replicates (coefficient of 0.56) and when the samples belong to different individuals with different phenotypes (coefficient of 0.46) is very similar, suggesting that the relatively high correlation observed in column 1 is mostly due to factors other than the samples being replicates. This is further corroborated by columns 3 and 4, as one would expect the correlation coefficients in columns 2 to 4 to decrease relative to that in column 1.
The platform effect is highlighted by the fact that in the Log2 row the correlation between unrelated samples evaluated with platforms of the same manufacturer (Illumina, 4th column) is almost twice as big as the one between replicates (1st column) that have been analysed with platforms from two different manufacturers.
We also note that, compared to Log2, the only method that increases the correlation (in all cases) is ComBat, which is also the only method that normalizes the platforms together. CuBlock, on the other hand, reduces the correlation between unrelated samples to 0 and, although the replicates show low correlation in column 1, it is the method with the highest increase in correlation in column 1 (where the correlation between replicates of AFF and ILL1 is assessed), relative to column 2 (where the correlation between unrelated samples from the same two sets, AFF and ILL1, is assessed). The greater tendency shown by CuBlock two decorrelate the samples (one could argue that a certain level of correlation is expected between samples from the same tissue type, due to the constitutive expression of many proteins) is inherent to the cubic polynomial fitting procedure.
With this table, the results shown in Figure 1.11 can be better understood: CuBlock clusters ILL1 with ILL2 more than with AFF because, as it can be seen in all the other rows in the table, they are highly correlated, even if they are composed of different samples. While CuBlock reduced this platform-induced correlation much better than the other methods, column 4 still shows a correlation higher than those in columns 2 and 3 and close to that in column 1. Moreover, while it is true that, for ComBat and DBNorm, AFF and ILL1 seem to be more related, it is also true that when using DBNorm ILL2 is also related to ILL1 and when using ComBat the inclusion of ILL2 leads to rather poor results in terms of separation of the biological groups.
We can thus distinguish methods that tend to correlate the different platforms to enable the comparison of samples from methods that tend to decorrelate the platforms to highlight existing patterns in the samples. The pur-

Kendall correlation table

	replicates AFF/ILL1	distinct groups AFF/ILL1	distinct groups AFF/ILL2	distinct groups ILL1/ILL2
CuBlock	0.11 ± 0.04	-0.03 ± 0.04	0.00 ± 0.02	0.09 ± 0.05
Log2	0.21 ± 0.07	0.16 ± 0.05	0.12 ± 0.04	0.41 ± 0.11
ComBat	0.56 ± 0.08	0.46 ± 0.10	0.57 ± 0.08	0.53 ± 0.08
DBNorm	0.18 ± 0.06	0.14 ± 0.04	0.11 ± 0.03	0.38 ± 0.09
YuGene	0.17 ± 0.06	0.13 ± 0.04	0.11 ± 0.03	0.37 ± 0.09
UPC	0.21 ± 0.07	0.16 ± 0.04	0.12 ± 0.03	0.42 ± 0.11

Table 1.2: The table displays average Kendall rank correlation coefficients, with standard deviation, calculated on the 16'936 proteins common to the different platforms. For two sets of samples A and B, the value given corresponds to the average of the Kendall coefficients between each sample of A and each sample of B.

pose of CuBlock is to compare samples from different platforms in order to find patterns (or conclude the absence of patterns), it is not designed to perfectly overlap cross-platform replicates. Moreover, as already shown, high correlation does not imply meaningful correlation, i.e. methods that excel at overlapping cross-platform replicates tend also to overlap non-replicates.

1.6 Conclusion

We have introduced an algorithm for cross-platform normalization of gene-expression microarray data as well as a strategy to validate cross-platform normalization methods, with a focus on the capacity of the algorithm to properly separate samples from different biological groups after normalization and across multiple platforms. Overall, CuBlock showed good results on the two data sets used in this evaluation, a data set specifically generated for testing and standardization purposes and a data set from an actual experimental study. CuBlock could always differentiate, clearly, the underlying biological groups after mixing data from up to 6 different platforms. Nevertheless, we observed that within each biological group the algorithm tends to subcluster samples by platform, indicating a remaining, yet comparatively small, platform effect. The ComBat algorithm (Johnson *et al.*, 2007) showed also good performance on the reference data set, with better mixing of data from different platforms than all the other methods tested. However, on the experimental data set, where samples are from different individuals and the difference between biological groups might become less obvious than in the reference data set, ComBat did not perform as well. Platform mixing was still good but the distinction between the two biological groups was not clear. To rationalize the differences in platform-mixing properties between CuBlock and ComBat, we performed a rank correlation analysis and show that methods normalizing all platforms together, like ComBat, tend to correlate the data from the different platforms, as opposed to methods normalizing platforms separately, which tend to decorrelate them. While high correlation among samples from different platforms might be desirable in some set-ups, there is no guarantee that this correlation will be meaningful, as shown for ComBat, which in our test induced correlation among platforms almost independently of the actual level of expected correlation between them. As already mentioned, ComBat also requires the platforms to be normalized together, making it a less convenient method for systematic application to multiple data sets. On the other hand, DBNorm —used only with the experimental data set, as it proved computationally much more time demanding than the rest, YuGene and UPC performed only slightly better than $\log_2$. For the set evaluated, Shambhala lagged clearly behind the other methods, arguably including a simple $\log_2$ transformation. Finally, we also showed, by training SVMs with single-platform data from the experimental set, that when normalizing the data with CuBlock the patterns that are found using one platform can be extrapolated to the other platforms significantly better than when the normalization is done with any other of the methods.

Because no assumptions are made on the distributions underlying the input data, CuBlock can be thought of as a transformation which result remains close to the input. CuBlock fits cubic polynomials to data blocks that are found by k-means clustering, thereby trying to best fit the different distributions found in the data corresponding to a sample (different blocks need not have the same distribution) and it does so without assuming a shape for these distributions. As a consequence, CuBlock emphasizes the differences within the input data at the platform level, making it sample-composition dependent despite the only step in the algorithm where the microarray samples from a given platform are considered together is when applying the k-means algorithm (afterwards, each sample is considered separately). This sample dependence was highlighted in this study when analysing the reference data set after removing biological groups in some platforms. Nevertheless, the sample dependence is even more prominent for algorithms normalizing platforms together, such as ComBat.

We note that, while in this study the results were presented at the protein level, CuBlock is applied and returns the normalized data at the probe level, as an appropriate level for gene-expression microarrays. In fact, like most other methods, it could technically be applied at any expression $\times$ samples level. Studying the effect of the application of CuBlock and other cross-platform normalization methods at different levels with appropriate (and possibly study-specific) preprocessing steps could be an interesting development which would ideally lead to an appropriate systematic integration of RNA-seq with gene expression microarrays. This however goes beyond the scope of this study.

In summary, we have shown that CuBlock can be applied to data from multiple microarrays in a platform agnostic way and preserves the biological grouping of the samples, demonstrating a good performance for different types of samples. It is therefore a tool appropriate for gene-expression studies based on multiple microarray sets collected along multiple platforms and at different times, thus facilitating the extraction of knowledge from the wealth of microarray data available in public repositories and enabling the use of these repositories as sources of Real-World Data.

Chapter 2

CNN-PepPred: An open-source tool to create convolutional NN models for the discovery of patterns in peptide sets. Application to peptide-MHC class II binding prediction

2.1 Abstract

Summary: The ability to unveil binding patterns in peptide sets has important applications in several biomedical areas, including the development of vaccines. We present an open-source tool, CNN-PepPred, that uses convolutional neural networks to discover such patterns, along with its application to peptide-HLA class II binding prediction. The tool can be used locally on different operating systems, with CPUs or GPUs, to train, evaluate, apply and visualize models.
Availability and Implementation: CNN-PepPred is freely available as a Python tool with a detailed User's Guide at: `https://github.com/ComputBiol-IBB/CNN-PepPred`

2.2 Introduction

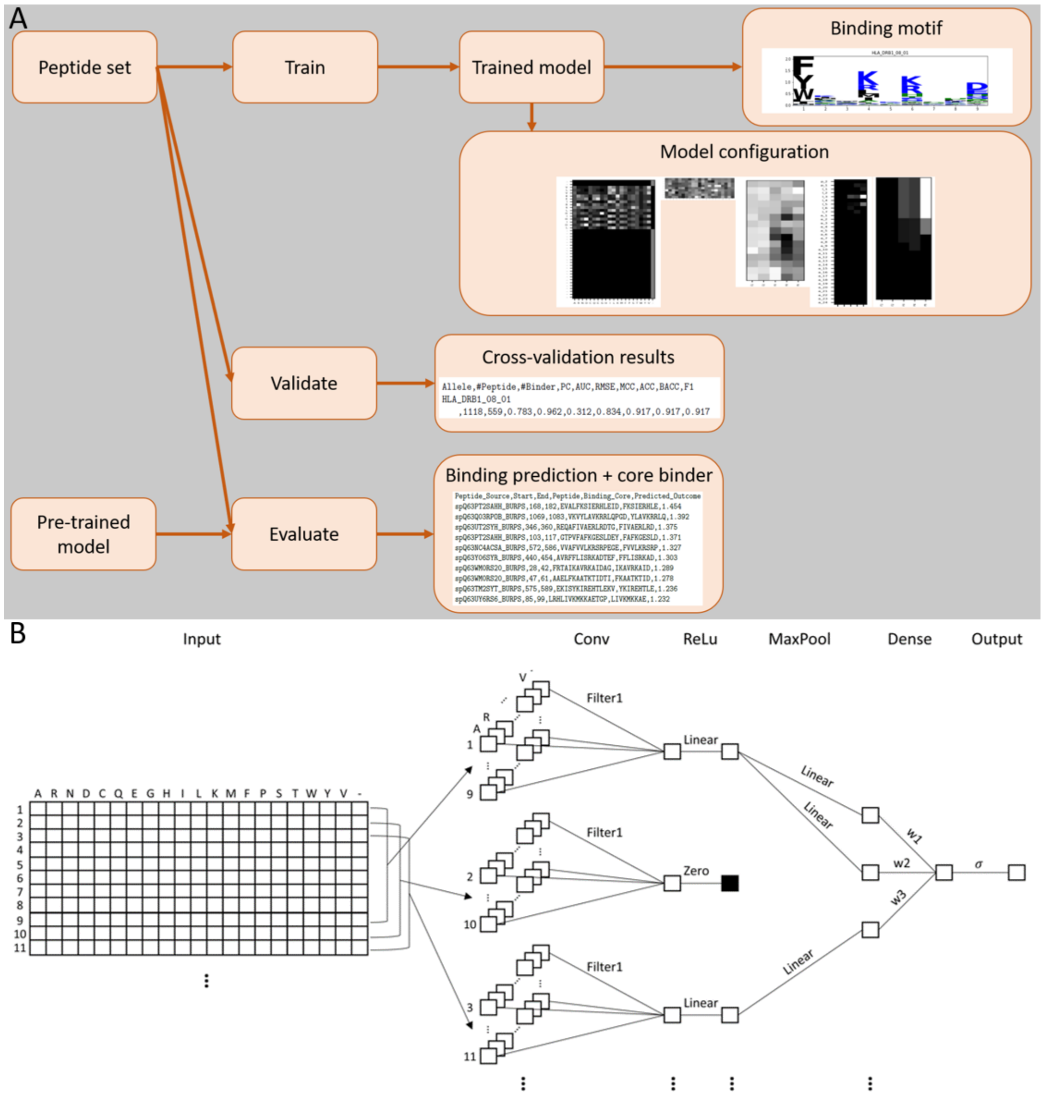
The prediction of antigenic determinants —epitopes– of specific proteins or full proteomes is a key aspect of immunoinformatics, with a wide-range of applications from the study of autoimmunity to the development of cancer immunotherapies and therapeutic as well as prophylactic vaccines. For MHC epitopes in particular, experiments coupling natural-ligand elution to mass spectrometry identification (Caron *et al.*, 2015) have led in recent years to an explosion in the amount of data available on MHC peptide recognition, enabling data-driven approaches to *in-silico* epitope prediction (Wang *et al.*, 2010).

The state-of-the-art in the prediction of peptide-MHC class II binding is arguably represented by the NetMHCII family of algorithms (Andreatta *et al.*, 2015). Their core is a neural-network-based algorithm called NNAlign (Nielsen and Lund, 2009). NetMHCII features an allele-specific algorithm, while NetMHCIIpan deploys a universal network for the prediction of binding to any MHC class II allele with know protein sequence (Jensen *et al.*, 2018). The functionality of both tools is constrained to the application of pre-trained models. NNAlign (Nielsen and Andreatta, 2017), on the other hand, offers the possibility of training new models. These tools, however, are only available as on-line platforms or executables and there exists, to our knowledge, no reference providing information on the implementation details of the neural network behind them, in a manner that would make it reproducible. Recently, these algorithms have taken a new research direction in order to include multiple-allele data sets (MA) (Alvarez *et al.*, 2019; Reynisson *et al.*, 2020). Other groups have also developed their own methods to include MA data (Chen *et al.*, 2019; Racle *et al.*, 2019), but are also only available for application purposes in the form of a platform or an executable. While all these tools are being successfully used in multiple applications and are amenable to non-expert use, we believe that the lack of clear implementation details and open code for the core predictive algorithm hinder their use by bioinformatics researchers who need a tool that can be fully integrated in larger codes and possibly modified to fit particular needs in different areas of peptide recognition.

It is indeed to satisfy our own needs in this field that we developed CNN-PepPred, a new tool for the discovery of patterns in peptide sets. The tool was developed to conveniently fit general purpose needs such as the training of models, their application to new data, their evaluation by cross-validation and gaining visual insight on the training process. The software is accompanied by a detailed User's Guide and has been tested on Linux and Windows, using CPUs and GPUs. In the following section, we provide a concise description of CNN-PepPred.

2.3 Description

CNN-PepPred is based on Convolutional Neural Networks (CNN) and written in Python. Figure 2.1B illustrates the architecture of the CNN, implemented with the open-source libraries Keras (`github.com/fchollet/keras`) and TensorFlow (`tensorflow.org`). The peptides are first encoded using the blosum62 similarity matrix (Henikoff and Henikoff, 1992). In the schema, this corresponds to filling the input table with the blosum62 similarity between the residue (row) and the amino acid type (column). In order to deal with the different lengths in the peptide sets, the symbol "-" representing the absence of further residues was introduced and added at the end of all peptides in the training set so that their lengths match. The convolutional layer applies different filters (only one is illustrated in the schema) to all overlapping l-mers in the peptide, where l is a user defined parameter, typically $l = 9$ in applications related to MHC class II. The ReLu activation layer sets all negative outputs to zero. The MaxPool layer will select the highest outputs and the dense layer will connect the results to generate the output, i.e. the predicted binding score. The final model is an equally-weighted ensemble of CNNs, resulting from different numbers of filters in the convolutional layer times the number of initial configurations.
Both NNAlign and CNN-PepPred use a blosum encoding. The main difference between the two methods is that while NNAlign uses a two-step procedure that identifies a core l-mer within a peptide and then applies a network weight configuration to the core (and flanking regions) to generate the output, CNN-PepPred uses convolution to slide through all possible overlapping l-mers and subsequent layers to activate them and generate the output. In other words, CNN-PepPred does not select a core binder but makes full use of all overlapping l-mers within the peptide, and what we may considered the core binder is the overlapping l-mer that is most activated during the feed-forward pass.



```
Allele,#Peptide,#Binder,PC,AUC,RMSE,MCC,ACC,BACC,F1
HLA_DRB1_08_01
    ,1118,559,0.783,0.962,0.312,0.834,0.917,0.917,0.917
```

```
Peptide_Source,Start,End,Peptide,Binding_Core,Predicted_Outcome
spQ63PT2SAHH_BURPS,168,182,EVALFKSIERHLEID,FKSIERHLE,1.454
spQ63Q03RPOB_BURPS,1069,1083,VKVYLAVKRRLQPGD,YLAVKRRLQ,1.392
spQ63UT2SYH_BURPS,346,360,REQAFIVAERLRDTG,FIVAERLRD,1.375
spQ63PT2SAHH_BURPS,103,117,GTPVFAFKGESLDEY,FAFKGESLD,1.371
spQ63NC4ACSA_BURPS,572,586,VVAFVVLKRSRPEGE,FVVLKRSRP,1.327
spQ63Y06SYR_BURPS,440,454,AVRFFLISRKADTEF,FFLISRKAD,1.303
spQ63WM0RS20_BURPS,28,42,FRTAIKAVRKAIDAG,IKAVRKAID,1.289
spQ63WM0RS20_BURPS,47,61,AAELFKAATKTIDTI,FKAATKTID,1.278
spQ63TM2SYT_BURPS,575,589,EKISYKIREHTLEKV,YKIREHTLE,1.236
spQ63UY6RS6_BURPS,85,99,LRHLIVKMKKAETGP,LIVKMKKAE,1.232
```

Figure 2.1: A. Main workflow of CNN-PepPred. Given a peptide set, the tool can be used to train a model to fit known binding outcomes and visualize the binding motif and the trained model configuration. The user can cross-validate the model and use a previously trained model to evaluate new instances. B. Schematic representation of the convolutional neural network architecture of CNN-PepPred.

The tool contains many functionalities that can be conveniently called by filling a simple template. Users can train their own models, evaluate them

through cross-validation, save them and apply them to new data (Figure 2.1A). It is also possible to visualize the binding motif of a trained model with *Logomaker* (Tareen and Kinney, 2019). For more insight on the feed-forward pass on a selected peptide set, the user can also print all layers and corresponding outputs of the trained model. In the latest version to date (Version 0.1.1), the tool allows training with transfer learning (see Section 2.4.5). In addition, advanced users may incorporate the CNNPepPred class, providing all the functionality, to other Python developments.
When applied to HLA class II binding, using 51 alleles with data curated by the IEDB (`www.iedb.org`, see Section 2.5.1 below for details), our tool showed significant improvement over NNAlign-2.1 (Nielsen and Andreatta, 2017) in predictive performance. To ensure balanced binding/non-binding sets, CNN-PepPred provides a function that generates random, naturally existing peptides following the length distribution of the binding peptides. The models trained with these sets are available and can be applied to new data using the template. In order to test and benchmark the method, we performed a 5 fold cross-validation with CNN-PepPred and NNAlign-2.1, where the folds are minimizing the overlap between folds of nonamers contained within their respective peptides and were the same for both methods. The results are summarized in Figure 2.2, which illustrates the boxplot of the F1 and AUC scores for all alleles and per group of alleles (DR, DP and DQ). The weighted (by the number of binders) average F1(AUC) score for CNN-PepPred is 0.845(0.919) while it is 0.824(0.896) for NNAlign. A Wilcoxon signed-rank test was performed to compare the scores, giving a p-value of 0.00082 for the F1 score and smaller than 10^{-5} for the AUC. The data with cross-validation folds are available in the repository `https://github.com/ComputBiol-IBB/CNN-PepPred` and a more detailed description of the analysis is given in Section 2.5.1.2.
We used the T-cell epitope benchmark data set from Jensen *et al.* (2018) as an independent evaluation set for the models trained with the data retrieved from IEDB. CNN-PepPred was found to perform better on average than NetMHCIIpan-3.2 (Jensen *et al.*, 2018), although the results were overall similar as shown in the below Section 2.5.3. Section 2.5.2 contains also a cross-validation comparison with the data from NetMHCII-2.3 and NetMHCIIpan-3.2, giving overall similar results. Note that the latter two tools cannot be used themselves to train new models.
In the below Sections 2.5.1.2 and 2.5.2.2 we compare computation times for CNN-PepPred, NNAlign and NetMHCIIpan, showing that, for the specific

CPU and GPU used, CNN-PepPred compares well to NNAlign for training and clearly outperforms NetMHCIIpan for prediction, NNAlign being the faster prediction tool.
In the latest version of CNN-PepPred (Version 0.1.1), the tool offers the possibility to train using transfer learning (see Section 2.4.5). The application of the cross-validation set up to models trained with transfer learning showed an increase in predictive performances (see Section 2.5.1.4). With application on the evaluation set, an increase in performances was also observed, however to a lesser extent (see Section 2.5.3). This could be an indication that training with transfer learning might overfit some redundancy present in the full training set and could not generalize as well to different data if the parameters are such that the training rate or number of training cycles are too high.
The github repository contains two main Python scripts, one with the CNNPepPred class and one that uses this class following a template filled by the user. The class was written so that it could be also used independently from the template. The Appendix B contains a full description of the class, its parameters, the template, the installation and some examples on how to run it.
The core training algorithm of CNN-PepPred consists only of a few Keras lines of code. It can therefore be also used independently from the tool, as a starting point for further analysis and/or generalizations that could fit other types of peptide sets. Thanks to the ease of use of libraries like Keras and TensorFlow, the implementation does not require the developer to fully implement specific feed-forward/back-propagation passes that would fit one type of data. The CNN-PepPred code is provided under an open-source license.

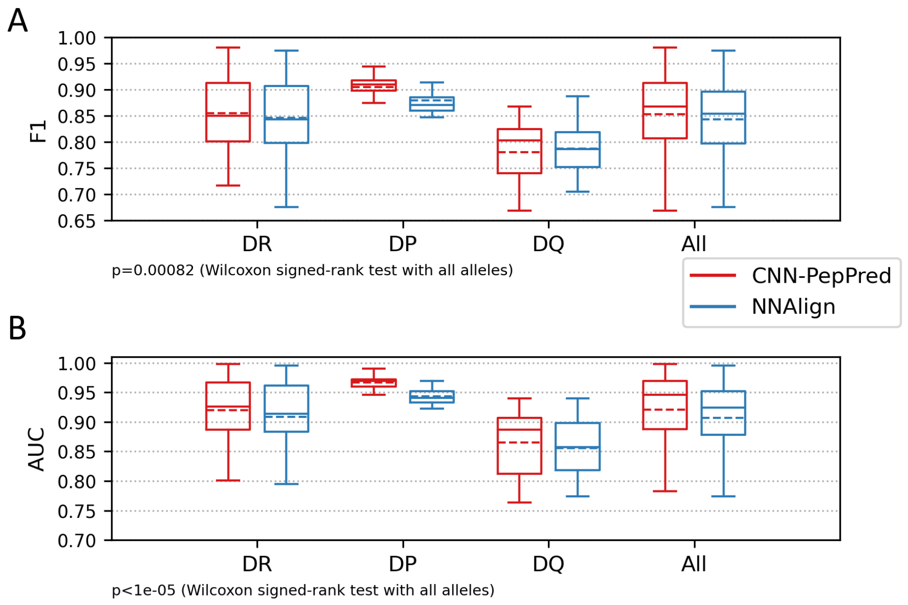


Figure 2.2: A. Boxplot of cross-validation F1 score of CNN-PepPred (red) and NNAlign-2.1 (blue) on all alleles and different subsets of them (DR, DP and DQ). The p-value corresponds to a Wilcoxon signed-rank test between the scores of the two methods on all alleles. The solid/dashed lines represent the median/mean values. B. Equivalent to B. with the AUC score.

2.4 Method

2.4.1 Peptide's encoding

The peptides are encoded using the *blosum62* similarity matrix Henikoff and Henikoff (1992).

```
,A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V,-
A,4,-1,-2,-2,0,-1,-1,0,-2,-1,-1,-1,-1,-2,-1,1,0,-3,-2,0,-4
R,-1,5,0,-2,-3,1,0,-2,0,-3,-2,2,-1,-3,-2,-1,-1,-3,-2,-3,-4
N,-2,0,6,1,-3,0,0,0,1,-3,-3,0,-2,-3,-2,1,0,-4,-2,-3,-4
D,-2,-2,1,6,-3,0,2,-1,-1,-3,-4,-1,-3,-3,-1,0,-1,-4,-3,-3,-4
C,0,-3,-3,-3,9,-3,-4,-3,-3,-1,-1,-3,-1,-2,-3,-1,-1,-2,-2,-1,-4
Q,-1,1,0,0,-3,5,2,-2,0,-3,-2,1,0,-3,-1,0,-1,-2,-1,-2,-4
E,-1,0,0,2,-4,2,5,-2,0,-3,-3,1,-2,-3,-1,0,-1,-3,-2,-2,-4
G,0,-2,0,-1,-3,-2,-2,6,-2,-4,-4,-2,-3,-3,-2,0,-2,-2,-3,-3,-4
```

```
H,-2,0,1,-1,-3,0,0,-2,8,-3,-3,-1,-2,-1,-2,-1,-2,-2,2,-3,-4
I,-1,-3,-3,-3,-1,-3,-3,-4,-3,4,2,-3,1,0,-3,-2,-1,-3,-1,3,-4
L,-1,-2,-3,-4,-1,-2,-3,-4,-3,2,4,-2,2,0,-3,-2,-1,-2,-1,1,-4
K,-1,2,0,-1,-3,1,1,-2,-1,-3,-2,5,-1,-3,-1,0,-1,-3,-2,-2,-4
M,-1,-1,-2,-3,-1,0,-2,-3,-2,1,2,-1,5,0,-2,-1,-1,-1,-1,1,-4
F,-2,-3,-3,-3,-2,-3,-3,-3,-1,0,0,-3,0,6,-4,-2,-2,1,3,-1,-4
P,-1,-2,-2,-1,-3,-1,-1,-2,-2,-3,-3,-1,-2,-4,7,-1,-1,-4,-3,-2,-4
S,1,-1,1,0,-1,0,0,0,-1,-2,-2,0,-1,-2,-1,4,1,-3,-2,-2,-4
T,0,-1,0,-1,-1,-1,-1,-2,-2,-1,-1,-1,-1,-2,-1,1,5,-2,-2,0,-4
W,-3,-3,-4,-4,-2,-2,-3,-2,-2,-3,-2,-3,-1,1,-4,-3,-2,11,2,-3,-4
Y,-2,-2,-2,-3,-2,-1,-2,-3,2,-1,-1,-2,-1,3,-3,-2,-2,2,7,-1,-4
V,0,-3,-3,-3,-1,-2,-2,-3,-3,3,1,-2,1,-1,-2,-2,0,-3,-1,4,-4
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
```

The symbol ”-” stands for the absence of amino acids. A similar type of encoding is used in the models of the netMHCII family Jensen *et al.* (2018). With the template, you can set your own similarity matrix keeping the above format and amino acid order.
A peptide will then be encoded as an ”image” for the input of the convolutional neural network. This image can be though of as a table where the rows are the residues of the peptide and the columns are the 20 amino acids + the absence of amino acid. This table is then filled using the corresponding similarity value. To account for the difference in peptides' lengths, the absence-of-amino-acid character ”-” will be added at the end of each peptide until its length matches the maximal length in the training data set. Moreover, a fixed number of character ”-” will be added at the beginning and end of the peptide; this step can be thought of as a sequence equivalent to an image zero-padding. The exact number of additional characters ”-” at the beginning and end of the sequence is determined by the parameters *nbPrev* and *nbAfter*, respectively; they are both set to 2 by default. Therefore, the number of rows, i.e. the length of the input peptide, will be the length of the maximal peptide in the training data set + *nbPrev* + *nbAfter*.
For example, the peptide *MSAIESVLHERRQFA*, in a model where the maximal length is 20, will be encoded as:

```
,A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V,-
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
M,-1,-1,-2,-3,-1,0,-2,-3,-2,1,2,-1,5,0,-2,-1,-1,-1,-1,1,-4
```

```
S,1,-1,1,0,-1,0,0,0,-1,-2,-2,0,-1,-2,-1,4,1,-3,-2,-2,-4
A,4,-1,-2,-2,0,-1,-1,0,-2,-1,-1,-1,-1,-2,-1,1,0,-3,-2,0,-4
I,-1,-3,-3,-3,-1,-3,-3,-4,-3,4,2,-3,1,0,-3,-2,-1,-3,-1,3,-4
E,-1,0,0,2,-4,2,5,-2,0,-3,-3,1,-2,-3,-1,0,-1,-3,-2,-2,-4
S,1,-1,1,0,-1,0,0,0,-1,-2,-2,0,-1,-2,-1,4,1,-3,-2,-2,-4
V,0,-3,-3,-3,-1,-2,-2,-3,-3,3,1,-2,1,-1,-2,-2,0,-3,-1,4,-4
L,-1,-2,-3,-4,-1,-2,-3,-4,-3,2,4,-2,2,0,-3,-2,-1,-2,-1,1,-4
H,-2,0,1,-1,-3,0,0,-2,8,-3,-3,-1,-2,-1,-2,-1,-2,-2,2,-3,-4
E,-1,0,0,2,-4,2,5,-2,0,-3,-3,1,-2,-3,-1,0,-1,-3,-2,-2,-4
R,-1,5,0,-2,-3,1,0,-2,0,-3,-2,2,-1,-3,-2,-1,-1,-3,-2,-3,-4
R,-1,5,0,-2,-3,1,0,-2,0,-3,-2,2,-1,-3,-2,-1,-1,-3,-2,-3,-4
Q,-1,1,0,0,-3,5,2,-2,0,-3,-2,1,0,-3,-1,0,-1,-2,-1,-2,-4
F,-2,-3,-3,-3,-2,-3,-3,-3,-1,0,0,-3,0,6,-4,-2,-2,1,3,-1,-4
A,4,-1,-2,-2,0,-1,-1,0,-2,-1,-1,-1,-1,-2,-1,1,0,-3,-2,0,-4
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
-,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,1
```

2.4.2 The model's architecture

The neural networks are implemented as *Keras* sequential models with the following architecture:

1. Convolutional layer with *ReLu* activation.
2. Maxpooling layer.
3. Dense (or fully connected) layer with parameter-defined activation function.

The initial weights of both the first and third layers are randomly generated from a normal distribution with zero mean and a standard deviation defined by the parameter *initializeStd* (0.01 by default).
The filters of the convolutional layers are of size $l \times 21$ where l (the length of the binding core) is defined in the parameter file (9 by default) and 21

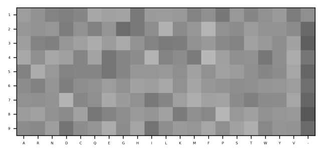
are the 20 amino acids plus the absence of amino acid. The strides are of size 1×21, so that, in practice, the filters will only convolute along the first dimension with stride 1. The model therefore optimizes the search for l-mer core binders contained within a peptide.
The number of filters is defined in the parameter file. Multiple different numbers can be chosen and *rep* neural networks will be trained for each number of filters, where *rep* is the number of repetitions so that the final model is trained from different initial configurations. By default, the final model will be an equally weighted ensemble of 40 neural networks: 10 of them with 5 filters, 10 with 10 filters, 10 with 20 filters and 10 with 30 filters.
The pooling size of the maxpooling layer is of size $m \times 1$ with stride 1×1. The parameter m can be set as *nMaxPool* in the parameter file (Appendix B.2.2). By default m is defined as follows:

$$m := \max(\{6, L_{max} - l - L_{freq} + \mathit{nbPrev} + \mathit{nbAfter} + 2\})$$

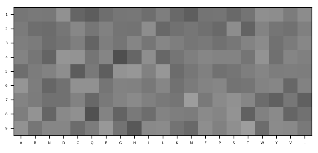
where L_{max} is the maximal peptide length in the training data set, L_{freq} the most frequent one and *nbPrev* and *nbAfter* are the number of characters ”-” added at the beginning and end of each sequence (see Section 2.4.1). The formula for m is defined to make sure that it will neither be too small (the minimal value is 6) nor too big compared to L_{max} which changes from data set to data set; it will control the size of the maxpooling layer's output to be equal to $L_{freq} \times F$, where F is the number of filters. Indeed, the height of the input image is $h := L_{max} + \mathit{nbPrev} + \mathit{nbAfter}$; therefore, the size of the convolutional layer's output is $(h - l + 1) \times F$ and the size of the maxpooling layer's output is $(h - l - m + 2) \times F = L_{freq} \times F$. However, m has a minimum value of 6 to avoid having a pool size too small, so the first dimension of the maxpooling layer's output might be smaller.
The weight optimization is done with a mini batch stochastic gradient descent with parameters defined in the parameters file (Appendix B.2.2).

2.4.3 Visualization of the feed-forward pass

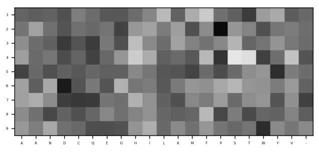
Calling the function *feedForwardVisualization* will generate images to visualize the feed-forward pass of each net in the trained model on the sequences given as input of the function. The results will be saved in a folder called *feed_forward_visualization* in the saving pathway *savePath* of the class.
You can call this function through a class with a trained model. It can be applied to a previously trained and saved model in the following way:



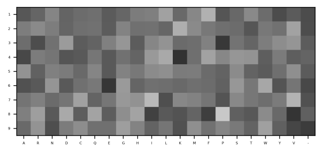
(a) filter_1



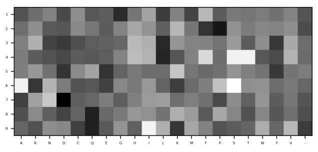
(b) filter_2



(c) filter_3



(d) filter_4



(e) filter_5

Figure 2.3: The 5 filters of the convolutional layer.

```
from model_initializer import CNNPepPred
path_to_trained_model = ...
s = ['KPTHFTVLTKGAGK', 'SEIQYKILTQKEDD', 'TAVFLAAGVGMRL']
myModel = CNNPepPred(trainedModelsFile=path_to_trained_model,
   doApplyData=True,applyData=s)
yhat = myModel.feedForwardVisualization(myModel.applyData)
```

where *path_to_trained_model* is either a *.pkl* saved class object or the path of the saved model (like the input *trainedModelsFile* in the template, see Appendix B.2.3).

In Figure 2.3 we present an example of this visualization for the 5 filters of the convolutional layer of one net. These filters are of size 9×21, where the columns are the amino acids *ARNDCQEGHILKMFPSTWYV-* and 9 is the length of the *l*-mers that the filters will highlight. Pixels with higher values are in white.

The weights in the filters could be thought of as PSSM matrices highlighting particular amino acids at given positions within a nonamer. For example, the filter (e) in Figure 2.3 will activate nonamers containing the amino acids S and T in position 4, A and S in position 6 and I in position 9 (and to some

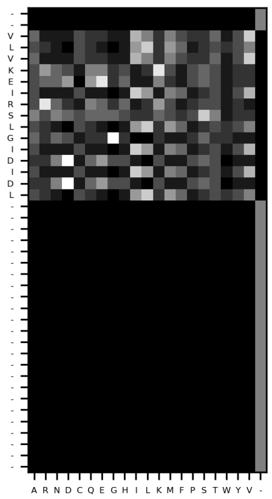


Figure 2.4: The encoded input image

extent F in position 1 and N in position 7).
The encoded image of the peptide sequence *VLVKEIRSLGIDIDL* is printed below in Figure 2.4, with *nbPrev* and *nbAfter* equal to 2 and the maximal length in the training data set being 37.
After the convolution of the filters on the peptide's encoding image, the output is printed in Figure 2.5, where each column represents the output after the convolution of each filter $F1$, $F2$, $F3$, $F4$ and $F5$. The rows corresponds to the application of each filter on an overlapping nonamer of the sequence --*VLVKEIRSLGIDIDL*- -. The overlapping nonamers of the original peptide *VLVKEIRSLGIDID* are labelled *l_x* where x is the start position of the nonamer in this sequence. The overlapping nonamers containing the characters '-' that were added before the first residue of the original peptide are labelled *p_x*, where x is an integer that decreases when approaching the first nonamer fully composed of the original peptide's residues, i.e. without special characters added. Similarly, the overlapping nonamers containing the added characters after the last residue of the peptide are labelled *a_x*, where x increases when moving away from the last nonamer fully composed of the original peptide's resiudes.
The output of the maxpooling layer is printed in Figure 2.6, where the image

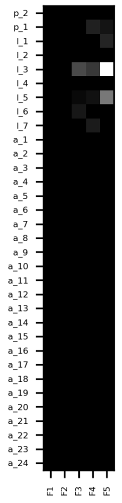
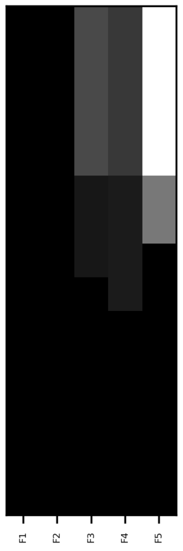
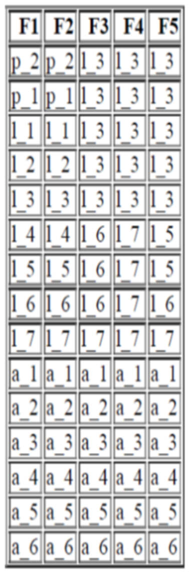


Figure 2.5: The output of the convolutional layer.





F1	F2	F3	F4	F5
p_2	p_2	l_3	l_3	l_3
p_1	p_1	l_3	l_3	l_3
l_1	l_1	l_3	l_3	l_3
l_2	l_2	l_3	l_3	l_3
l_3	l_3	l_3	l_3	l_3
l_4	l_4	l_6	l_7	l_5
l_5	l_5	l_6	l_7	l_5
l_6	l_6	l_6	l_7	l_6
l_7	l_7	l_7	l_7	l_7
a_1	a_1	a_1	a_1	a_1
a_2	a_2	a_2	a_2	a_2
a_3	a_3	a_3	a_3	a_3
a_4	a_4	a_4	a_4	a_4
a_5	a_5	a_5	a_5	a_5
a_6	a_6	a_6	a_6	a_6

Figure 2.6: Left: the output of the MaxPool layer. Right: the argument of each pixel in the output.

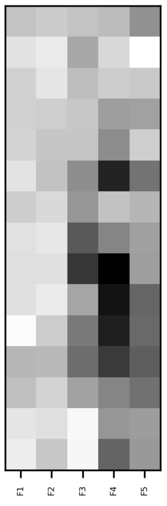


Figure 2.7: The weights of the dense layer.

on the left is the output of the maxpooling layer with each column corresponding to a filter and the table on the right corresponds to the argument of each pixel in the image. In other words, each cell of the table corresponds to a pixel in the output image and the content of this cell is the overlapping nonamer (labelled as described above) that was selected during the maxpooling layer. The table will be saved as an html table.
Finally, the dense layer with the below weights (Figure 2.7) is applied to this output to obtain the net's prediction. Note that the biases of the net are not represented in this visualization.
The feed-forward pass visualization will generate many images; it is therefore recommended to first select a small subset of peptides of interest and only then call the function with this subset.

2.4.4 The contribution score

Let l be the length of the core binder. We define here the *contribution score* associated to each of the overlapping l-mers of a peptide sequence. This score can be understood as the relative importance of an l-mer to the predicted outcome of the corresponding peptide.

For a fixed peptide, let s be any of its overlapping l-mers and we will give a brief description of how the model is applied with respect to s.
The first layer of the model is a convolutional layer with F filters and the corresponding output layer consists of one value for each overlapping l-mer and each filter, i.e. each filter is applied to each overlapping l-mer to obtain an output layer with size the number of overlapping l-mers times the number of filters. Let $x_i^{(s)}$ denote the output value after the application of the filter i to the l-mer s for $i = 1, ..., F$. The activation function of this layer is the *ReLu* activation, i.e. $x_i^{(s)}$ will be mapped to 0 if it is negative and will remain unchanged otherwise. For ease of notation, let $x_i^{(s)}$ be the output value after the *ReLu* activation.
The second layer is a maxpooling layer, therefore only the maximal values will remain with possible repetitions, i.e. for filter i, the value of the application of the model so far to s will be $m_i \cdot x_i^{(s)}$ where m_i is a positive integer (including zero).
The final layer is a dense layer with a parameter-defined activation function σ. This layer will multiply all the values by the weights $d_i^{(s)}$ of the layer and sum them to obtain one remaining value. Keeping only the terms related to s, we define

$$w^{(s)} := \sum_i d_i^{(s)} \cdot m_i \cdot x_i^{(s)}$$

which corresponds to the application of the model restricted to s. In particular, the predicted outcome $\hat{y}$ will be

$$\hat{y} = \sigma \left(\sum_{s'} w^{(s')} + b \right)$$

where b is the bias of the dense layer.
Therefore, the relative contribution of s, with respect to the other l-mers, to the predicted outcome can be thought of as

$$\phi^{(s)} := \frac{w^{(s)}}{\sum_{s'} w^{(s')}}$$

where we define $\phi^{(s)}$ to be the contribution score of s. Note that this value can be smaller than 0 and bigger than 1.
The final model is an ensemble of N convolutional neural networks. Let $w_n^{(s)}$ be the above defined value for the net n and let b_n be its last layer's bias,

then the predicted outcome is

$$\hat{y} = \frac{1}{N} \sum_n \sigma \left(\sum_{s'} w_n^{(s')} + b_n \right)$$

and we define the contribution score of s for an ensemble of nets to be

$$\phi^{(s)} := \frac{\sum_n w_n^{(s)}}{\sum_n \sum_{s'} w_n^{(s')}}.$$

Note that $\sum_{s'} \phi^{(s')} = 1$ and, if σ is the linear activation, then $\phi^{(s)} = \frac{\sum_n w_n^{(s)}}{N\hat{y} - \sum_n b_n}$. The predicted binding core, s_{core}, is then defined to be the overlapping l-mer of the peptipe with the highest contribution score, i.e.

$$s_{\text{core}} \in \text{argmax}_{s'}(\phi^{(s')}).$$

2.4.5 Transfer learning

Transfer learning uses previously trained models to solve new similar problems. A guide on transfer learning with Keras is available at: `https://keras.io/guides/transfer_learning/`.
In the context of MHC-class II peptide binding prediction, if a model trained for a given allele is available, transfer learning can be used to fit new data from another similar allele. In CNN-PepPred, transfer learning is implemented in two steps:

1. The convolution layer is extracted from the given previously trained model and is used as the corresponding layer for the new model. The weights of this layer are frozen, i.e. they will not be updated during the optimization. A first round of optimization is performed, where only the weights of the dense layer are updated.

2. The weights of the convolutional layer are unfrozen and a second round of optimization is performed where all weights are updated.

In the parameter file (Appendix B.2.2), two different values can be given for the optimization parameters (*alpha*, *gamma*, *maxEpochs*, *miniBatchSize*) each corresponding to the two different rounds of optimization. The second step can be skipped by setting the corresponding value of *maxEpochs* to 0.

Transfer learning can be used from the template (Appendix B.2.3) by inputting training data (in *trainingDataPath*) and a trained model (in *trainedModelsFile*).
Transfer learning can lead to overfitting and some considerations must be taken with the optimization parameters. It is recommended to reduce the number of epochs *maxEpochs* (for example setting it to 15) and/or reduce the learning rate *alpha* (for example to 0.001). The first step will correspond to the same model as the pre-trained model with a dense layer fitted to the new data. The second step will fine-tune this model to fit the new data. It is therefore possible that the final model differs from the pre-trained one if the learning is too large and it can also lead to overfitting if the data from the two models are similar.
In Sections 2.5.1.4 and 2.5.3, cross-validation and evaluation results using transfer learning with IEDB data (Section 2.5.1.1) are presented.

2.5 Discussion: application and benchmarking

2.5.1 IEDB data

2.5.1.1 Data preparation

We extracted the data from the IEDB web page `https://www.iedb.org/mhcdetails_v3.php` in the *Assays* tab with filters *Epitope Structure Type: Linear Epitopes*, *Host Organism: Homo sapiens (human)* and *assay-mhc_allele-mhc_Blass: II*. The outcome was taken from the column *qualitative_measure*, the sequences with value *Positive* and *Positive-High* were tagged as binders (1), the ones with value *Positive-Low* and *Positive-Intermediate* were ignored as they might be too weak binders and the rest of the sequences with value *Negative* were tagged as non-binders (0). For each allele's data set, if there were more non-binders than binders, a subset of non-binders was selected at random to balance the data set. If there were more binders than non-binders, the set was balanced using the script *generateRandomNonBinders.py* (Appendix B.2.6). This script generates a given number of non-binders selected from FASTA sequences in a given folder. The sequences used to randomly select non-binders were retrieved from `https://www.uniprot.org/uniparc/`. To improve the computational speed, we only downloaded some batches of se-

quences from UniParc, namely all the entries starting with *UPI00XX* where *XX*$= 00, 01, 02, ..., 10, 11$. There were then more than 70 millions sequences. In order to avoid repetitive sequences in an allele's data set, which can bias the training and testing of the model, for each of the unique overlapping 11-mers contained in one class (binding or non-binding) of the allele's data, only the shortest peptide containing the 11-mer was included. The length 11 was selected because it can remove most of the repetitive sequences without being as restrictive as the length 9 (i.e. the length of the binding core).
Moreover, the cross-validation partition was set to avoid testing with peptides containing too many nonamers also contained in the training data (see next subsection).
Only alleles containing at least 100 positive peptides were included.

2.5.1.2 Cross-validation result

We performed a k-fold cross-validation with k$= 5$ on the allele specific data retrieved as described in the previous subsection. The cross-validation partition was generated using a simple approach in order to reduce the number of l-mers present in both the training and testing data, where l is the length of the core binder ($l= 9$ here).
First, a random cross-validation partition is generated. Then the l-mers shared between the training and testing splits of the random cross-validation partition (within each positive or negative class) are selected. Finally, the peptides containing each of the previously selected l-mers are re-assigned to the fold which occurs the most in the set of peptides sharing the same l-mer. In this way, the number of l-mers shared between folds is greatly reduced compared to a random assignment and all of the cross-validation partitions have a similar number of peptides. Note that this procedure doesn't guarantee that the folds won't share any l-mers. Such a procedure would likely be computationally expensive and could lead to very imbalanced partitions.
This procedure was implemented as *generateCVpartWithLeastLmerOverlap* and if cross-validation is selected in the template and no partition is given with the training data, the model assigns a partition that is fixed before training using this procedure. Moreover, this function will also count the number of overlapping l-mers between each of the training and testing splits and return the average count per split; it will be saved as an attribute called *averageLmersOverlappingCV*.
The cross-validation results are reported in the table below with the fol-

lowing scores: AUC (area under the curve), MCC (Matthews correlation coefficient), ACC (accuracy), F1 (F1-score), where we used CNN-PepPred with default parameters, namely an ensemble of 10 nets per number of filters (5/10/20/30), totalling 40 nets.
We also include the results for the same cross-validation folds using the NNAlign-2.1 method (see Section 2.5.2 for more details on NNAlign). We trained NNAlign using 10 seeds with 5,10,20,30 hidden neurons and the option 'Impose amino acid preference at P1 during burn-in' set to true, the cross-validation partition was given as input and the rescaling of the outcome was set to "No rescale", since the outcome was binary. Note that while NNAlign was rather meant for regression on a quantitative outcome, our model was also set to optimize the mean squared error (this can be set in the parameters), so that it could have been used with the exact same parameters on a quantitative outcome, just like NNAlign.
The best scores are highlighted in bold. For most alleles, CNN-PepPred outperformed NNAlign-2.1.

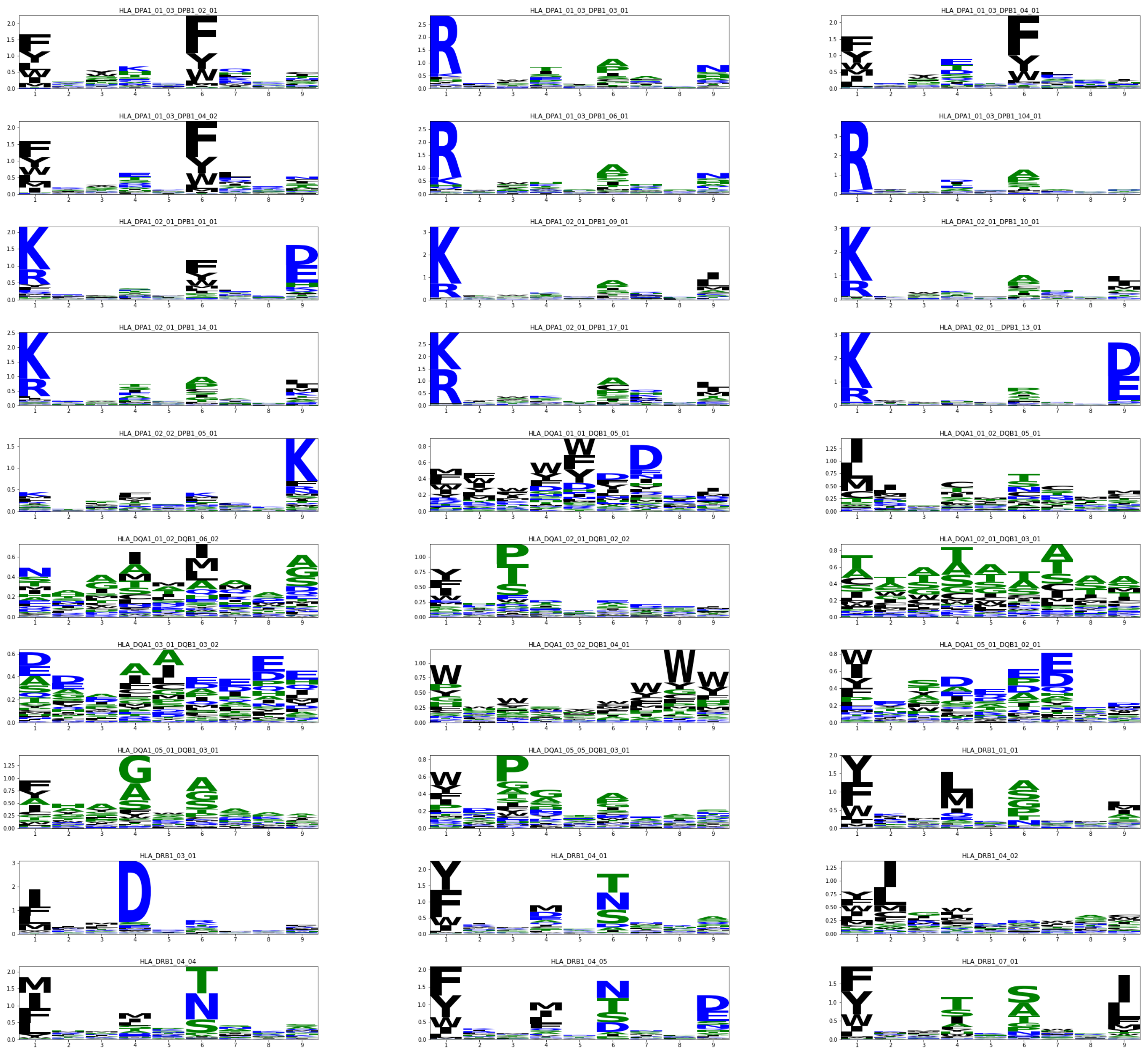
Allele	#Peptide	#Binder	CNN-PepPred				NNAlign-2.1			
			AUC	MCC	ACC	F1	AUC	MCC	ACC	F1
HLA_DPA1_01_03_DPB1_02_01	5177	2589	**0.951**	**0.763**	**0.88**	**0.874**	0.937	0.729	0.864	0.861
HLA_DPA1_01_03_DPB1_03_01	5025	2512	**0.959**	**0.803**	**0.901**	**0.898**	0.936	0.741	0.87	0.868
HLA_DPA1_01_03_DPB1_04_01	7984	3993	**0.946**	**0.747**	**0.872**	**0.865**	0.924	0.701	0.85	0.847
HLA_DPA1_01_03_DPB1_04_02	4643	2322	**0.971**	**0.839**	**0.919**	**0.917**	0.952	0.791	0.895	0.895
HLA_DPA1_01_03_DPB1_06_01	946	473	**0.965**	**0.784**	**0.891**	**0.887**	0.952	0.772	0.886	0.885
HLA_DPA1_01_03_DPB1_104_01	300	150	**0.99**	0.887	0.943	0.944	0.982	**0.9**	**0.95**	**0.95**
HLA_DPA1_02_01_DPB1_01_01	4292	2146	**0.969**	**0.829**	**0.914**	**0.913**	0.948	0.768	0.884	0.884
HLA_DPA1_02_01_DPB1_09_01	2692	1346	**0.972**	**0.835**	**0.917**	**0.916**	0.944	0.767	0.883	0.884
HLA_DPA1_02_01_DPB1_10_01	3628	1814	**0.97**	**0.822**	**0.911**	**0.909**	0.933	0.722	0.861	0.859
HLA_DPA1_02_01_DPB1_14_01	6035	3018	**0.966**	**0.808**	**0.904**	**0.902**	0.924	0.712	0.856	0.855
HLA_DPA1_02_01_DPB1_17_01	2170	1085	**0.974**	**0.839**	**0.919**	**0.918**	0.941	0.738	0.869	0.87
HLA_DPA1_02_01__DPB1_13_01	1968	984	**0.975**	**0.845**	**0.922**	**0.919**	0.969	0.826	0.913	0.913
HLA_DPA1_02_02_DPB1_05_01	7889	3945	**0.96**	**0.799**	**0.899**	**0.898**	0.922	0.708	0.854	0.854
HLA_DQA1_01_01_DQB1_05_01	208	104	**0.94**	0.741	0.87	0.867	**0.94**	**0.77**	**0.885**	**0.887**
HLA_DQA1_01_02_DQB1_05_01	410	206	0.764	0.362	0.68	0.668	**0.774**	**0.42**	**0.71**	**0.705**
HLA_DQA1_01_02_DQB1_06_02	1498	749	**0.915**	**0.672**	**0.835**	**0.829**	0.88	0.624	0.812	0.809
HLA_DQA1_02_01_DQB1_02_02	5772	2886	**0.901**	**0.653**	**0.826**	**0.82**	0.862	0.572	0.786	0.783
HLA_DQA1_02_01_DQB1_03_01	256	128	0.884	0.603	0.801	0.794	**0.904**	**0.664**	**0.832**	**0.83**
HLA_DQA1_03_01_DQB1_03_02	350	175	0.783	0.402	0.7	0.685	**0.795**	**0.475**	**0.737**	**0.729**
HLA_DQA1_03_02_DQB1_04_01	206	103	0.792	0.488	0.743	0.728	**0.815**	**0.564**	**0.782**	**0.789**
HLA_DQA1_05_01_DQB1_02_01	4051	2025	**0.872**	**0.574**	**0.786**	**0.776**	0.829	0.512	0.755	0.746
HLA_DQA1_05_01_DQB1_03_01	617	307	**0.909**	**0.668**	**0.833**	**0.825**	0.904	0.658	0.828	0.821
HLA_DQA1_05_05_DQB1_03_01	5882	2941	**0.889**	**0.63**	**0.815**	**0.811**	0.853	0.549	0.774	0.769
HLA_DRB1_01_01	12412	6208	**0.824**	**0.492**	**0.744**	**0.73**	0.795	0.445	0.722	0.711
HLA_DRB1_03_01	2178	1089	**0.866**	**0.553**	**0.775**	**0.763**	0.85	0.54	0.769	0.76
HLA_DRB1_04_01	5110	2557	**0.846**	**0.544**	**0.77**	**0.755**	0.834	0.533	0.765	0.752
HLA_DRB1_04_02	256	128	**0.764**	**0.469**	**0.734**	**0.73**	0.744	0.423	0.711	0.699
HLA_DRB1_04_04	3076	1538	**0.801**	**0.447**	**0.723**	**0.716**	0.754	0.376	0.687	0.675
HLA_DRB1_04_05	3972	1986	**0.913**	**0.676**	**0.837**	**0.83**	0.887	0.631	0.814	0.807
HLA_DRB1_07_01	4466	2233	**0.916**	**0.684**	**0.841**	**0.835**	0.907	0.675	0.837	0.834
HLA_DRB1_08_01	1118	559	**0.96**	**0.827**	**0.913**	**0.911**	0.957	0.806	0.903	0.904
HLA_DRB1_08_02	838	419	0.829	0.49	0.745	0.737	**0.839**	**0.523**	**0.761**	**0.758**
HLA_DRB1_09_01	1056	528	**0.906**	**0.672**	**0.836**	**0.835**	0.889	0.631	0.815	0.816
HLA_DRB1_10_01	2582	1291	**0.969**	**0.833**	**0.917**	**0.916**	0.963	0.825	0.912	0.913
HLA_DRB1_11_01	4180	2089	**0.917**	**0.665**	**0.832**	**0.826**	0.904	0.655	0.827	0.824
HLA_DRB1_11_03	422	211	**0.956**	**0.853**	**0.927**	**0.926**	0.95	0.801	0.9	0.901
HLA_DRB1_12_01	992	496	**0.966**	**0.809**	**0.904**	**0.903**	0.961	0.806	0.903	**0.903**
HLA_DRB1_13_01	1287	643	**0.935**	**0.74**	**0.869**	**0.865**	0.917	0.699	0.849	0.848
HLA_DRB1_13_02	1460	731	0.885	**0.607**	**0.803**	**0.802**	**0.886**	0.599	0.799	0.796
HLA_DRB1_13_03	1966	983	**0.986**	0.894	0.947	0.947	0.984	**0.896**	**0.948**	**0.948**
HLA_DRB1_14_01	681	340	**0.988**	**0.918**	**0.959**	**0.959**	0.971	0.88	0.94	0.94
HLA_DRB1_14_54	788	394	**0.998**	**0.959**	**0.98**	**0.98**	0.995	0.947	0.973	0.974
HLA_DRB1_15_01	4400	2201	**0.887**	**0.615**	**0.806**	0.796	0.876	0.607	0.803	**0.798**
HLA_DRB1_16_01	423	211	**0.959**	**0.822**	**0.91**	**0.907**	0.948	0.778	0.889	0.89
HLA_DRB3_01_01	1280	640	**0.951**	**0.819**	**0.905**	**0.899**	**0.951**	0.798	0.898	0.896
HLA_DRB3_02_02	1386	694	**0.979**	**0.865**	**0.932**	**0.931**	0.974	0.853	0.926	0.926
HLA_DRB3_03_01	210	105	**0.963**	**0.803**	**0.9**	**0.896**	0.956	0.784	0.89	0.895
HLA_DRB4_01_01	1316	658	**0.914**	**0.654**	**0.827**	**0.822**	0.897	0.628	0.814	0.812
HLA_DRB4_01_03	856	428	**0.971**	0.824	0.911	0.908	0.967	**0.832**	**0.916**	**0.916**
HLA_DRB5_01_01	3329	1664	**0.913**	0.676	0.837	0.833	0.911	**0.678**	**0.839**	**0.837**
HLA_DRB5_02_02	926	463	**0.989**	**0.92**	**0.96**	**0.96**	0.98	0.892	0.946	0.947
Average			**0.921**	**0.716**	**0.857**	**0.853**	0.907	0.691	0.845	0.843
Average weighted by #Binder			**0.919**	**0.702**	**0.85**	**0.845**	0.896	0.657	0.828	0.824

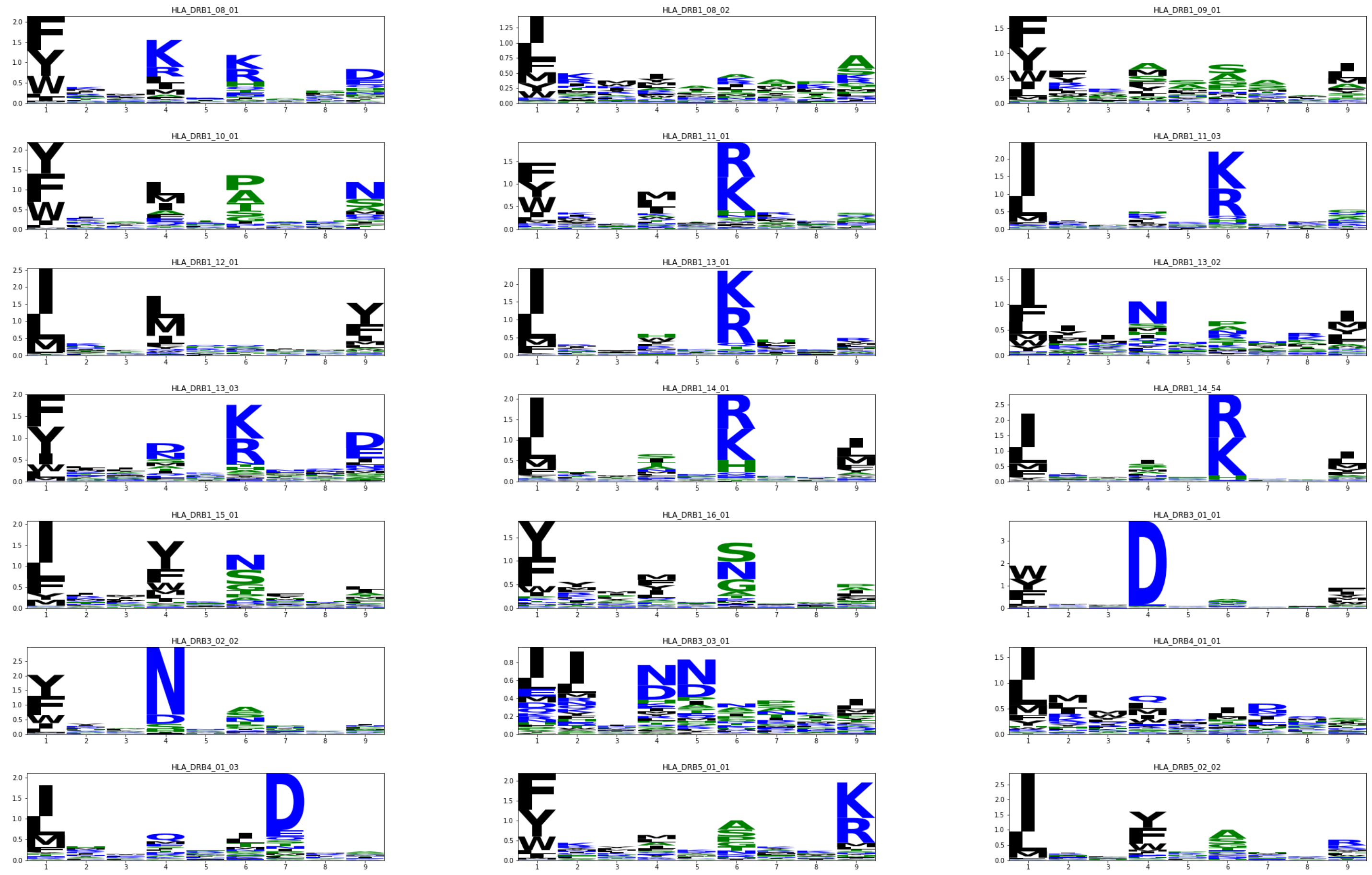
The table below shows the average number of shared nonamers between training/testing splits as given by the output *averageLmersOverlappingCV* of the function *generateCVpartWithLeastLmerOverlap*.
We also compared the computation times of CNN-PedPred (with GPU) and NNAlign for cross-validation. For CNN-PepPred, the total run time, including cross-validation, training on the full training data set and logo plot, is given in a separate column (called *total*) than the run time for cross-validation alone (called *cv*).
It can be observed that CNN-PepPred is generally faster for the alleles with a larger number of sequences and slower for the alleles with a smaller number of sequences. It also seems that with GPU, alleles that were computed towards the end (the order in the table corresponds to the chronological order of computation) used more time than those computed at the beginning. This is likely due to suboptimal implementation for consecutive runs, possibly in connection with the low-level GPU used (NVIDIA GeForce GTX 1080 under Windows OS). CPU runs were performed on the same computer, with an AMD Ryzen 7 1700 8-core, under Windows Susbsystem for Linux (WSL), since the NNAlign executable requires a Linux OS. We couldn't perform the GPU runs on WSL since the system does not support it. This benchmark is only indicative, since WSL is known to decrease performance by about 30% in average compared to native Linux and performance is anyway bound to the specific CPU and GPU used.

Allele	#Peptide	#Binder	Average number of shared nonamers between training/testing splits	CNN-PepPred time[s]: total	CNN-PepPred cv	NNAlign-2.1 time[s]
HLA_DPA1_01_03_DPB1_02_01	5177	2589	5.6	806	619	1610
HLA_DPA1_01_03_DPB1_03_01	5025	2512	0.8	811	624	1548
HLA_DPA1_01_03_DPB1_04_01	7984	3993	6.4	1292	1015	2304
HLA_DPA1_01_03_DPB1_04_02	4643	2322	0.4	824	634	1553
HLA_DPA1_01_03_DPB1_06_01	946	473	1.6	365	282	344
HLA_DPA1_01_03_DPB1_104_01	300	150	0	317	239	186
HLA_DPA1_02_01_DPB1_01_01	4292	2146	0.8	895	690	1353
HLA_DPA1_02_01_DPB1_09_01	2692	1346	0	686	528	890
HLA_DPA1_02_01_DPB1_10_01	3628	1814	2	842	657	1146
HLA_DPA1_02_01_DPB1_14_01	6035	3018	0.8	1245	972	1888
HLA_DPA1_02_01_DPB1_17_01	2170	1085	0	694	540	757
HLA_DPA1_02_01__DPB1_13_01	1968	984	0	704	543	696
HLA_DPA1_02_02_DPB1_05_01	7889	3945	4.8	1528	1221	2213
HLA_DQA1_01_01_DQB1_05_01	208	104	0	196	138	159
HLA_DQA1_01_02_DQB1_05_01	410	206	0	240	170	222
HLA_DQA1_01_02_DQB1_06_02	1498	749	8.8	412	306	545
HLA_DQA1_02_01_DQB1_02_02	5772	2886	12.4	955	744	1512
HLA_DQA1_02_01_DQB1_03_01	256	128	0	289	214	181
HLA_DQA1_03_01_DQB1_03_02	350	175	0	344	251	219
HLA_DQA1_03_02_DQB1_04_01	206	103	0	335	253	156
HLA_DQA1_05_01_DQB1_02_01	4051	2025	13.6	863	665	1138
HLA_DQA1_05_01_DQB1_03_01	617	307	28.4	440	332	308
HLA_DQA1_05_05_DQB1_03_01	5882	2941	9	1076	844	1620
HLA_DRB1_01_01	12412	6208	138.8	1686	1316	3987
HLA_DRB1_03_01	2178	1089	7.2	449	330	893
HLA_DRB1_04_01	5110	2557	11.2	874	670	1762
HLA_DRB1_04_02	256	128	0.8	250	184	186
HLA_DRB1_04_04	3076	1538	3.6	675	493	1026
HLA_DRB1_04_05	3972	1986	5.2	825	626	1301
HLA_DRB1_07_01	4466	2233	9.4	921	715	1549
HLA_DRB1_08_01	1118	559	0	503	372	425
HLA_DRB1_08_02	838	419	6	477	358	367
HLA_DRB1_09_01	1056	528	8	527	404	440
HLA_DRB1_10_01	2582	1291	1.6	745	586	880
HLA_DRB1_11_01	4180	2089	10.8	1005	792	1470
HLA_DRB1_11_03	422	211	0.8	509	390	221
HLA_DRB1_12_01	992	496	1.2	655	493	399
HLA_DRB1_13_01	1287	643	2.4	727	581	527
HLA_DRB1_13_02	1460	731	4.8	736	580	558
HLA_DRB1_13_03	1966	983	0	811	648	688
HLA_DRB1_14_01	681	340	0	643	517	286
HLA_DRB1_14_54	788	394	0	669	536	317
HLA_DRB1_15_01	4400	2201	11.2	1208	952	1518
HLA_DRB1_16_01	423	211	0.4	680	531	233
HLA_DRB3_01_01	1280	640	2.8	811	650	484
HLA_DRB3_02_02	1386	694	0.4	852	683	517
HLA_DRB3_03_01	210	105	0.4	702	574	160
HLA_DRB4_01_01	1316	658	4	977	760	521
HLA_DRB4_01_03	856	428	0.8	905	745	353
HLA_DRB5_01_01	3329	1664	8.8	1264	1016	1145
HLA_DRB5_02_02	926	463	0	922	753	366
Average	2646.37	1323.29	6.59	748.37	583.06	884.84

2.5.1.3 Binding motive

The binding motives are obtained by generating 200000 random 15-mer peptides and plotting, with the package *logomaker* Tareen and Kinney (2019), the core binders of the top 2000 highest predictions.
Below are the binding motives of the alleles retrieved from the IEDB website. In some cases, such as allele HLA_DRB3_03_01 or some of the DQ alleles, misalignments can be observed in the plots. For the prediction, all of the overlapping nonamers contained in the peptide are used, and the binding core is taken as the nonamer that contributes to the final prediction the most (as described in Section 2.4.4). Therefore, the act of selecting a core is relevant for the logo plot but not for the prediction itself. In this regard, logo plots in CNN-PepPred involve a model reduction with loss of information. The missing weights in the logo from other overlapping nonamers contributing also to the binding score adds in this case to the common problems of low number of sequences and sequence bias in some peptide sets, rendering some of the logo plots, such as that for HLA_DRB3_03_01 (105 binding peptides) rather uninformative.


HLA_DPA1_01_03_DPB1_02_01
HLA_DPA1_01_03_DPB1_03_01
HLA_DPA1_01_03_DPB1_04_01
HLA_DPA1_01_03_DPB1_04_02
HLA_DPA1_01_03_DPB1_06_01
HLA_DPA1_01_03_DPB1_104_01
HLA_DPA1_02_01_DPB1_01_01
HLA_DPA1_02_01_DPB1_09_01
HLA_DPA1_02_01_DPB1_10_01
HLA_DPA1_02_01_DPB1_14_01
HLA_DPA1_02_01_DPB1_17_01
HLA_DPA1_02_01__DPB1_13_01
HLA_DPA1_02_02_DPB1_05_01
HLA_DQA1_01_01_DQB1_05_01
HLA_DQA1_01_02_DQB1_05_01
HLA_DQA1_01_02_DQB1_06_02
HLA_DQA1_02_01_DQB1_02_02
HLA_DQA1_02_01_DQB1_03_01
HLA_DQA1_03_01_DQB1_03_02
HLA_DQA1_03_02_DQB1_04_01
HLA_DQA1_05_01_DQB1_02_01
HLA_DQA1_05_01_DQB1_03_01
HLA_DQA1_05_05_DQB1_03_01
HLA_DRB1_01_01
HLA_DRB1_03_01
HLA_DRB1_04_01
HLA_DRB1_04_02
HLA_DRB1_04_04
HLA_DRB1_04_05
HLA_DRB1_07_01

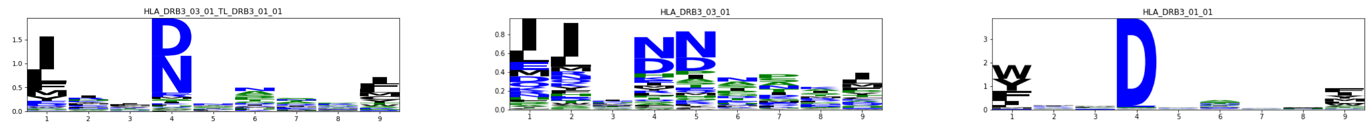


2.5.1.4 Transfer learning results

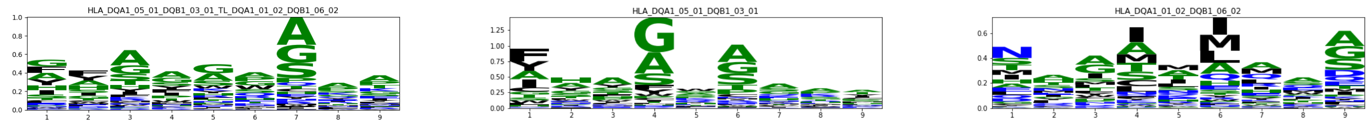
We used the trained IEDB models to perform another round of training using transfer learning (see Section 2.4.5). The previously trained model of an allele was assigned to the training data of another one based on the similarity between their sequences. The same cross-validation set up was used to test the models. The new training instances, which contain nonamers present in the pre-trained model, were removed from the testing set but were kept for training. For this reason, the number of peptides tested in this cross-validation set up is lower than in Section 2.5.1.2. For both optimization steps, the number of epochs was reduced to 15 (compared to 30 by default). The cross-validation results are reported in the table below. The "Allele" column contains first the name of the allele from which the training data were taken, then the allele of the pre-trained model used for transfer learning. Both allele names are separated by "_TL_". The models with transfer learning systematically outperform the ones without. It is however possible that those performances do not generalize to fully new data due to the level of redundancy present in the IEDB data. In Section 2.5.3, we can observe that, while the results on an independent evaluation set are slightly better with transfer learning, the increase in performance on the evaluation set is not proportional to the one in the cross-validation set up.

Allele	CNN-PepPred (transfer learning)						CNN-PepPred					
	#Peptide	#Binder	AUC	MCC	ACC	F1	#Peptide	#Binder	AUC	MCC	ACC	F1
HLA_DPA1_01_03_DPB1_02_01_TL_DPA1_01_03_DPB1_03_01	5025	2458	0.98	0.853	0.926	0.922	5177	2589	0.951	0.763	0.88	0.874
HLA_DPA1_01_03_DPB1_03_01_TL_DPA1_01_03_DPB1_02_01	4876	2372	0.982	0.867	0.933	0.931	5025	2512	0.959	0.803	0.901	0.898
HLA_DPA1_01_03_DPB1_04_01_TL_DPA1_01_03_DPB1_02_01	6764	3111	0.967	0.802	0.902	0.889	7984	3993	0.946	0.747	0.872	0.865
HLA_DPA1_01_03_DPB1_04_02_TL_DPA1_01_03_DPB1_02_01	3889	1578	0.986	0.881	0.943	0.929	4643	2322	0.971	0.839	0.919	0.917
HLA_DPA1_01_03_DPB1_06_01_TL_DPA1_01_03_DPB1_02_01	905	432	0.996	0.94	0.97	0.969	946	473	0.965	0.784	0.891	0.887
HLA_DPA1_01_03_DPB1_104_01_TL_DPA1_01_03_DPB1_02_01	288	138	0.998	0.952	0.976	0.975	300	150	0.99	0.887	0.943	0.944
HLA_DPA1_02_01_DPB1_01_01_TL_DPA1_01_03_DPB1_02_01	3806	1928	0.988	0.903	0.951	0.952	4292	2146	0.969	0.829	0.914	0.913
HLA_DPA1_02_01_DPB1_09_01_TL_DPA1_01_03_DPB1_02_01	2604	1261	0.991	0.922	0.961	0.96	2692	1346	0.972	0.835	0.917	0.916
HLA_DPA1_02_01_DPB1_10_01_TL_DPA1_01_03_DPB1_02_01	3527	1714	0.989	0.9	0.95	0.948	3628	1814	0.97	0.822	0.911	0.909
HLA_DPA1_02_01_DPB1_13_01_TL_DPA1_01_03_DPB1_02_01	1907	925	0.994	0.938	0.969	0.968	6035	3018	0.966	0.808	0.904	0.902
HLA_DPA1_02_01_DPB1_14_01_TL_DPA1_01_03_DPB1_02_01	5859	2846	0.982	0.865	0.932	0.93	2170	1085	0.974	0.839	0.919	0.918
HLA_DPA1_02_01_DPB1_17_01_TL_DPA1_01_03_DPB1_02_01	2094	1010	0.994	0.928	0.964	0.963	1968	984	0.975	0.845	0.922	0.919
HLA_DPA1_02_02_DPB1_05_01_TL_DPA1_01_03_DPB1_02_01	7629	3689	0.976	0.845	0.923	0.92	7889	3945	0.96	0.799	0.899	0.898
HLA_DQA1_01_01_DQB1_05_01_TL_DQA1_01_02_DQB1_06_02	112	72	0.999	0.924	0.964	0.972	208	104	0.94	0.741	0.87	0.867
HLA_DQA1_01_02_DQB1_05_01_TL_DQA1_01_02_DQB1_06_02	406	202	0.978	0.833	0.916	0.915	410	206	0.764	0.362	0.68	0.668
HLA_DQA1_01_02_DQB1_06_02_TL_DQA1_02_01_DQB1_02_02	1324	601	0.975	0.844	0.923	0.915	1498	749	0.915	0.672	0.835	0.829
HLA_DQA1_02_01_DQB1_02_02_TL_DQA1_01_02_DQB1_06_02	5576	2720	0.951	0.768	0.884	0.879	5772	2886	0.901	0.653	0.826	0.82
HLA_DQA1_02_01_DQB1_03_01_TL_DQA1_01_02_DQB1_06_02	256	128	0.986	0.875	0.938	0.938	256	128	0.884	0.603	0.801	0.794
HLA_DQA1_03_01_DQB1_03_02_TL_DQA1_01_02_DQB1_06_02	232	148	0.98	0.869	0.94	0.953	350	175	0.783	0.402	0.7	0.685
HLA_DQA1_03_02_DQB1_04_01_TL_DQA1_01_02_DQB1_06_02	189	102	0.962	0.831	0.915	0.92	206	103	0.792	0.488	0.743	0.728
HLA_DQA1_05_01_DQB1_02_01_TL_DQA1_01_02_DQB1_06_02	3661	1848	0.951	0.755	0.877	0.879	4051	2025	0.872	0.574	0.786	0.776
HLA_DQA1_05_01_DQB1_03_01_TL_DQA1_01_02_DQB1_06_02	336	105	0.975	0.815	0.92	0.873	617	307	0.909	0.668	0.833	0.825
HLA_DQA1_05_05_DQB1_03_01_TL_DQA1_01_02_DQB1_06_02	5656	2724	0.941	0.751	0.876	0.87	5882	2941	0.889	0.63	0.815	0.811
HLA_DRB1_01_01_TL_DRB1_15_01	10674	4935	0.877	0.586	0.794	0.765	12412	6208	0.824	0.492	0.744	0.73
HLA_DRB1_03_01_TL_DRB1_13_01	1887	956	0.961	0.792	0.895	0.893	2178	1089	0.866	0.553	0.775	0.763
HLA_DRB1_04_01_TL_DRB1_04_05	4231	2032	0.907	0.656	0.828	0.812	5110	2557	0.846	0.544	0.77	0.755
HLA_DRB1_04_02_TL_DRB1_04_04	67	54	0.94	0.676	0.896	0.935	256	128	0.764	0.469	0.734	0.73
HLA_DRB1_04_04_TL_DRB1_04_01	2234	1079	0.927	0.7	0.85	0.843	3076	1538	0.801	0.447	0.723	0.716
HLA_DRB1_04_05_TL_DRB1_04_01	3192	1524	0.966	0.809	0.905	0.898	3972	1986	0.913	0.676	0.837	0.83
HLA_DRB1_07_01_TL_DRB1_09_01	3665	1777	0.961	0.804	0.902	0.898	4466	2233	0.916	0.684	0.841	0.835
HLA_DRB1_08_01_TL_DRB1_13_03	1072	514	0.989	0.912	0.956	0.954	1118	559	0.96	0.827	0.913	0.911
HLA_DRB1_08_02_TL_DRB1_08_01	831	412	0.97	0.841	0.921	0.921	838	419	0.829	0.49	0.745	0.737
HLA_DRB1_09_01_TL_DRB1_07_01	306	113	0.966	0.823	0.915	0.89	1056	528	0.906	0.672	0.836	0.835
HLA_DRB1_10_01_TL_DRB1_01_01	2182	943	0.994	0.923	0.962	0.956	2582	1291	0.969	0.833	0.917	0.916
HLA_DRB1_11_01_TL_DRB1_13_03	4012	1945	0.945	0.736	0.868	0.861	4180	2089	0.917	0.665	0.832	0.826
HLA_DRB1_11_03_TL_DRB1_11_01	307	131	0.998	0.953	0.977	0.973	422	211	0.956	0.853	0.927	0.926
HLA_DRB1_12_01_TL_DRB1_13_01	950	470	0.991	0.907	0.954	0.953	992	496	0.966	0.809	0.904	0.903
HLA_DRB1_13_01_TL_DRB1_13_02	1145	558	0.982	0.876	0.938	0.936	1287	643	0.935	0.74	0.869	0.865
HLA_DRB1_13_02_TL_DRB1_13_01	1310	635	0.973	0.843	0.921	0.919	1460	731	0.885	0.607	0.803	0.802
HLA_DRB1_13_03_TL_DRB1_11_01	1823	844	0.995	0.942	0.971	0.969	1966	983	0.986	0.894	0.947	0.947
HLA_DRB1_14_01_TL_DRB1_13_03	668	327	0.997	0.958	0.979	0.979	681	340	0.988	0.918	0.959	0.959
HLA_DRB1_14_54_TL_DRB1_13_03	754	360	1	0.987	0.993	0.993	788	394	0.998	0.959	0.98	0.98
HLA_DRB1_15_01_TL_DRB1_01_01	2981	1350	0.96	0.79	0.896	0.885	4400	2201	0.887	0.615	0.806	0.796
HLA_DRB1_16_01_TL_DRB1_15_01	341	168	0.997	0.965	0.982	0.982	423	211	0.959	0.822	0.91	0.907
HLA_DRB3_01_01_TL_DRB3_02_02	915	521	0.966	0.836	0.916	0.922	1280	640	0.951	0.819	0.905	0.899
HLA_DRB3_02_02_TL_DRB3_01_01	941	587	0.996	0.941	0.972	0.978	1386	694	0.979	0.865	0.932	0.931
HLA_DRB3_03_01_TL_DRB3_01_01	188	83	0.996	0.968	0.984	0.982	210	105	0.963	0.803	0.9	0.896
HLA_DRB4_01_01_TL_DRB1_10_01	1226	595	0.98	0.879	0.94	0.938	1316	658	0.914	0.654	0.827	0.822
HLA_DRB4_01_03_TL_DRB4_01_01	678	251	0.998	0.949	0.976	0.968	856	428	0.971	0.824	0.911	0.908
HLA_DRB5_01_01_TL_DRB1_01_01	2222	927	0.966	0.834	0.919	0.904	3329	1664	0.913	0.676	0.837	0.833
HLA_DRB5_02_02_TL_DRB5_01_01	845	382	0.995	0.941	0.97	0.968	926	463	0.989	0.92	0.96	0.96
Average			0.975	0.857	0.93	0.926			0.921	0.716	0.857	0.853
Average weighted by #Binder			0.962	0.812	0.906	0.9			0.919	0.702	0.85	0.845

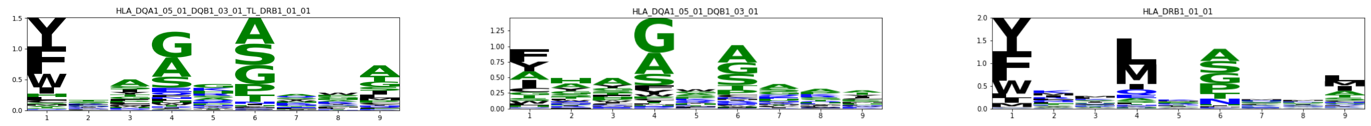
Transfer learning will also affect the logo plots. For example, as it can be seen in the figure below, using the pre-trained model of HLA_DRB3_01_01 has helped aligning the binding motif of the allele HLA_DRB3_03_01.



On the other hand, transfer learning with alleles that do not share strong fixed positions can reduce the alignment. For example, the pre-trained model of HLA_DQA1_01_02_DQB1_06_02 do not help aligning the binding motif of the allele HLA_DQA1_05_01_DQB1_03_01:



However, if instead the pre-trained model of the allele HLA_DRB1_01_01 is used, the shared patterns in position 1 and 6 help aligning the binding motif of the allele HLA_DQA1_05_01_DQB1_03_01:



2.5.2 NetMHCII data

In this appendix, we benchmark the convolutional neural network approach with the state-of-the-art methods from the netMHCII family Jensen *et al.* (2018). The netMHCII family consists of two main methods: an allele-specific one (netMHCII) and a pan-specific one (netMHCIIpan). They are both based on the same core algorithm NNAlign (Nielsen and Lund (2009), Nielsen and Andreatta (2017)) which consists of a two-step optimization procedure that simultaneously estimates the core (nonamer) binder and the network weight configuration for the binding prediction. The pan-specific method is trained with all of the peptides of all of the alleles and can make predictions for all alleles with known alpha and beta chains. The pan-specific version is therefore more adequate for alleles with few training data. However, for alleles with enough training data, the authors report Jensen *et al.*

(2018) that the allele specific method outperforms the pan specific one.
More modern versions of netMHCII include the possibility of training with multi-allele (MA) peptides Reynisson *et al.* (2020). While this is an interesting recent research direction (Alvarez *et al.* (2019),Racle *et al.* (2019),Chen *et al.* (2019)), our aim in developing CNN-PepPred has been to proveide an open-source efficient core algorithm that can be easily integrated in more complex pipelines and modified to fit specific purposes, including the incorporation of MA data.
The convolutional neural network approach is similar to the strategy of netMHCII, since both use similar blosum encoding and rely on an ensemble of neural networks. The main difference is that NNAlign is a two steps procedure that first identifies a core nonamer and applied it (with flanking region) to a network weight configuration. Our model uses convolution to slide through the possible nonamers contained within a peptide, therefore using the peptide in its full length. This strategy is also convenient to implement since it only requires building a sequential convolution neural network using user friendly libraries such as Keras.
NetMHCII methods are web based and meant to be used to predict binding with pre-trained models. While executables are available upon request, the core algorithm training the models (NNAlign) is not open-source and full development details have not been, to our knowledge, provided in any publication. As CNN-PepPred, NNAlign can be also used as an executable to train models with specific data sets.

2.5.2.1 Cross-validation result

The data and the 5 fold cross-validation partition for this set-up were taken from the paper presenting netMHCIIpan-3.2, which is the latest version of the model not including multiple-allele data. The results of netMHCIIpan-3.2 and netMHCII-2.3 were taken from the supplementary file, Suppl Table 3, of Jensen *et al.* (2018). The authors only reported the AUC score but we also included the Pearson correlation (PC) and root mean squared error (RMSE) scores of our model for further information. The best AUC score for each allele is highlighted in bold.
We used CNN-PepPred with default parameters, namely, an ensemble of 10 nets per number of filters (5/10/20/30), totaling 40 nets. The threshold used to binarize the quantitative outcome was set to the log50k transform of 500nM as in Jensen *et al.* (2018), namely $1 - \log(500)/\log(50000) \approx 0.426$.

As it can be seen in the table, if we also include the alleles with few training data (for which allele-specific methods are clearly not fitted), netMHCIIpan outperforms (on average) the two allele specific methods. However, considering different sets of alleles with different minimum numbers of binding training peptides, our model outperforms (on average) the models from the netMHCII family. In any cases, the performances are overall similar.

Allele	#Peptide	#Binder	CNN-PepPred			NetMHCII-2.3	NetMHCIIPan-3.2
			PC	AUC	RMSE	AUC	AUC
DRB1_0101	10412	6376	0.69	**0.837**	0.195	0.829	0.832
DRB1_0103	42	4	-0.231	0.204	0.208	0.25	**0.678**
DRB1_0301	5352	1457	0.646	**0.836**	0.181	0.816	0.816
DRB1_0401	6317	3022	0.613	**0.811**	0.198	0.798	0.809
DRB1_0402	53	19	0.419	0.669	0.249	0.633	**0.701**
DRB1_0403	59	14	0.511	0.703	0.152	0.644	**0.841**
DRB1_0404	3657	1852	0.636	0.803	0.189	0.787	**0.812**
DRB1_0405	3962	1653	0.669	**0.841**	0.171	0.839	0.827
DRB1_0701	6325	3456	0.748	**0.884**	0.171	0.877	0.875
DRB1_0801	937	390	0.658	0.836	0.165	0.834	**0.844**
DRB1_0802	4465	2036	0.673	**0.838**	0.184	0.834	0.834
DRB1_0901	4318	2164	0.657	**0.833**	0.175	0.832	**0.833**
DRB1_1001	2066	1521	0.754	0.915	0.157	0.912	**0.923**
DRB1_1101	6045	2667	0.734	0.866	0.174	**0.867**	0.864
DRB1_1201	2384	759	0.771	**0.894**	0.141	0.891	0.868
DRB1_1301	1034	520	0.673	0.851	0.22	0.828	**0.857**
DRB1_1302	4477	2249	0.774	**0.89**	0.176	0.889	0.885
DRB1_1501	4850	2107	0.679	**0.839**	0.187	0.833	0.834
DRB1_1602	1699	989	0.778	**0.886**	0.151	0.879	0.883
DRB3_0101	4633	1415	0.813	**0.912**	0.149	0.898	0.888
DRB3_0202	3334	1055	0.808	**0.889**	0.171	0.887	0.869
DRB3_0301	884	510	0.646	0.826	0.192	0.824	**0.84**
DRB4_0101	3961	1540	0.706	**0.851**	0.171	0.837	0.822
DRB4_0103	846	525	0.67	**0.849**	0.197	0.839	0.841
DRB5_0101	5125	2430	0.714	**0.855**	0.191	0.849	0.849
H_2_IAb	1794	431	0.703	0.885	0.163	0.884	**0.894**
H_2_IAd	774	321	0.611	0.813	0.202	**0.819**	**0.819**
H_2_IAk	115	4	0.332	0.619	0.137	0.628	**0.635**
H_2_IAs	190	48	0.534	0.815	0.195	0.761	**0.825**
H_2_IAu	56	22	0.603	**0.898**	0.262	0.83	0.765
H_2_IEd	245	28	0.4	0.706	0.18	0.73	**0.754**
H_2_IEk	68	40	0.633	0.754	0.216	0.836	**0.853**
HLA_DPA10103_DPB10201	787	141	0.72	0.903	0.146	0.91	**0.917**
HLA_DPA10103_DPB10301	1563	575	0.796	**0.914**	0.166	**0.914**	0.902
HLA_DPA10103_DPB10401	2725	786	0.882	**0.939**	0.14	0.935	0.935
HLA_DPA10103_DPB10402	45	9	0.194	0.596	0.18	0.497	**0.71**
HLA_DPA10103_DPB10601	584	282	0.958	0.995	0.116	**0.996**	0.995
HLA_DPA10201_DPB10101	2447	859	0.833	0.897	0.149	**0.903**	**0.903**
HLA_DPA10201_DPB10501	2470	713	0.806	0.913	0.154	**0.914**	0.911
HLA_DPA10201_DPB11401	2302	849	0.851	**0.942**	0.151	0.937	0.93
HLA_DPA10301_DPB10402	2641	921	0.834	0.903	0.157	**0.906**	0.904
HLA_DQA10101_DQB10501	2946	815	0.813	**0.917**	0.138	**0.917**	0.9
HLA_DQA10102_DQB10501	833	458	0.662	0.865	0.194	**0.867**	0.839
HLA_DQA10102_DQB10502	800	158	0.675	**0.851**	0.159	0.85	0.835
HLA_DQA10102_DQB10602	2747	1256	0.814	0.902	0.148	**0.905**	0.89
HLA_DQA10103_DQB10603	462	90	0.503	0.803	0.199	0.816	**0.861**
HLA_DQA10104_DQB10503	883	105	0.635	0.837	0.143	**0.844**	0.805
HLA_DQA10201_DQB10202	944	119	0.644	**0.86**	0.131	0.851	0.814
HLA_DQA10201_DQB10301	827	374	0.696	**0.876**	0.187	0.864	0.849
HLA_DQA10201_DQB10303	761	265	0.721	0.886	0.152	0.887	**0.894**
HLA_DQA10201_DQB10402	768	241	0.638	0.854	0.181	0.858	**0.86**
HLA_DQA10301_DQB10301	207	66	0.591	0.774	0.195	0.761	**0.839**
HLA_DQA10301_DQB10302	3111	568	0.702	0.846	0.126	**0.849**	0.81
HLA_DQA10303_DQB10402	567	117	0.632	**0.844**	0.168	0.836	0.82
HLA_DQA10401_DQB10402	2890	928	0.794	**0.903**	0.116	0.894	0.883
HLA_DQA10501_DQB10201	2897	874	0.78	**0.889**	0.131	**0.889**	0.876
HLA_DQA10501_DQB10301	3585	1812	0.812	**0.926**	0.143	0.922	0.915
HLA_DQA10501_DQB10302	847	203	0.6	0.82	0.139	**0.831**	0.822
HLA_DQA10501_DQB10303	564	179	0.68	0.869	0.138	**0.884**	0.876
HLA_DQA10501_DQB10402	749	337	0.718	**0.877**	0.157	0.857	0.868
HLA_DQA10601_DQB10402	565	133	0.622	**0.854**	0.18	0.845	0.848
Average			0.666	0.839	0.17	0.833	**0.847**
Average weighted by #Binder			0.722	**0.865**	0.172	0.86	0.858
Average over alleles with >=100binders			0.723	**0.872**	0.164	0.869	0.864
Average over alleles with >=500binders			0.745	**0.876**	0.165	0.871	0.867
Average over alleles with >=1000binders			0.719	**0.863**	0.174	0.856	0.854

2.5.2.2 Run time comparison

To evaluate computational time, we used different numbers of sequences of the proteome of *Burkholderia pseudomallei* (`https://www.uniprot.org/proteomes/UP000000605`) to the trained model of HLA-DRB1*07:01 (with the data set from NetMHCIIpan3.2). Each sequence was cut into all of its overlapping 15-mers and a prediction was made for all of them. In the table below, the time is reported in seconds for different number of sequences. The full proteome contains 5717 sequences corresponding to 1908278 15-mers. Note that the number of 15-mers corresponds to the non-unique amount (no check performed for sequence identity). The last row of the table corresponds to the application of the model against the human proteome with ca. 75000 proteins (`https://www.uniprot.org/proteomes/UP000005640`); it was added to have an idea of how long the GPU version would take on really big data sets (more than 20 millions non-unique 15-mers). We tested our model using the GPU and CPU versions and we downloaded the latest version of netMHCIIpan4.0 from its webserver (`https://services.healthtech.dtu.dk/service.php?NetMHCIIpan-4.0`) and ran it for allele DRB1*07:01, with length 15 and the print unique binding core option. We used NNAlign with the same parameters as in Section 2.5.1.2(except for output rescaling which was set to the default, since the training outcome is quantitative).
Results can be found in the table below. For the application of a trained model on new instances, the fastest model is NNAlign followed by CNN-PepPred with GPU.
We can also notice that the time grows more or less linearly with increasing number of sequences, which makes sense as our model analyses new sequences in batches of 50000 (by default). The application in batches also means that there will never be a memory issue whatever the number of peptides to analyse. For the GPU computation we used NVIDIA GeForce GTX 1080 under Windows OS. The CPU runs were performed for CNN-PepPred, NNAlign and netMHCIIpan under Windows Subsystem for Linux (WSL) in the same computer (equipped with an AMD Ryzen 7 1700 8-core), since the executables of the latter two require a Linux OS. We couldn't perform the GPU computations on WSL since the system does not support it.

#seq	#15-mers (non-unique)	time[s]			
		CNN-PepPred (cpu)	CNN-PepPred (gpu)	NetMHCIIpan4.0	NNAlign2.1
100	33130	41	27	239	6
500	157401	153	71	1121	12
1000	333632	323	127	2369	24
2000	666696	630	237	4738	44
3000	1006260	953	355	7118	68
4000	1336574	1256	470	9412	89
5000	1663243	1530	581	11754	108
5717	1908278	1774	664	13466	127
75069	24752058		4184		738

2.5.3 Evaluation set

For the evaluation of the models trained with the data retrieved from IEDB as described in Section 2.5.1.1, we used the T-cell epitope benchmark from Jensen et al.(Jensen *et al.* (2018)). This data set contains, for different alleles, several pairs of epitope and epitope-source protein sequences. The epitope source is split into all of its overlapping l-mers, where l is the length of the epitope. The actual epitope is labelled as binding while the overlapping non-epitope l-mers in the epitope source are labelled as non-binding. The trained model is then applied to all l-mers and the evaluation is performed using two metrics: the AUC and the F-rank. The F-rank corresponds to the ratio between the number of peptides from the source with predicted binding score higher than that of the epitope and the total number of peptides in the source. Therefore, a value of 0 means that no peptides are predicted as stronger binders than the epitope and a value of 1 means that all peptides are predicted as stronger binders than the epitope. Both scores are computed for each pair separately and averaged per allele. As noted by the authors, this evaluation will tend to underestimate the performances since some negatively labelled peptides might still be presented by the human MHC molecule.
We selected the epitopes for alleles that are present in our data set (listed in Appendix B.2.5). We then removed a few epitopes that were already present in our training data set. The table below contains the scores of this evaluation. The results for CNN-PepPred are overall similar to the ones of NetMHCIIpan3.2, with a slight advantage on average for CNN-PepPred: the average F-rank/AUC is 0.174/0.825 for CNN-PepPred and 0.193/0.806 for NetMHCIIpan3.2 (with the values as reported by the authors in suppl. table 5 of the paper's supplementary file).

Allele	CNN-PepPred			NetMHCIIpan3.2		
	#Epitope	average F-rank	average AUC	#Epitope	average F-rank	average AUC
HLA_DPA1_01_03_DPB1_02_01	1	**0.005**	**0.995**	1	0.02	0.98
HLA_DQA1_01_02_DQB1_06_02	2	0.085	0.915	2	**0.051**	**0.948**
HLA_DRB1_01_01	235	0.194	0.806	240	**0.181**	**0.818**
HLA_DRB1_03_01	96	0.147	0.853	101	**0.14**	**0.86**
HLA_DRB1_04_01	220	**0.157**	**0.842**	232	0.195	0.804
HLA_DRB1_04_02	3	0.22	0.78	3	**0.206**	**0.793**
HLA_DRB1_04_04	142	0.266	0.734	146	**0.19**	**0.81**
HLA_DRB1_04_05	3	**0.01**	**0.99**	3	0.03	0.964
HLA_DRB1_07_01	196	**0.159**	**0.841**	197	0.179	0.821
HLA_DRB1_08_01	19	**0.199**	**0.801**	22	0.24	0.76
HLA_DRB1_09_01	40	**0.277**	**0.721**	40	0.326	0.672
HLA_DRB1_10_01	9	0.496	0.503	10	**0.328**	**0.672**
HLA_DRB1_11_01	192	**0.106**	**0.894**	196	0.14	0.859
HLA_DRB1_12_01	2	0.139	0.86	2	**0.086**	**0.914**
HLA_DRB1_13_01	12	**0.087**	**0.913**	12	0.245	0.754
HLA_DRB1_13_02	3	**0.293**	**0.706**	3	0.547	0.45
HLA_DRB1_14_01	20	0.234	0.766	20	**0.206**	**0.795**
HLA_DRB1_15_01	113	**0.18**	**0.82**	122	0.184	0.815
HLA_DRB3_01_01	4	**0.034**	**0.966**	4	0.068	0.932
HLA_DRB3_02_02	7	**0.144**	**0.856**	7	0.149	0.85
HLA_DRB4_01_01	3	**0.279**	**0.72**	3	0.372	0.628
HLA_DRB5_01_01	119	**0.123**	**0.877**	120	0.17	0.83
Average		**0.174**	**0.825**		0.193	0.806

This evaluation set up was also performed on the models trained with transfer learning (see Sections 2.4.5 and 2.5.1.4). As discussed in Section 2.5.1.4, while the results are on average slightly better with transfer learning, the increment in predictive performance on the evaluation set does not match the one in the cross-validation set up. This might be due to the redundancy present in the IEDB data. The previously trained models are already adapted to this type of data and using transfer learning helps finding patterns, however it doesn't seem to generalize as well with independent data. Therefore, some caution must be observed when using transfer learning to train and the parameters should be set to reduce the learning during optimization (lower learning rate and/or number of epochs).

Allele	#Epitope	CNN-PepPred		CNN-PepPred (transfer learning)	
		average F-rank	average AUC	average F-rank	average AUC
HLA_DPA1_01_03_DPB1_02_01	1	0.005	0.995	**0.004**	**0.996**
HLA_DQA1_01_02_DQB1_06_02	2	**0.085**	**0.915**	0.196	0.803
HLA_DRB1_01_01	235	**0.194**	**0.806**	**0.194**	**0.806**
HLA_DRB1_03_01	96	**0.147**	**0.853**	0.151	0.849
HLA_DRB1_04_01	220	0.157	0.842	**0.149**	**0.851**
HLA_DRB1_04_02	3	**0.22**	**0.78**	0.294	0.705
HLA_DRB1_04_04	142	**0.266**	**0.734**	0.28	0.719
HLA_DRB1_04_05	3	0.01	0.99	**0.009**	**0.991**
HLA_DRB1_07_01	196	**0.159**	**0.841**	0.165	0.835
HLA_DRB1_08_01	19	**0.199**	**0.801**	0.203	0.797
HLA_DRB1_09_01	40	0.277	0.721	**0.257**	**0.742**
HLA_DRB1_10_01	9	0.496	0.503	**0.45**	**0.55**
HLA_DRB1_11_01	192	**0.106**	**0.894**	0.111	0.889
HLA_DRB1_12_01	2	0.139	0.86	**0.028**	**0.972**
HLA_DRB1_13_01	12	0.087	0.913	**0.084**	**0.916**
HLA_DRB1_13_02	3	0.293	0.706	**0.221**	**0.779**
HLA_DRB1_14_01	20	**0.234**	**0.766**	0.246	0.753
HLA_DRB1_15_01	113	0.18	0.82	**0.168**	**0.832**
HLA_DRB3_01_01	4	0.034	0.966	**0.032**	**0.968**
HLA_DRB3_02_02	7	0.144	0.856	**0.129**	**0.87**
HLA_DRB4_01_01	3	0.279	0.72	**0.227**	**0.773**
HLA_DRB5_01_01	119	**0.123**	**0.877**	0.15	0.85
Average		0.174	0.825	**0.17**	**0.829**

Chapter 3

SICPAC study: Computational simulation of pancreatic cancer patients as clinical decision support system based on machine learning techniques and systems biology

3.1 Abstract

Pancreatic cancer is one of the most aggressive cancer types, with a 5-year survival rate lower than 5%. Despite having approved regimens, it is still necessary to determine the molecular features leading to a low prognosis. In recent years, computational approaches based on systems biology that generate patient models based on multiple data sources have been developed. These approaches have proven to accurately reproduce results from clinical trials, making them suitable for discovery of new biomarkers.
Clinical variables from 20 patients that had previously undergone palliative first-line chemotherapy with Gemcitabine/Nab-paclitaxel were retrieved during multiple hospital visits and corresponding molecular information was simulated through TPMS technology. TPMS is a system biology approach which simulates patients by generating a protein activation network based on clinical information, drug effectors and known relations between proteins. More-

over, the clinical variables along with the protein activations from TPMS models were used for the training of multivariate regression tree models to predict the next visit's outcome value of a patient. Six clinically relevant outcomes were selected: the concentration of eosinophil, platelet, red blood cell, leukocyte, monocyte and hemoglobin.
Although the correlations between predicted and expected outcome values in a validation set up remain too low to build a trustworthy clinical predictive tool, the use of regression trees with clinical data alongside TPMS simulated data revealed to be an accurate systems biology approach to predict increase and decrease trends of the six studied clinical outcomes, especially considering the low number of patients involved in the study. Furthermore, promising results were obtained, such as the identification of the time period between visits and of several clinical and molecular variables as a potential source of variation for the studied outcomes. Such methodology could be further applied to other cancer types using appropriate clinical data. To assist in decision making and render this approach available to interested researchers, a friendly graphic user interface applying the described models has been deployed in the following web-application: `http://sicpac.anaxomics.com:81`.

3.2 Introduction

Cancer is a clinical term that englobes several diseases with well-differentiated histologic characteristics, heterogeneous clinical performances and failed clinical response in many cases. Among all the cancer types, pancreatic cancer is one of the most aggressive ones, with a poor five-year survival rate, lower than 5% (Siegel *et al.*, 2018). A lack in early detection methods along with a tendency for early metastasis contribute to this poor survival rate (Maitra and Hruban, 2008; Garrido-Laguna and Hidalgo, 2015). Minor improvements have been obtained with FOLFIRINOX (folinic acid, 5-fluorouracil [5 FU], irinotecan, and oxaliplatin) and paclitaxel/nab-paclitaxel plus gemcitabine chemotherapy (Conroy *et al.*, 2011; Hoff *et al.*, 2013). Despite these advancements, surgical resection remains as the main treatment procedure for pancreatic cancer patients (Deplanque and Demartines, 2017).
A patient's cancer treatment can be stopped due to efficacy or security issues. Efficacy of a treatment can be monitored with measurements on solid tumors such as TAC, PET, or any other resonance-based imaging techniques. Blood

biomarkers can also be used to assess the malignant impact of the tumor on the affected organ. Patient's general status and tolerability can be screened with markers of the liver, kidney and immune system obtained in blood tests. Patients are evaluated using performance scales such as the Karnofsky and ECOG Performance Status scales to get a complete evaluation considering quality of life-related terms and their ability to perform daily routine tasks. Cancer patients, especially those with advanced-stage cancer diseases, usually display a non-recurrent clinical behavior. On one side, after some time, oncologic treatments become less efficient due to resistance. On the other side, oncologic treatment might induce toxic effects, causing its interruption, variations in dose quantity and their frequency of administration, and changes on the treatment, despite being effective. Alternative treatments that reduce the toxicity induced by the main treatment and comorbidities due to tumors can also be given to cancer patients in parallel.

Every aspect described above makes the continuous evaluation of patients' status necessary for the oncologist, as well as taking decisions about the patients' treatments. Integration of this set of patient and treatment information is not easy. As a solution, a tool based on computational simulations of the patients of interest using mathematics models might be of utility. This tool would be used to explore the effects of the treatment in the following cycle, thus helping the oncologist in the decision-making process.

Computational tools to create patient models from several data sources have gained popularity to determine new biomarkers in various diseases (Gulshan *et al.*, 2016; Kamnitsas *et al.*, 2017; Segú-Vergés *et al.*, 2021). In silico simulation of metabolic pathways, tissues, organs and use of animal models in clinical research has already been put in place extensively in cancer (Schultz *et al.*, 2016; Obrzut *et al.*, 2017; Gal *et al.*, 2020). Computational techniques have been able to replace clinical trials in some drug development processes using systems biology (Lucas *et al.*, 2016; Giménez *et al.*, 2019; Villalba *et al.*, 2020). Systems biology is a multidisciplinary domain that integrates information from molecular biology and several computational and mathematical methods. It has been used to model mechanism of action of drugs, predict novel biomarkers and identify indications. The main goal of systems biology in this area is to obtain computational models of patients that can be used to test different therapeutic strategies. Partial or complete simulation of patients needs to be done to generate these alternative strategies in an in-silico way.

In the last years, several predictive tools that model disease progression, drug

interaction and patient simulation based on systems biology have been developed (Kolpakov *et al.*, 2019; Subbalakshmi *et al.*, 2021). One example of these models is the Therapeutic Performance Mapping System (TPMS) (Jorba *et al.*, 2020). This kind of model gathers all the information on the human found in accessible databases containing different information types (physiological, metabolic, clinical studies, etc.). All this information is used as constraints in the in-silico models, which makes the modelling of specific processes possible, such as metabolism, mechanism of action of drugs and variations at the protein expression level caused by patients' clinical conditions, including main pathological conditions such as cancer and minor adverse effects. This approach has been employed to study several pathologies and their drug development (Moncunill *et al.*, 2020; Segú-Vergés *et al.*, 2021), including several cancer types (Morales *et al.*, 2017; Giménez *et al.*, 2018).
The main objective of this study is to perform computational simulations to predict the level of eosinophils, red blood cells, hemoglobin, leukocytes, monocytes, and platelets in metastatic pancreatic cancer patients and evaluate them as clinical decision support system. Model generation is done using supervised machine learning techniques on real patients' data that underwent first-line palliative chemotherapy under the regime Gemcitabine/Nab-Paclitaxel, enrolled in the SICPAC study (code MOR-GEM-2018-01). Other goals of the study were the identification of characteristics of patients, treatments and time intervals between cycles that might influence the measurement of the blood cells of interest. Moreover, the implementation of systems biology will shed light on proteins and pathways that might have a key role in the variation of the variables measured in this study.

3.3 Materials and methods

SICPAC study data
Data to be analysed by systems biology and Machine Learning techniques to perform computational simulations on pancreatic cancer patients was obtained via an observational study with authorization prior to recruitment with a retrospective monitoring. The study, SICPAC, was carried out in Hospital General La Mancha Centro, Alcázar de San Juan, Ciudad Real, Spain (code MOR-GEM-2018-01) under the supervision of Doctor Rafael Morales and fulfilled all the requirements needed: approval of research of drugs by

the Comité de Ética de la Investigación and presentation of the study to the Agencia Española de Medicamentos y Productos Sanitarios (AEMPS).

Study population
Data from 20 patients diagnosed with locally advanced or metastatic adenomatous pancreatic cancer from 2015 to 2018 were recruited. These patients had previously undergone palliative first-line chemotherapy with Gemcitabine/Nab-paclitaxel and were over 18 years old at the time of recruitment. Clinical information of at least 3 visits (1 treatment cycle) of each patient was also required.

Data retrieval
Data collected from the Case Report Form (CRF) of the SICPAC study contains different features measured for each patient. They can be mainly grouped by demographic data, previous personal records, symptoms of disease, pharmacologic treatment, physical exploration, additional explorations, and tumor biology. A list of the CRF variables is given in Appendix C.1.

Simulation of patients by Systems Biology
Data collected from the SICPAC study were employed to model each of the 20 patients recruited using System Biology approaches. In this study the TPMS method (Jorba *et al.*, 2020; Gutiérrez-Casares *et al.*, 2021) was used, which is based on a map of all known human proteins with links to metabolic pathways, signalling pathways and annotation of proteins pairs with identified physical interaction, as seen in Figure 3.1. The role of most human proteins in human physiology is unknown. Some proteins, however, have a strong relation with a certain phenotype, such as the Na/K pump, which is highly related to diarrhea. The Biological Effectors Database (BED) (Jorba *et al.*, 2020) is a database containing 305 different phenotypes, which are grouped by the biological motives that induce them, i.e., diabetes can be caused by beta cell destruction, problems in insulin production, etc. Each motive is described by a protein set having a strong relation to it, proved in the literature. Furthermore, several databases that describe proteins targeted by drugs were used (Wishart *et al.*, 2017).
The models built by TPMS must include all the patients' clinical states. Every effect caused by the administration of drugs, either through the target proteins or through off-targets causing adverse effects, must be considered in these models. In TPMS, this would be represented as the activation and

inhibition of the target proteins of the drugs along with the protein expression state of the proteins closely related to the disease, known as protein effectors. Models built in TPMS consider all the information on the human condition at several levels in order to run the simulations. This data has been collected and compiled using information from several sources, such as demographic information on patients (height, weight, age, etc.) amount of drug administered and the protein-protein interactions defined from various databases like KEGG (Kanehisa, 2000, 2019; Kanehisa *et al.*, 2020), REACTOME (Jassal *et al.*, 2019) and BioGRID (Oughtred *et al.*, 2020). Extra information is obtained from the CRF records, where additional drug treatments (co-treatments) and adverse effects were described.
Patients analysed in this study suffered from pancreatic cancer. In the TPMS models, a main treatment with Gemcitabine/Nab-Paclitaxel along with cotreatments were defined for all patients, based on the CRF data. Target proteins were identified for each different drug using information from DrugBank (Wishart *et al.*, 2017) and added into the TPMS models. Several phenotypic traits defined as adverse effects were also identified and incorporated into the TPMS models with their protein effectors extracted from BED. A different TPMS model was built per patient and visit. For each visit, over 250 solutions were calculated in each of these models. These solutions have a high accuracy and are closely related to the clinical information of each patient and visit described in the CRF. Thus, starting from the data collected in the CRF and using the TPMS models we obtain a set of proteins and biologic pathways that are activated.

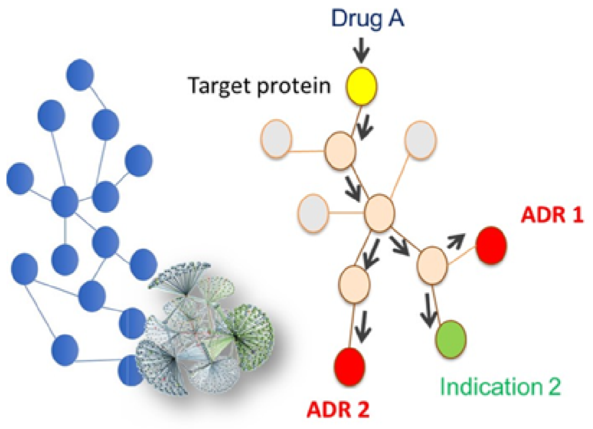


Figure 3.1: Outline of a human protein network. On the left, proteins are represented by the blue nodes and linked between them by their signalling, interaction, or metabolic relations (edges). A network of high complexity can be seen only by selecting a few proteins as shown. On the right, Drug A activates its target protein, and the activation or inhibition signals distribute across the network until they reach the protein effectors that cause indication (green) or adverse effects (red).

Analysis of SICPAC study using Machine Learning techniques

Time series analysis was performed using features identified on the CRF as well as those obtained with TPMS as the main Machine Learning method. Multivariate models based on regression trees were generated to predict the next visit's values of the 6 clinical outcomes selected (concentration of eosinophils, platelets, red blood cells, leukocytes, monocytes and hemoglobin). All the consecutive patients' visits noted in the CRF were used to train the models, where two visits are considered as successive if the time period between them is less than 50 days. The model consists of an ensemble of up to 5 regression trees selecting up to 3 variables (4 including the time period between visits) from up to 3 consecutive visits. For each of the 6 clinical outcomes, a model was trained, and its performance was evaluated in a Leave One Patient Out (LOPO) cross-validation set up. In this set up, all the visits' data from all but one patient were used in the training and all the visits from the left-out patient were used for validation. This training-validation splitting process was repeated for all patients and the validation metric used was the Spearman correlation between the experimental values from the CRF and the predicted values. To avoid overfitting, the selection of the variables was performed within the LOPO set up. Categorical variables are labelled as

1 and 0, depending on whether the patient was under the category effects or not, respectively. In addition, continuous variables (including the outcomes) were normalized by subtracting from their values the median of all their values up to the last available visit (for each patient separately). The number of days between consecutive visits was also used as a predictive variable and was normalized as such. A representation of the variables' nomenclature is shown in Table 3.1.

Nomenclature of the variables

Visit 3	**Visit 2**	**Visit 1**	**Upcoming visit**
Nausea_3	Nausea_2	Nausea_1	
	Time_3	Time_2	Time_1

Table 3.1: Variables are created by indicating the clinical term followed by the visit number (Nausea_n in this table). Nausea_n corresponds to the variable Nausea, n visits before the upcoming visit and similarly for all other variables except for Time. Time_n corresponds to the number of days between the n^th visit and the $(n-1)^th$ visit before the upcoming visit. This is illustrated in the table by positioning the Time variable between two columns.

Following the nomenclature of Table 3.1, the normalized visit values for a continuous variable Var after n visits are:

$$\begin{gathered}Var\_1 - \text{median}\{Var\_1, \ldots, Var\_n\}\\ Var\_2 - \text{median}\{Var\_1, \ldots, Var\_n\}\\ Var\_3 - \text{median}\{Var\_1, \ldots, Var\_n\}\end{gathered}$$

In this study, count values of eosinophils, red blood cells, leukocytes, monocytes and platelets together with concentration values of hemoglobin to be obtained in future visits were predicted based on the values of features identified on the CRF in earlier visits. Spearman's correlation with the LOPO set up and a prediction score followed by the same cross-validation method were calculated to evaluate the ability to predict new concentration values in any upcoming clinical visit as well as detecting concentration trends in the patients. This was done using regression trees, a Machine Learning method that has been previously used for predicting either categories or continuous values based on training data. Regression trees have been proven to be effective methods to handle structured data sets. In this study, variables extracted from the TPMS models were added to the clinical data, yielding a high dimensional training dataset. Variables used in TPMS models are mainly proteins, so a remarkable difference would be expected using these variables with the clinical data. In consequence, two separate analyses, one with CRF data and the other with CRF and TPMS variables, were performed using regression trees.

Creation of a webserver to facilitate the usage of ML models
Given their complexity and the skill needed to run and analyse the regression trees presented in this work, a graphic user-friendly interface was built and deployed as a webapp, accessible at `http://sicpac.anaxomics.com:81`, as means of facilitating its usage at a scientific and clinical setting. The tool was built in a python-based environment using the Django back-end (`https://www.djangoproject.com/`). Considering that all the regression trees were built on MATLAB, we used the MATLAB Compiler SDK toolbox as interface between Python and the original programming language.

3.4 Results

3.4.1 Population study

A total of 274 clinical visits were gathered for the 20 patients recruited in this study, where 138 variables (consisting of analytical measures, presence of AES and intake of co-treatments, see Appendix C.1) were recorded per visit for every patient. All patients were over 18 years old, with a mean value of 64.4 years for the whole population (range between 47 and 82, with the 40% and 60% being women and men, respectively). In Table 3.2, the principal variables of the study (concentration values of eosinophils, red blood cells, hemoglobin, leukocytes, monocytes, and platelets) along with their mean, median and standard deviation are shown. Moreover, an average of 2.95 adverse effects and an average intake of 5.7 complementary treatments were found per visit.

Outcomes values

	All			**Female**			**Male**		
	Mean	Std	Median	Mean	Std	Median	Mean	Std	Median
Leukocyte [$10^3/\mu L$]	6.61	3.97	5.8	7.26	4.72	5.95	5.73	2.35	5.7
Monocyte [$10^3/\mu L$]	0.64	0.5	0.5	0.72	0.57	0.5	0.54	0.37	0.5
Hemoglobin [g/dL]	11.37	1.43	11.2	11.07	1.39	11	11.77	1.4	11.65
Red blood cell [$10^6/\mu L$]	3.85	1.98	3.76	3.79	2.58	3.62	3.94	0.43	3.92
Eosinophil [$10^3/\mu L$]	0.2	0.31	0.1	0.24	0.38	0.1	0.15	0.17	0.1
Platelet [$10^3/\mu L$]	227.75	142.46	185	234.39	155.25	174	218.63	122.82	192
Number AES per visit	2.95	2.18	2	3.04	2.19	3	2.83	2.17	2
Number co-treatment per visit	5.68	4.88	5	5.69	5.07	4	5.65	4.64	5

Table 3.2: Values of eosinophils, red blood cells, hemoglobin, leukocytes, monocytes, and platelets, together with the number of adverse effects and complementary treatments per visit. Mean, median and standard deviation values are computed for the male, female, and the whole population.

3.4.2 Study of correlation between CRF and variables from Systems Biology

Regression trees were built from both variables described in the CRF and variables generated by the TPMS to predict each of the 6 principal outcomes analysed in this study. Table 3.3 contains the Spearman correlation between the desired and predicted values in the LOPO set up for the models trained with the CRF variables (coulmn CRF) and the models trained with the CRF and TPMS variables (column CRF+TPMS). Spearman correlation values in Table 3.3 show that the variables obtained from the TPMS models did not have a great impact in predicting the concentration values of the 6 variables since the correlation values obtained with the variables from the CRF alone or in combination with those from TPMS did not vary greatly. For each of the 6 outcomes, Figure 3.2 shows the desired values vs the predicted values of the model trained with the CRF variables (in the LOPO set up). It is interesting to note that different results were obtained from analysing patients' data at the same time or separately. Known as the Simpson's Paradox (Hernán *et al.*, 2011), weaker correlation values were calculated by averaging the correlation values obtained for each patient separately for the 6 variables (3.4). As shown in the table, this was seen for both sets of variables, CRF and CRF + TPMS. This implies that any prediction done for a specific patient should account for this correction, thus making these models not eligible for predicting expected concentration values in the following visits.

		CRF	**CRF+TPMS**
Spearman correlation	Leukocyte	0.71	0.71
	Monocyte	0.64	0.60
	Hemoglobin	0.76	0.77
	Red blood cell	0.76	0.78
	Eosinophil	0.72	0.71
	Platelet	0.68	0.67

Table 3.3: LOPO set up's Spearman correlation between the concentration values of the 6 main variables when the models were trained with the CRF variables alone and in combination with the TPMS ones.

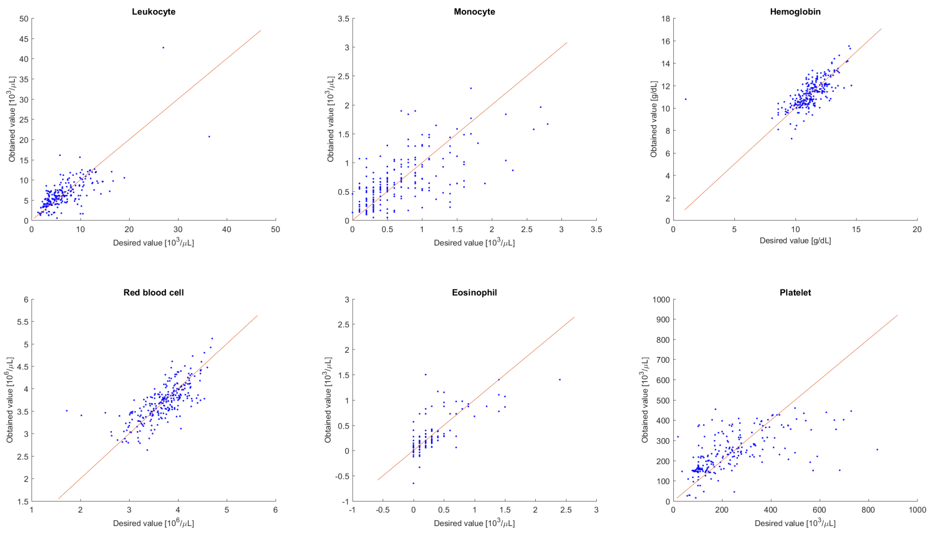


Figure 3.2: Plot of desired vs predicted values in the LOPO set up with the CRF variables. Each plot corresponds to one of the 6 outcomes. The x-axis contains the desired values while the y-axis contains the predicted values of the model trained with the CRF variables in the LOPO set up. The red line corresponds to the identity, i.e., where the points would ideally lie.

		CRF	CRF+TPMS
Spearman correlation	Leukocyte	0.37	0.36
	Monocyte	0.47	0.44
	Hemoglobin	0.45	0.51
	Red blood cell	0.45	0.52
	Eosinophil	0.39	0.30
	Platelet	0.40	0.35

Table 3.4: LOPO set up's Spearman correlation values, considering the Simpson's Paradox, i.e. the correlation is computed per patient and the average among patient is given in the table.

3.4.3 Prediction score of increasing or decreasing trends compared to the previous visit

As seen in Figure 3.2 in the scatter plots, a correlation between experimental and predicted concentration values could be found. Consequently, prediction scores, expressed as balanced accuracies, were calculated to measure the ability to predict increase vs decrease of the outcomes compared to the previous visit's values, based on the visits annotated in the CRF. Note that the visits for which there is no increase or decrease compared to the previous visits were ignored in the computation of this score. The prediction scores reflect the ability of the model, in the same LOPO set up previously described, to classify an increase vs a decrease of an outcome's value compared to its value during the last known visit. As seen in Table 3.5, except for the hemoglobin concentration and the red blood cell count, the increases or decreases of the outcomes in the following visits can be predicted with a balanced accuracy of around 0.77.

3.4.4 Variables selected in the models

The variables selected in the models can be considered as the factors that had the greatest influence on the increase or decrease with respect to the median value of a patient's outcome. These factors are shown in Table 3.6, with both the variables from the CRF and CRF + TPMS models. The variables shown in this table highlight which clinical features affect the most the increase or decrease with respect to the median of the concentration, where variables obtained from TPMS models give a deeper insight on the molecular pro-

Increase/decrease prediction score

	Prediction score (balanced accuracy)	
	CRF	**CRF+TPMS**
Leukocyte	0.770	0.757
Monocyte	0.774	0.768
Hemoglobin	0.609	0.685
Red blood cell	0.608	0.666
Eosinophil	0.843	0.865
Platelet	0.773	0.755

Table 3.5: Prediction scores (balanced accuracy) to measure the tendency of the 6 main outcomes to increase or decrease compared to the previous visit's outcome values. Both models, with variables from the CRF and with variables from the CRF and TPMS, are included.

cesses that underlie those trends. Regression trees built to get these results are available in Appendix C.3, with the CRF (Appendix C.3.1) and CRF + TPMS models (Appendix C.3.2). Appendix C.2 contains an explanation on how to read the regression trees. Results for each 6 principal variables of the study are described below. Information on genes was retrieved from UniProt (`https://www.uniprot.org/`).

Leukocytes

In regression tree models only built using CRF data, increase and decrease of leukocyte concentration with respect to the patient's median value was observed mainly due to the following factors. Diagnosis of neutropenia, which is defined by the low concentration of neutrophils in blood, in the last clinical visit recorded in the CRF is found to be linked. Moreover, administration of fentanyl, an opioid analgesic, and naloxone, an opioid receptor antagonist, in the last two visits noted in the CRF was also related. Finally, diagnosis of neuropathy in the penultimate visit was also found to influence leukocyte concentration. Adding the information extracted from Systems Biology, regression trees also predicted the effect of diagnosis of neutropenia, along with the overexpression and underexpression of the proteins shown in Table 3.7.

Monocytes

In the models using only CRF data, as observed in the corresponding plots

Selected variables

	CRF	**CRF+TPMS**
Leukocyte	DB00813 (Fentanyl), DB01183 (Naloxone), NEUROPATHY, NEUTROPENIA, Time	NEUTROPENIA, P11021 (HSPA5), P32246 (CCR1), Q13464 (ROCK1), Q5VWK5 (IL23R), Q9UHD2 (TBK1), Time
Monocyte	Creatinine, Metastasis, NEUTROPENIA, Sodium, Time	NEUTROPENIA, O75881 (CYP7B1), Time
Hemoglobin	Hemoglobin, Time	Q6ZNA4 (RNF111), Q99466 (NOTCH4), Q9Y210 (TRPC6)
Red blood cell	Red blood cell	O75460 (ERN1), Q9NZJ5 (EIF2AK3)
Eosinophil	NEUTROPENIA, Time	NEUTROPENIA, Time
Platelet	ARTHRITIS, DIARRHEA, Lymphocyte, MUCOSITIS, NEUTROPENIA, Time	Q07864 (POLE), Q15054 (POLD3), Q9HCU8 (POLD4), Q9NR33 (POLE4), Q9NRF9 (POLE3), Time

Table 3.6: List of variables obtained from the regression models to predict the increasing or decreasing trends with respect to the median value of the patient's outcome, from the CRF and CRF + TPMS models.

Gene name	Gene ID	UniProt ID	Time instance	Cellular function
Rho-associated protein kinase 1	ROCK1	Q13464	Last two clinical visits (ROCK_2, ROCK_1)	Regulation of smooth muscle contraction, actin cytoskeleton organization, stress fiber and focal adhesion, motility, etc.
Endoplasmic reticulum chaperone BiP	HSPA5	P11021	Penultimate visit (HSPA5_2)	Correct folding and degradation of misfolded proteins
Serine/threonine-protein kinase	TBK1	Q9UHD2	Last visit (TBK_1)	Cell death programming
Interleukin-23 receptor	IL23R	Q5VWK5	Penultimate visit (IL23R_2)	Stimulation of immune cells through Jak-Stat signaling cascade
C-C chemokine receptor type 1	CCR1	P32246	Penultimate visit (CCR1_2)	Stem cell proliferation

Table 3.7: Genes identified for the prediction of Leukocyte concentration.

from Appendix C.3.1, diagnosis of neutropenia in patients was also considered as having a significant effect. Time periods between any upcoming visit and the last visit, and between the penultimate and the last visit affected the concentration of monocytes in patients. In addition, concentration differences of creatinine and sodium in the penultimate and last visit were labelled as relevant. Creatinine concentration variation might indicate renal failure since it might cause creatinine accumulation in blood. Lastly, detection of metastasis in the penultimate visit was also found to be relevant for monocyte concentration. Merging the information given by Systems Biology methods, diagnosis of neutropenia and time periods between visits were still relevant in the regression trees. Moreover, over and underexpression of cytochrome P450 7B1 (Gene ID: CYP7B1, UniProt ID: O75881) was found to have a relevant effect. This protein is involved in the metabolism of endogenous oxysterols and steroid hormones, including neurosteroids. It also regulates B-cell migration in germinal centers of lymphoid organs, thus guiding effective maturation of plasma B-cells and overall antigen-specific humoral immune response.

Hemoglobin
Use of clinical data from the CRF revealed that the factors that affect the most hemoglobin concentration in any upcoming visit are hemoglobin concentration variation in the last visit as well as the time period between the last and the upcoming visit, as seen in the corresponding plots from Appendix C.3.1. Using CRF data and information given by TPMS models, no other variable recorded in the CRF than hemoglobin itself was labelled as significant in the regression trees, while finding expression variation significant for the following proteins in Table 3.8.

Red blood cells
No relevant factors that might have an influence on red blood cells in future visits were found, as observed in the corresponding plots from Appendix C.3.1. Adding the data extracted from Systems Biology methods, two proteins were found to influence red blood cell concentration in visits to come. The serine/threonine-protein kinase/endoribonuclease IRE1 (Gene ID: ERN1, UniProt ID: O76460) is a protein found in the endoplasmic reticulum that acts as a key sensor for the endoplasmic reticulum unfolded protein response (UPR) (Yoshida *et al.*, 2001; Iwawaki *et al.*, 2001; Liu *et al.*, 2003, 2017). A high expression of this protein in pancreatic cells has been reported

Gene name	**Gene ID**	**UniProt ID**	**Time instance**	**Cellular function**
Short transient receptor potential channel 6	TRPC6	Q9Y210	Last visit (TRPC6_1)	Form a receptor-activated non-selective calcium permeant cation channel
E3 ubiquitin-protein ligase Arkadia	RNF111	Q6ZNA4	Last visit (RNF111_1)	Acts as an enhancer of the transcriptional responses of the SMAD2/SMAD3 effectors
Neurogenic locus notch homolog protein 4	NOTCH4	Q99466	Last visit (NOTCH4_1)	Functions as a receptor for membrane-bound ligands Jagged1, Jagged2 and Delta1 to regulate cell-fate determination

Table 3.8: Genes identified for the prediction of Hemoglobin concentration.

(Tirasophon *et al.*, 1998). In addition, expression variation obtained in the last visit of eukaryotic translation initiation factor 2-alpha kinase 3 (Gene ID EIF2AK3, UniProt ID: Q9NZJ5) was found to be also relevant. This protein is a key activator of the integrated stress response needed for adaptation to several stresses, like UPR and low amino acid availability (Sood *et al.*, 2000). Both proteins found are closely linked to oxidative stress and high expression in pancreas. Therefore, concentration differences of red blood cells might be due to pancreatic cancer rather than treatment effects in the patients studied.

Eosinophils
As for the eosinophils, two factors were found to be related to their concentration variations based on the information collected at the CRF: diagnosis of neutropenia in the last two clinical visits and time periods between the last and upcoming visit and between the first and second clinical visit. No extra information was found adding data form the TPMS models.

Platelets
Several factors that might affect platelet concentration extracted from only CRF data were determined. All time periods possible were found to be relevant (last to upcoming visit, penultimate to last visit and ante-penultimate to penultimate visit). Diagnosis of neutropenia in the last and penultimate visit recorded was also labelled as important by the regression trees. Moreover, detection of mucositis, an inflammation of the mucous membranes across the internal intestine tract, was also significant. Merging data obtained from TPMS models into the CRF data, several additional factors were found to be important in platelet concentration prediction. Time period between the last recorded and future visit together with the time period between the ante-penultimate and penultimate visit recorded were categorized as relevant. Furthermore, several proteins were found to influence platelet concentration based on their expression variation. These proteins are listed in Table 3.9.

3.4.5 Clinical tool for aiding in clinical decision-making

The web-application allows users to predict levels of leukocytes, monocytes, eosinophiles, red blood cells, hemoglobin and platelets given the clinical variables displayed by the patient in recent visits to the clinician's office, thus providing physiological insight into patient's response to the clinical strategy in study. Figure 3.3 provides a general overview of this tool. After logging in,

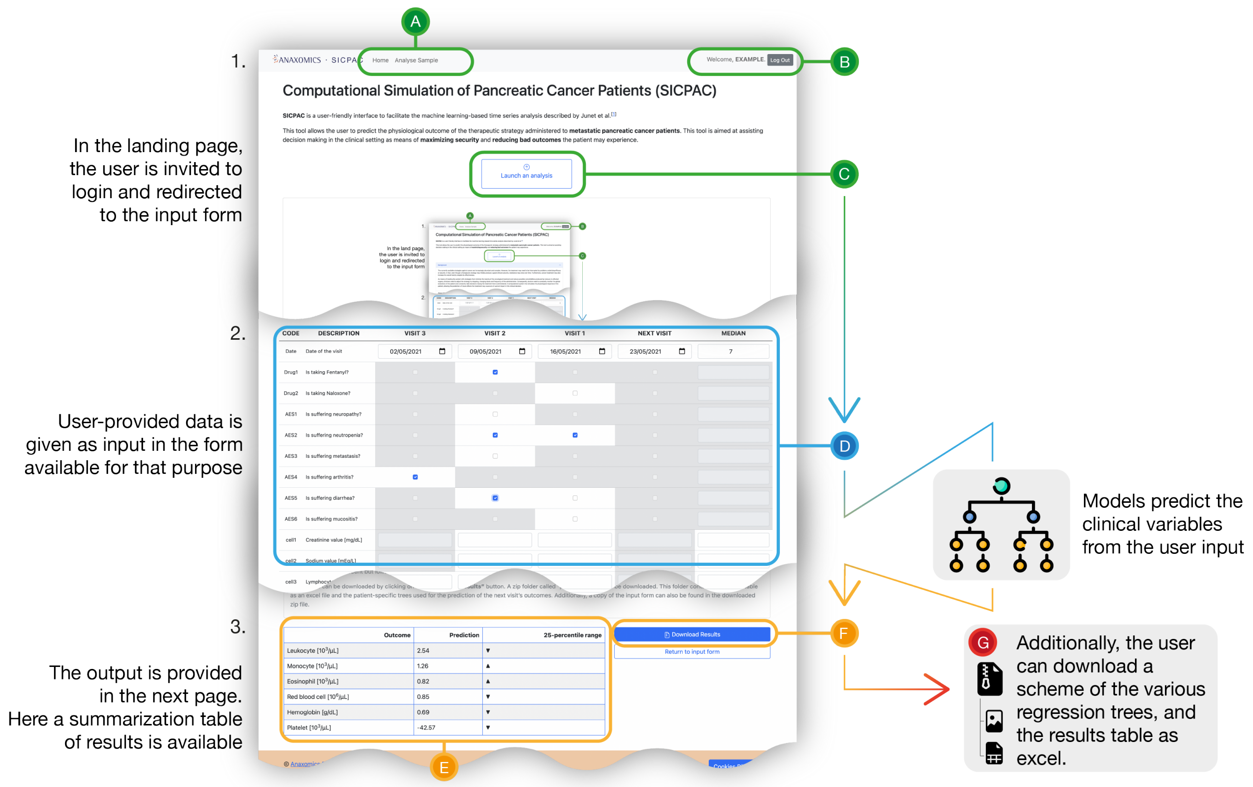
Gene name	**Gene ID**	**UniProt ID**	**Time instance**	**Cellular function**
DNA polymerase delta subunit 3	POLD3	Q15054	Last visit (POLD3_1), Ante-penultimate visit (POLD3_3)	Genome replication and polymerase-delta stabilization and stimulation by PCNA
DNA polymerase epsilon subunit 3	POLE3	Q9NRF9	Last visit (POLE3_1) Ante-penultimate visit (POLE3_3)	DNA repair and replication
DNA polymerase epsilon catalytic subunit 3	POLE	Q07864	Last visit (POLE_1) Ante-penultimate visit (POLE_3)	DNA replication and DNA synthesis during DNA repair
DNA polymerase delta subunit 4	POLD4	Q9HCU8	Last visit (POLD4_1) Ante-penultimate visit (POLD4_3)	DNA replication and repair. Increases rate of DNA synthesis and decreases fidelity
DNA polymerase epsilon subunit 4	POLE4	Q9NR33	Last visit (POLE4_1) Ante-penultimate visit (POLE4_3)	DNA chromosomal replication and repair

Table 3.9: Genes identified for the prediction of Platelet concentration.

the user can start a new analysis. This procedure is done through the provided input form (Figure 3.3D). Here, the user must describe the clinical case by providing dates of the visits; administered medication (if the patient was being prescribed Fentanyl or Naloxone in the different visits); symptomology (if the patient was experiencing neuropathy, neutropenia, metastasis, arthritis, diarrhea, or mucositis); and the values of different circulating biomarkers (levels of creatinine, sodium, lymphocytes, red blood cells, hemoglobin, leukocytes, monocytes, eosinophils and platelets). Submitting the form will run the regression tree models and redirect the user to the results page (step 3 on Figure 3.3). Here lies a summary table (Figure 3.3E) containing the predicted value of the next visit for the six different outcomes: leukocyte, monocyte, eosinophile, red blood cell, hemoglobin and platelet levels. Additionally, an extra column (25-percentile range) provides information on where the predicted values lie relative to the values in the training data set. Namely, if the inserted case lies in the top or bottom 25 percent of the corresponding values in the training data. Finally, the results can be downloaded by clicking on the Download Results button (Figure 3.3F). By this process, a zip file containing the output table and the patient-specific trees used for the prediction of the next visit's outcomes will be provided.

3.5 Conclusion

In this study, the use of mathematics models via TPMS has turned out to be effective to support clinical decisions based on data regularly collected in record in the CRF. Despite the low number of samples, some remarkable results have been obtained.
One of the main conclusions that can be drawn from this study is the difficulty in predicting future count of eosinophils, red blood cells, leukocytes, monocytes and platelets and concentration of hemoglobin using data extracted from previous clinical visits. This might be due to the limited number of samples available (only 20 patients were recruited and monitored in this study). However, having a larger number of samples or a longer monitoring of the patients might improve the models, thus making the prediction possible.
Despite this, an analysis of the increasing and decreasing trends of the count of eosinophils, red blood cells, leukocytes, monocytes and platelets and concentration of hemoglobin was performed using regression trees. These Ma-


1.
In the landing page, the user is invited to login and redirected to the input form
2.
User-provided data is given as input in the form available for that purpose
3.
The output is provided in the next page. Here a summarization table of results is available
A
B
C
D
E
F
G
Models predict the clinical variables from the user input
Additionally, the user can download a scheme of the various regression trees, and the results table as excel.
ANAXOMICS · SICPAC
Home
Analyse Sample
Welcome, EXAMPLE.
Log Out
Launch an analysis
Download Results
Return to input form

Computational Simulation of Pancreatic Cancer Patients (SICPAC)

SICPAC is a user-friendly interface to facilitate the machine learning-based time series analysis described by Junet et al.[1]

This tool allows the user to predict the physiological outcome of the therapeutic strategy administered to **metastatic pancreatic cancer patients.** This tool is aimed at assisting decision making in the clinical setting as means of **maximizing security** and **reducing bad outcomes** the patient may experience.

CODE	DESCRIPTION	VISIT 3	VISIT 2	VISIT 1	NEXT VISIT	MEDIAN
Date	Date of the visit	02/05/2021	09/05/2021	16/05/2021	23/05/2021	7
Drug1	Is taking Fentanyl?	☐	☑	☐	☐	
Drug2	Is taking Naloxone?	☐	☐	☐	☐	
AES1	Is suffering neuropathy?	☐	☐	☐	☐	
AES2	Is suffering neutropenia?	☐	☑	☑	☐	
AES3	Is suffering metastasis?	☐	☐	☐	☐	
AES4	Is suffering arthritis?	☑	☐	☐	☐	
AES5	Is suffering diarrhea?	☐	☑	☐	☐	
AES6	Is suffering mucositis?	☐	☐	☐	☐	
cell1	Creatinine value [mg/dL]					
cell2	Sodium value [mEq/L]					
cell3	Lymphocyt					

can be downloaded by clicking on ults" button. A zip folder called be downloaded. This folder con able as an excel file and the patient-specific trees used for the prediction of the next visit's outcomes. Additionally, a copy of the input form can also be found in the downloaded zip file.

Outcome	Prediction	25-percentile range
Leukocyte [10^3/µL]	2.54	▼
Monocyte [10^3/µL]	1.26	▲
Eosinophil [10^3/µL]	0.82	▲
Red blood cell [10^6/µL]	0.85	▼
Hemoglobin [g/dL]	0.69	▼
Platelet [10^3/µL]	-42.57	▼

© Anaxomics

Figure 3.3: Overall scheme of the SICPAC user interface and working mechanics. The letters A to G point towards the various parts with which the user may interact with the webserver: (A) the main menu, (B) the login and logout buttons, (C) contextual button that launches a new analysis or login, (D) the input form in which the user can describe the previous clinical condition of the patient, (E) a summary table containing the predictions of the levels of leukocytes, monocytes, eosinophiles, red blood cells, hemoglobin and platelets, (F and G) a button to allow the download of a zip file containing schematic representations of the regression trees that were created and the overall results in a excel spreadsheet. The numbers 1 to 3 refer to the major pages through which the user will be carried: (1) Landing page, (2) Input form, and (3) Results page.

chine Learning models cannot be applied to other cancer patients at first since only pancreatic cancer patients which had undergone palliative first-line chemotherapy with Gemcitabine/Nab-paclitaxel were used in this study, but considering the type of data used, relevant results could be obtained using these models on other cancer patients.
Several factors were found to influence the studied clinical outcomes. One of the factors that affect significantly the clinical results is the time period between visits. This suggests that the time period between two consecutive visits might influence the clinical features analysed here, potentially driving the oncologist to incorrect conclusions. Cancer patients usually go through a high number of clinical visits in short time periods. This could cause a lack of stability in patients' response to the drugs administered and might have a high significance for eosinophils and hemoglobin, where only diagnosis of neutropenia and time periods between visits are found to have an influence.
Apart from time periods between visits, various proteins were identified and labelled as significantly linked to increasing and decreasing trends. Most of the proteins found in this study were related to cell proliferation processes, like cytoskeleton organization and cell death programming, and DNA replication. Such information could give the clinician a deeper insight on the molecular processes that underlie disease progression in the patients analysed.
Finally, a user-friendly graphic interface was built and presented as a website, as an implementation of the prediction tools built in this study. The use of the regression trees is therefore accessible to any researcher interested. Results are displayed in a very intuitive way, as shown in Figure 3.3. In addition, results can be exported into Excel files, which could be further processed in downstream analyses.

Discussion and outlook

Among all the considerations to take when organizing a pipeline for the analysis of biomedical data, the first one should be about the data themselves. We saw an example of how relevant it is to properly prepare a data set in Chapter 1. We presented a novel algorithm, CuBlock, for the normalization of gene-expression microarray data originating from different experiments and available on public repositories. We showed that after normalization, biological patterns could be found across the different experiments, therefore enabling the integration of multiple data sets for subsequent analysis. If one such method is not applied, the main effect to observe across the data would be the difference between batches. Considering the steadily increasing amount of sequencing data deposited in public repositories, it would be interesting to test whether an approach like CuBlock could be also relevant for the integration of RNA-seq data.
With all the available databases, the first chapter constitutes a practical example of some considerations to have when starting the analysis of retrieved data sets. Of course, there are many others such as the cleaning of data, dealing with outliers or missing data and engineering relevant features. To extract meaningful information from the data, appropriate care must be taken.
In order to build and test an open-source tool for the binding prediction of MHC-class II peptides in Chapter 2, the data set retrieved from IEDB was thus carefully constructed. Indeed, these data come from similar experimental designs and contain some level of redundancy. For this reason, we identified all the unique 11-mers within a given peptide set and only selected the shortest peptide containing each 11-mer. Moreover, the experimental results mostly report positive binders. Consequently, few non-binders could be found in comparison. To balance the sets, a random selection was performed. The negative instances were randomly selected among known human proteins so that the length distribution of the positive set was respected.

Appropriate data processing is not only important to extract meaningful information but also to properly validate a model. Based on previous knowledge about MHC-class II peptides, the method was adapted to highlight binding nonamers contained within a peptide (the tool also accepts other length than 9) and therefore the partition for the cross-validation was generated such that a minimal amount of nonamers are shared across folds. Moreover, for an unbiased independent evaluation, the instances from the training data that were also contained in the evaluation sets were removed. While some of the considerations for data processing described above are rather general and apply in most situations, some others are specific to the data sets and can therefore not be part of a fully automated workflow.

Specific considerations are not only relevant for data preparation but also for the development of a method. CNN-PepPred was designed for application on peptide sets and makes use of a simple type of convolutional neural network architecture. This architecture is particularly adapted to the difference in length among peptides and to highlight sub-strings of a given length within a peptide while ignoring non-relevant information. CNN-PepPred was also designed as an open-source tool with a detailed User's guide so that it can be used and modified by the community. The possibility of modifying CNN-PepPred is relevant since it was intended to be an allele-specific core prediction method with the option of generalization in other contexts. For example, in the latest version of CNN-PepPred (Version 0.1.1), the training was extended to give the possibility to train on new data sets including the knowledge extracted from previously trained models which is known as training with transfer learning. It would be interesting to investigate how to further extend CNN-PepPred to include training on multiple-allele data sets following the new research direction in this field.

The data set from the SICPAC study in Chapter 3 required a different kind of attention. As clinical data, there was missing information, there were few samples, mixing categorical variables with continuous ones and the model had to be -to some level- easily and directly interpretable. For this reasons, decision trees were chosen as the core of the method. To improve performance, a model was made from an ensemble of trees and, to render it interpretable and avoid overfitting, a low number of variables was used per individual tree. Care also had to be taken when interpreting the results; considering all patients together, a correlation between true and predicted outcome of around 0.7 could be observed while only an average correlation of around 0.4 could be reached when considering the patients separately. It is therefore important to understand the nuance of a data set for the design and interpretation of a model. For further developments of the strategy presented in this study, it would be relevant to test it on a larger amount of data and to see whether it can generalize to other cancer types.

During this thesis, we have addressed, in an application-oriented way, a number of objectives all revolving around careful attention to the data. A particular focus was given to the preparation of data sets, the features to use as predictors and the methods chosen. ML algorithms were consequently applied and validated to highlight patterns contained within the data analysed. This thesis does not aim at giving a recipe to find the best pipeline for the analysis of biomedical data, instead we followed specific processes towards a solution to given problems. Through each of the different works in the main body, various non-exhaustive considerations arising from particular data sets were highlighted and we considered ways of approaching such challenges with ML techniques in bioinformatics. Interesting developments for application-oriented ML strategies will likely arise from AutoML approaches. Finding a way to automatize decision making while considering the specificities of a given data set will be critic to this development. It will additionally help in finding a consensus among methods. Instead of developing new tools with new or modified methods which are potentially flawed and require extensive validation, the focus will rather be on the automation of the decision making process while using standard and well described methods at its core. Combining knowledge from the analysis of specific data sets with automated processes might play an important role in the future of ML techniques in bioinformatics.

Conclusion

In this thesis, we presented three independent studies each highlighting different aspects to consider in a pipeline for the analysis of biomedical data.
Firstly, we could conclude the importance of a normalization method for the integration of gene-expression microarray data originating from different platforms. The novel algorithm developed was efficient at highlighting biological patterns contained in data from various sources. It should be noted that some batch effect remained.
Secondly, we were able to accurately predict the binding of MHC-class II peptides and extract knowledge from the trained models. The tool was made available for the community with an open-source license and a detailed User's guide.
Thirdly, promising results where obtained in the application of a comprehensible time series analysis in the context of a retrospective study in pancreatic cancer. Having as objective the prediction of a patient's outcome at a future hospital visit, the approach should be used with care to assist decision making since the results didn't justify using the models as stand-alone tools.
Lastly, we discussed the relevance of a data centric approach, underlying the specific care required by some data sets.

Bibliography

Alvarez, B., Reynisson, B., Barra, C., Buus, S., Ternette, N., Connelley, T., Andreatta, M., and Nielsen, M. (2019). NNAlign_MA; MHC Peptidome Deconvolution for Accurate MHC Binding Motif Characterization and Improved T-cell Epitope Predictions. *Mol. Cell. Proteom.*, **18**(12), 2459–2477.

Andreatta, M., Karosiene, E., Rasmussen, M., Stryhn, A., Buus, S., and Nielsen, M. (2015). Accurate pan-specific prediction of peptide-MHC class II binding affinity with improved binding core identification. *Immunogenetics*, **67**(11), 641–650.

Baldi, P. and Brunak, S. (1998). *Bioinformatics: The machine learning approach*. MIT Press.

Bartlett, V. L., Dhruva, S. S., Shah, N. D., Ryan, P., and Ross, J. S. (2019). Feasibility of Using Real-World Data to Replicate Clinical Trial Evidence. *JAMA Netw Open*, **2**(10), e1912869–e1912869.

Bateman, A. (2007). EDITORIAL. *Nucleic Acids Research*, **35**(suppl_1), D1–D2.

Benito, M., Parker, J., Du, Q., Wu, J., Xiang, D., Perou, C. M., and Marron, J. S. (2004). Adjustment of systematic microarray data biases. *Bioinformatics*, **20**(1), 105–114.

Berger, M. L., Sox, H., Willke, R. J., Brixner, D. L., Eichler, H.-G., Goettsch, W., Madigan, D., Makady, A., Schneeweiss, S., Tarricone, R., Wang, S. V., Watkins, J., and Mullins, C. D. (2017). Good practices for real-world data studies of treatment and/or comparative effectiveness: Recommendations from the joint ISPOR-ISPE Special Task Force on real-world evidence in health care decision making. *Pharmacoepidemiol Drug Saf*, **26**(9), 1033–1039.

Bhaskar, H., Hoyle, D. C., and Singh, S. (2006). Machine learning in bioinformatics: A brief survey and recommendations for practitioners. *Computers in Biology and Medicine*, **36**(10), 1104–1125.

Borisov, N., Shabalina, I., Tkachev, V., Sorokin, M., Garazha, A., Pulin, A., Eremin, I. I., and Buzdin, A. (2019). Shambhala: a platform-agnostic data harmonizer for gene expression data. *BMC Bioinformatics*, **20**(1), 66.

Boughorbel, S., Jarray, F., and El-Anbari, M. (2017). Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. *PLOS ONE*, **12**(6), e0177678.

Breiman, L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32.

Bumgarner, R. (2013). Overview of DNA Microarrays: Types, Applications, and Their Future. *Curr Protoc Mol Biol*, **101**(1), 22.1.1–22.1.11.

Caron, E., Kowalewski, D., Koh, C. C., Sturm, T., Schuster, H., and Aebersold, R. (2015). Analysis of Major Histocompatibility Complex (MHC) Immunopeptidomes Using Mass Spectrometry*. *Mol. Cell. Proteom.*, **14**(12), 3105–3117.

Chen, B., Khodadoust, M. S., Olsson, N., Wagar, L. E., Fast, E., Liu, C. L., Muftuoglu, Y., Sworder, B. J., Diehn, M., Levy, R., Davis, M. M., Elias, J. E., Altman, R. B., and Alizadeh, A. A. (2019). Predicting HLA class II antigen presentation through integrated deep learning. *Nat. Biotechnol.*, **37**(11), 1332–1343.

Chen, T. and Guestrin, C. (2016). XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Conroy, T., Desseigne, F., Ychou, M., Bouché, O., Guimbaud, R., Bécouarn, Y., Adenis, A., Raoul, J.-L., Gourgou-Bourgade, S., de la Fouchardière, C., Bennouna, J., Bachet, J.-B., Khemissa-Akouz, F., Péré-Vergé, D., Delbaldo, C., Assenat, E., Chauffert, B., Michel, P., Montoto-Grillot, C., and Ducreux, M. (2011). FOLFIRINOX versus gemcitabine for metastatic pancreatic cancer. *New England Journal of Medicine*, **364**(19), 1817–1825.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Mach Learn*, **20**(3), 273–297.

Deplanque, G. and Demartines, N. (2017). Pancreatic cancer: are more chemotherapy and surgery needed? *The Lancet*, **389**(10073), 985–986.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognit Lett*, **27**(8), 861–874.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2018). Practical automated machine learning for the automl challenge 2018.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2021). Auto-sklearn 2.0: Hands-free automl via meta-learning.

Food and Drug Administration, U. (2018). Framework for FDAs Real-World Evidence program.

Freund, Y. and Schapire, R. E. (1999). A short introduction to boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, **29**(5).

Gal, J., Bailleux, C., Chardin, D., Pourcher, T., Gilhodes, J., Jing, L., Guigonis, J.-M., Ferrero, J.-M., Milano, G., Mograbi, B., Brest, P., Chateau, Y., Humbert, O., and Chamorey, E. (2020). Comparison of unsupervised machine-learning methods to identify metabolomic signatures in patients with localized breast cancer. *Computational and Structural Biotechnology Journal*, **18**, 1509–1524.

Galperin, M. Y., Fernández-Suárez, X. M., and Rigden, D. J. (2016). The 24th annual Nucleic Acids Research database issue: a look back and upcoming changes. *Nucleic Acids Research*, **45**(D1), D1–D11.

Garrido-Laguna, I. and Hidalgo, M. (2015). Pancreatic cancer: from state-of-the-art treatments to promising novel therapies. *Nature Reviews Clinical Oncology*, **12**(6), 319–334.

Giménez, N., Martínez-Trillos, A., Montraveta, A., Lopez-Guerra, M., Rosich, L., Nadeu, F., Valero, J. G., Aymerich, M., Magnano, L., Rozman, M., Matutes, E., Delgado, J., Baumann, T., Gine, E., González, M., Alcoceba, M., Terol, M. J., Navarro, B., Colado, E., Payer, A. R., Puente, X. S., López-Otín, C., Lopez-Guillermo, A., Campo, E., Colomer, D., and Villamor, N. (2018). Mutations in the RAS-BRAF-MAPK-ERK pathway define a specific subgroup of patients with adverse clinical features and provide new therapeutic options in chronic lymphocytic leukemia. *Haematologica*, **104**(3), 576–586.

Giménez, N., Schulz, R., Higashi, M., Aymerich, M., Villamor, N., Delgado, J., Juan, M., López-Guerra, M., Campo, E., Rosich, L., Seiffert, M., and Colomer, D. (2019). Targeting IRAK4 disrupts inflammatory pathways and delays tumor development in chronic lymphocytic leukemia. *Leukemia*, **34**(1), 100–114.

Gulshan, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., Kim, R., Raman, R., Nelson, P. C., Mega, J. L., and Webster, D. R. (2016). Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA*, **316**(22), 2402.

Gutiérrez-Casares, J. R., Quintero, J., Jorba, G., Junet, V., Martínez, V., Pozo-Rubio, T., Oliva, B., Daura, X., Mas, J. M., and Montoto, C. (2021). Methods to develop an in silico clinical trial: Computational head-to-head comparison of lisdexamfetamine and methylphenidate. *Frontiers in Psychiatry*, **12**, 1902.

Haury, A.-C., Gestraud, P., and Vert, J.-P. (2011). The influence of feature selection methods on accuracy, stability and interpretability of molecular signatures. *PLOS ONE*, **6**(12), 1–12.

Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, **89**(22), 10915–10919.

Hernán, M. A., Clayton, D., and Keiding, N. (2011). The Simpson's paradox unraveled. *International Journal of Epidemiology*, **40**(3), 780–785.

Hoff, D. D. V., Ervin, T., Arena, F. P., Chiorean, E. G., Infante, J., Moore, M., Seay, T., Tjulandin, S. A., Ma, W. W., Saleh, M. N., Harris, M., Reni, M., Dowden, S., Laheru, D., Bahary, N., Ramanathan, R. K., Tabernero, J., Hidalgo, M., Goldstein, D., Cutsem, E. V., Wei, X., Iglesias, J., and Renschler, M. F. (2013). Increased survival in pancreatic cancer with nab-paclitaxel plus gemcitabine. *New England Journal of Medicine*, **369**(18), 1691–1703.

Hubbell, E., Liu, W.-M., and Mei, R. (2002). Robust estimators for expression analysis. *Bioinformatics*, **18**(12), 1585–1592.

Irigoyen, A., Jimenez-Luna, C., Benavides, M., Caba, O., Gallego, J., Ortuño, F. M., Guillen-Ponce, C., Rojas, I., Aranda, E., Torres, C., and Prados, J. (2018). Integrative multi-platform meta-analysis of gene expression profiles in pancreatic ductal adenocarcinoma patients for identifying novel diagnostic biomarkers. *PLOS ONE*, **13**(4), e0194844.

Irizarry, R. A., Hobbs, B., Collin, F., BeazerBarclay, Y. D., Antonellis, K. J., Scherf, U., and Speed, T. P. (2003). Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, **4**(2), 249–264.

Iwawaki, T., Hosoda, A., Okuda, T., Kamigori, Y., Nomura-Furuwatari, C., Kimata, Y., Tsuru, A., and Kohno, K. (2001). Translational control by the ER transmembrane kinase/ribonuclease IRE1 under ER stress. *Nature Cell Biology*, **3**(2), 158–164.

Jassal, B., Matthews, L., Viteri, G., Gong, C., Lorente, P., Fabregat, A., Sidiropoulos, K., Cook, J., Gillespie, M., Haw, R., Loney, F., May, B., Milacic, M., Rothfels, K., Sevilla, C., Shamovsky, V., Shorser, S., Varusai, T., Weiser, J., Wu, G., Stein, L., Hermjakob, H., and D'Eustachio, P. (2019). The reactome pathway knowledgebase. *Nucleic Acids Research*.

Jensen, K. K., Andreatta, M., Marcatili, P., Buus, S., Greenbaum, J. A., Yan, Z., Sette, A., Peters, B., and Nielsen, M. (2018). Improved methods for predicting peptide binding affinity to MHC class II molecules. *Immunology*, **154**(3), 394–406.

Johnson, W. E., Li, C., and Rabinovic, A. (2007). Adjusting batch effects in microarray expression data using empirical Bayes methods. *Biostatistics*, **8**(1), 118–127.

Jorba, G., Aguirre-Plans, J., Junet, V., Segú-Vergés, C., Ruiz, J. L., Pujol, A., Fernández-Fuentes, N., Mas, J. M., and Oliva, B. (2020). In-silico simulated prototype-patients using TPMS technology to study a potential adverse effect of sacubitril and valsartan. *PLOS ONE*, **15**(2), e0228926.

Junet, V. and Daura, X. (2021). CNN-PepPred: An open-source tool to create convolutional NN models for the discovery of patterns in peptide sets. Application to peptide-MHC class II binding prediction. *Bioinformatics*. btab687.

Junet, V., Farrés, J., Mas, J. M., and Daura, X. (2021). CuBlock: a cross-platform normalization method for gene-expression microarrays. *Bioinformatics*, **37**(16), 2365–2373.

Kamnitsas, K., Ledig, C., Newcombe, V. F., Simpson, J. P., Kane, A. D., Menon, D. K., Rueckert, D., and Glocker, B. (2017). Efficient multi-scale 3d CNN with fully connected CRF for accurate brain lesion segmentation. *Medical Image Analysis*, **36**, 61–78.

Kanehisa, M. (2000). KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, **28**(1), 27–30.

Kanehisa, M. (2019). Toward understanding the origin and evolution of cellular organisms. *Protein Science*, **28**(11), 1947–1951.

Kanehisa, M., Furumichi, M., Sato, Y., Ishiguro-Watanabe, M., and Tanabe, M. (2020). KEGG: integrating viruses and cellular organisms. *Nucleic Acids Research*, **49**(D1), D545–D551.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 11371143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kolpakov, F., Akberdin, I., Kashapov, T., llya Kiselev, Kolmykov, S., Kondrakhin, Y., Kutumova, E., Mandrik, N., Pintus, S., Ryabova, A., Sharipov, R., Yevshin, I., and Kel, A. (2019). BioUML: an integrated environment for systems biology and collaborative analysis of biomedical data. *Nucleic Acids Research*, **47**(W1), W225–W233.

Larrañaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I., Lozano, J. A., Armañanzas, R., Santafé, G., Pérez, A., and Robles, V. (2006). Machine learning in bioinformatics. *Briefings in Bioinformatics*, **7**(1), 86–112.

Lashkari, D. A., DeRisi, J. L., McCusker, J. H., Namath, A. F., Gentile, C., Hwang, S. Y., Brown, P. O., and Davis, R. W. (1997). Yeast microarrays for genome wide parallel genetic and gene expression analysis. *Proc Natl Acad Sci USA*, **94**(24), 13057–13062.

Lê Cao, K.-A., Rohart, F., McHugh, L., Korn, O., and Wells, C. A. (2014). YuGene: A simple approach to scale gene expression data derived from different platforms for integrated analyses. *Genomics*, **103**(4), 239–251.

Liu, C. Y., Xu, Z., and Kaufman, R. J. (2003). Structure and intermolecular interactions of the luminal dimerization domain of human IRE1α. *Journal of Biological Chemistry*, **278**(20), 17680–17687.

Liu, J., Wang, Y., Song, L., Zeng, L., Yi, W., Liu, T., Chen, H., Wang, M., Ju, Z., and Cong, Y.-S. (2017). A critical role of DDRGK1 in endoplasmic reticulum homoeostasis via regulation of IRE1α stability. *Nature Communications*, **8**(1).

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Trans Inf Theory*, **28**(2), 129–137.

Lucas, F., Babot, E. D., Cañellas, M., del Río, J. C., Kalum, L., Ullrich, R., Hofrichter, M., Guallar, V., Martínez, A. T., and Gutiérrez, A. (2016). Molecular determinants for selective c25-hydroxylation of vitamins d2and d3by fungal peroxygenases. *Catalysis Science & Technology*, **6**(1), 288–295.

Maire, V., Némati, F., Richardson, M., Vincent-Salomon, A., Tesson, B., Rigaill, G., Gravier, E., Marty-Prouvost, B., De Koning, L., Lang, G., Gentien, D., Dumont, A., Barillot, E., Marangoni, E., Decaudin, D., Roman-Roman, S., Pierré, A., Cruzalegui, F., Depil, S., Tucker, G. C., and Dubois, T. (2013a). Polo-like Kinase 1: A potential therapeutic option in combination with conventional chemotherapy for the management of patients with triple-negative breast cancer. *Cancer Res*, **73**(2), 813–823.

Maire, V., Baldeyron, C., Richardson, M., Tesson, B., Vincent-Salomon, A., Gravier, E., Marty-Prouvost, B., De Koning, L., Rigaill, G., Dumont, A., Gentien, D., Barillot, E., Roman-Roman, S., Depil, S., Cruzalegui, F., Pierré, A., Tucker, G. C., and Dubois, T. (2013b). TTK/hMPS1 is an attractive therapeutic target for triple-negative breast cancer. *PLOS ONE*, **8**(5), 1–15.

Maitra, A. and Hruban, R. H. (2008). Pancreatic cancer. *Annual Review of Pathology: Mechanisms of Disease*, **3**(1), 157–188.

MAQC Consortium (2006). The MicroArray Quality Control (MAQC) project shows inter- and intraplatform reproducibility of gene expression measurements. *Nat Biotechnol*, **24**(9), 1151–1161.

Maubant, S., Tesson, B., Maire, V., Ye, M., Rigaill, G., Gentien, D., Cruzalegui, F., Tucker, G. C., Roman-Roman, S., and Dubois, T. (2015). Transcriptome analysis of Wnt3a-treated triple-negative breast cancer cells. *PLOS ONE*, **10**(4), 1–26.

Meng, Q., Catchpoole, D., Skillicorn, D., and Kennedy, P. J. (2017). DBNorm: normalizing high-density oligonucleotide microarray data based on distributions. *BMC Bioinformatics*, **18**(1), 527.

Moncunill, G., Scholzen, A., Mpina, M., Nhabomba, A., Hounkpatin, A. B., Osaba, L., Valls, R., Campo, J. J., Sanz, H., Jairoce, C., Williams, N. A., Pasini, E. M., Arteta, D., Maynou, J., Palacios, L., Duran-Frigola, M., Aponte, J. J., Kocken, C. H. M., Agnandji, S. T., Mas, J. M., Mordmller, B., Daubenberger, C., Sauerwein, R., and Dobaño, C. (2020). Antigen-stimulated pbmc transcriptional protective signatures for malaria immunization. *Science Translational Medicine*, **12**(543), eaay8924.

Morales, R., Serrano, R., Sardón, T., Román-Ortíz, C., González-Santiago, S., Mulet, R., and Mas, J. M. (2017). Functional, structural and contextual analysis of a variant of uncertain clinical significance in BRCA1: C.5434c->g (p.pro1812ala). *Journal of Cancer Genetics and Biomarkers*, **1**(2), 1–14.

Nagalakshmi, U., Wang, Z., Waern, K., Shou, C., Raha, D., Gerstein, M., and Snyder, M. (2008). The Transcriptional Landscape of the Yeast Genome Defined by RNA Sequencing. *Science*, **320**(5881), 1344–1349.

Natekin, A. and Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, **7**.

Nielsen, M. and Andreatta, M. (2017). NNAlign: a platform to construct and evaluate artificial neural network models of receptor-ligand interactions. *Nucleic Acids Res.*, **45**(W1), W344–W349.

Nielsen, M. and Lund, O. (2009). NN-align. An artificial neural network-based alignment algorithm for MHC class II peptide binding prediction. *BMC Bioinformatics*, **10**, 296.

Obrzut, B., Kusy, M., Semczuk, A., Obrzut, M., and Kluska, J. (2017). Prediction of 5–year overall survival in cervical cancer patients treated with radical hysterectomy using computational intelligence methods. *BMC Cancer*, **17**(1).

Olson, R. S., Cava, W. L., Mustahsan, Z., Varik, A., and Moore, J. H. (2017). Data-driven advice for applying machine learning to bioinformatics problems. In *Biocomputing 2018*. WORLD SCIENTIFIC.

Oughtred, R., Rust, J., Chang, C., Breitkreutz, B.-J., Stark, C., Willems, A., Boucher, L., Leung, G., Kolas, N., Zhang, F., Dolma, S., Coulombe-Huntington, J., Chatr-aryamontri, A., Dolinski, K., and Tyers, M. (2020). TheBioGRIDdatabase: A comprehensive biomedical resource of curated protein, genetic, and chemical interactions. *Protein Science*, **30**(1), 187–200.

Parvizpour, S., Pourseif, M. M., Razmara, J., Rafi, M. A., and Omidi, Y. (2020). Epitope-based vaccine design: a comprehensive overview of bioinformatics approaches. *Drug Discovery Today*, **25**(6), 1034–1042.

Piccolo, S. R., Sun, Y., Campbell, J. D., Lenburg, M. E., Bild, A. H., and Johnson, W. E. (2012). A single-sample microarray normalization method to facilitate personalized-medicine workflows. *Genomics*, **100**(6), 337–344.

Piccolo, S. R., Withers, M. R., Francis, O. E., Bild, A. H., and Johnson, W. E. (2013). Multiplatform single-sample estimates of transcriptional activation. *Proc Natl Acad Sci USA*, **110**(44), 17778–17783.

Platts, A. E., Dix, D. J., Chemes, H. E., Thompson, K. E., Goodrich, R., Rockett, J. C., Rawe, V. Y., Quintana, S., Diamond, M. P., Strader, L. F., and Krawetz, S. A. (2007). Success and failure in human spermatogenesis as revealed by teratozoospermic RNAs. *Hum Mol Genet*, **16**(7), 763–773.

Racle, J., Michaux, J., Rockinger, G. A., Arnaud, M., Bobisse, S., Chong, C., Guillaume, P., Coukos, G., Harari, A., Jandus, C., Bassani-Sternberg, M., and Gfeller, D. (2019). Robust prediction of HLA class II epitopes by deep motif deconvolution of immunopeptidomes. *Nat. Biotechnol.*, **37**(11), 1283–1286.

Rapaport, F., Khanin, R., Liang, Y., Pirun, M., Krek, A., Zumbo, P., Mason, C. E., Socci, N. D., and Betel, D. (2013). Comprehensive evaluation of differential gene expression analysis methods for rna-seq data. *Genome Biology*, **14**(1), 3158.

Reynisson, B., Alvarez, B., Paul, S., Peters, B., and Nielsen, M. (2020). NetMHCpan-4.1 and NetMHCIIpan-4.0: improved predictions of MHC antigen presentation by concurrent motif deconvolution and integration of MS MHC eluted ligand data. *Nucleic Acids Res.*, **48**(W1), W449–W454.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J Comput Appl Math*, **20**, 53–65.

Rudy, J. and Valafar, F. (2011). Empirical comparison of cross-platform normalization methods for gene expression data. *BMC Bioinformatics*, **12**(1), 467.

Saeys, Y., Inza, I., and Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, **23**(19), 2507–2517.

Schena, M., Shalon, D., Davis, R. W., and Brown, P. O. (1995). Quantitative Monitoring of Gene Expression Patterns with a Complementary DNA Microarray. *Science*, **270**(5235), 467–470.

Schultz, A., Mehta, S., Hu, C., Hoff, F., Horton, T., Kornblau, S., and Qutub, A. (2016). Identifying cancer specific metabolic signatures using constraint-based models. In *Biocomputing 2017*. World Scientific.

Segú-Vergés, C., Coma, M., Kessel, C., Smeets, S., Foell, D., and Aldea, A. (2021). Application of systems biology-based in silico tools to optimize treatment strategy identification in still's disease. *Arthritis Research & Therapy*, **23**(1).

SEQC/MAQC-III Consortium (2014). A comprehensive assessment of RNA-seq accuracy, reproducibility and information content by the Sequencing Quality Control Consortium. *Nat Biotechnol*, **32**(9), 903–914.

Sette, A. and Rappuoli, R. (2010). Reverse Vaccinology: Developing Vaccines in the Era of Genomics. *Immunity*, **33**(4), 530–541.

Shabalin, A. A., Tjelmeland, H., Fan, C., Perou, C. M., and Nobel, A. B. (2008). Merging two gene-expression studies via cross-platform normalization. *Bioinformatics*, **24**(9), 1154–1160.

Shastry, K. A. and Sanjay, H. A. (2020). *Machine Learning for Bioinformatics*, pages 25–39. Springer Singapore, Singapore.

Sherman, R. E., Davies, K. M., Robb, M. A., Hunter, N. L., and Califf, R. M. (2017). Accelerating development of scientific evidence for medical products within the existing US regulatory framework. *Nat Rev Drug Discov*, **16**(5), 297–298.

Shimodaira, H. (2004). Approximately unbiased tests of regions using multistep-multiscale bootstrap resampling. *Ann Stat*, **32**(6), 2616–2641.

Siegel, R. L., Miller, K. D., and Jemal, A. (2018). Cancer statistics, 2018. *CA: A Cancer Journal for Clinicians*, **68**(1), 7–30.

Simantiraki, O., Charonyktakis, P., Pampouchidou, A., Tsiknakis, M., and Cooke, M. (2017). Glottal source features for automatic speech-based depression assessment. In *Interspeech 2017*. ISCA.

Sood, R., Porter, A., Ma, K., Quilliam, L., and Wek, R. (2000). Pancreatic eukaryotic initiation factor-2alpha kinase (PEK) homologues in humans, drosophila melanogaster and caenorhabditis elegans that mediate translational control in response to endoplasmic reticulum stress. *Biochem J*, **346 Pt 2**, 281–293.

Storey, J. D. (2002). A Direct Approach to False Discovery Rates. *J Royal Stat Soc B*, **64**(3), 479–498.

Subbalakshmi, A. R., Sahoo, S., Biswas, K., and Jolly, M. K. (2021). A computational systems biology approach identifies SLUG as a mediator of partial epithelial-mesenchymal transition (EMT). *Cells Tissues Organs*, pages 1–14.

Suzuki, R. and Shimodaira, H. (2006). Pvclust: an R package for assessing the uncertainty in hierarchical clustering. *Bioinformatics*, **22**(12), 1540–1542.

Tareen, A. and Kinney, J. B. (2019). Logomaker: beautiful sequence logos in Python. *Bioinformatics*, **36**(7), 2272–2274.

Tirasophon, W., Welihinda, A. A., and Kaufman, R. J. (1998). A stress response pathway from the endoplasmic reticulum to the nucleus requires a novel bifunctional protein kinase/endoribonuclease (ire1p) in mammalian cells. *Genes & Development*, **12**(12), 1812–1824.

Trotta, F. (2012). Discrepancies between observational studies and randomized controlled trials. *Focus Farmacovigilanza*, **73**(11), 1.

Tsamardinos, I., Greasidou, E., and Borboudakis, G. (2018). Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation. *Machine Learning*, **107**(12), 1895–1922.

Tsamardinos, I., Charonyktakis, P., Lakiotaki, K., Borboudakis, G., Zenklusen, J. C., Juhl, H., Chatzaki, E., and Lagani, V. (2020). Just add data: Automated predictive modeling and biosignature discovery. *bioRxiv*.

van der Maaten, L. and Hinton, G. (2008). Visualizing Data using t-SNE. *J Mach Learn Res*, **9**, 2579–2605.

Villalba, A., Rodriguez-Fernandez, S., Perna-Barrull, D., Ampudia, R.-M., Gomez-Muñoz, L., Pujol-Autonell, I., Aguilera, E., Risueño, R. M., Cano-Sarabia, M., Maspoch, D., Vázquez, F., and Vives-Pi, M. (2020). Antigen-specific immunotherapy combined with a regenerative drug in the treatment of experimental type 1 diabetes. *Scientific Reports*, **10**(1).

Walsh, C. J., Hu, P., Batt, J., and Dos Santos, C. C. (2015). Microarray Meta-Analysis and Cross-Platform Normalization: Integrative Genomics for Robust Biomarker Discovery. *Microarrays*, **4**(3), 389–406.

Wang, P., Sidney, J., Kim, Y., Sette, A., Lund, O., Nielsen, M., and Peters, B. (2010). Peptide binding predictions for HLA DR, DP and DQ molecules. *BMC Bioinformatics*, **11**, 568.

Wishart, D. S., Feunang, Y. D., Guo, A. C., Lo, E. J., Marcu, A., Grant, J. R., Sajed, T., Johnson, D., Li, C., Sayeeda, Z., Assempour, N., Iynkkaran, I., Liu, Y., Maciejewski, A., Gale, N., Wilson, A., Chin, L., Cummings, R., Le, D., Pon, A., Knox, C., and Wilson, M. (2017). DrugBank 5.0: a major update to the DrugBank database for 2018. *Nucleic Acids Research*, **46**(D1), D1074–D1082.

Wren, J. D. and Bateman, A. (2008). Databases, data tombs and dust in the wind. *Bioinformatics*, **24**(19), 2127–2128.

Yang, F., Wang, H.-z., Mi, H., Lin, C.-d., and Cai, W.-w. (2009). Using random forest for reliable classification and cost-sensitive learning for medical diagnosis. *BMC Bioinformatics*, **10**(S1).

Yang, Y. H., Dudoit, S., Luu, P., Lin, D. M., Peng, V., Ngai, J., and Speed, T. P. (2002). Normalization for cdna microarray data: a robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Res*, **30**(4), e15.

Yoshida, H., Matsui, T., Yamamoto, A., Okada, T., and Mori, K. (2001). XBP1 mRNA is induced by ATF6 and spliced by IRE1 in response to ER stress to produce a highly active transcription factor. *Cell*, **107**(7), 881–891.

Zhang, S., Shao, J., Yu, D., Qiu, X., and Zhang, J. (2020). MatchMixeR: a cross-platform normalization method for gene expression data integration. *Bioinformatics*, **36**(8), 2486–2491.

Appendix A: Supplementary information for Chapter 1

GetTargetValues(BS)
BS is a sorted array with mean 0 and standard deviation 1.

```
$L \leftarrow$ line of same size as $BS$ between $-1$ and 1 with equidistant points
$iStdUp \leftarrow$ the index corresponding to the point in $BS$ which is the closest to 1
$iStdDown \leftarrow$ the index corresponding to the point in $BS$ which is the closest to $-1$
FOR $p \in [3,5,7,9,11,13,15,17,21]$
  $D \leftarrow L^p$
  IF mean($D[iStdDown: iStdUp]$) IS SMALLER than 0.1
    RETURN $D$
  END IF
END FOR
END FUNCTION
```

Figure A.1: Pseudocode describing the GetTargetValues algorithm called in the CuBlock algorithm (Figure 1 in the main text). An example illustration is given in Figure 1.4.

```
**ModPol**$(B, P)$
$B$ are the block values.
$P$ is the polynomial previously fitted to the block $B$.

    $n \leftarrow$ size of $B$
    $BS \leftarrow$ sort $B$ in ascending order
    $\widehat{BS} \leftarrow$ evaluate the polynomial $P$ at $BS$
    IF1 $\widehat{BS}$ ISNOT sorted in ascending order
      $[xDown1, iDown1] \leftarrow$ get the first value and corresponding index in $\widehat{BS}$
                                   where the array decreases
      $[xDownL, iDownL] \leftarrow$ get the last value and corresponding index in $\widehat{BS}$
                                   where the array decreases
      $[xUp1, iUp1] \leftarrow$ get the first value and corresponding index in $\widehat{BS}$
                               where the array increases
      $[xUpL, iUp] \leftarrow$ get the last value and corresponding index in $\widehat{BS}$
                              where the array increases
      IF2 $\widehat{BS}[iUp1]$ IS EQUAL to $\widehat{BS}[1]$ AND $\widehat{BS}[iUpL]$ IS EQUAL to $\widehat{BS}[n]$
        IF3 the number of points between the first index and $iDown1$ ISBIGGER
            than the number of point between $iDownL$ and the last index $n$
          $\widehat{BS}[iDown1{:}\, iDownL] \leftarrow xDown1$
          $\widehat{BS}[(iDownL + 1){:}\, n] \leftarrow xDown1 + \widehat{BS}[(iDownL + 1){:}\, n] - xDownL$
        ELSE3
          $\widehat{BS}[iDown1{:}\, iDownL] \leftarrow xDownL$
          $\widehat{BS}[1{:}\, (iDown1 - 1)] \leftarrow xDownL + \widehat{BS}[1{:}\, (iDown1 - 1)] - xDown1$
        END IF3
      ELSE2
        $\widehat{BS}[iUpL{:}\, n] \leftarrow xUpL$
        $\widehat{BS}[1{:}\, iUp1] \leftarrow xUp1$
      END IF2
    END IF1
    $\hat{B} \leftarrow$ sort $\widehat{BS}$ back in original order of $B$
    END FUNCTION
```

Figure A.2: Pseudocode describing the ModPol algorithm called in the CuBlock algorithm (Figure 1.2). An example illustration is given in Figure 1.5.

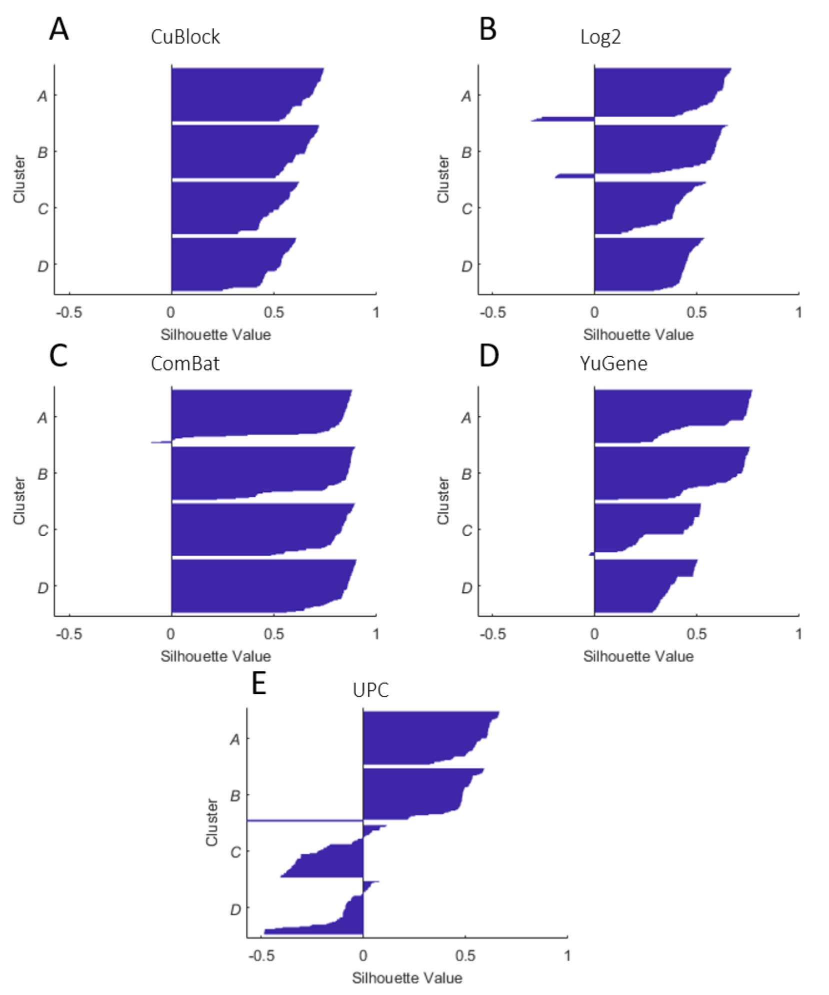


Figure A.3: Silhouette plots of the reference data set (six platforms) after normalization with CuBlock (A), $\log_2$ (B), ComBat (C), YuGene (D) and UPC (E), using the groups A, B, C and D as given clusters. Mean silhouette index (SI) values: A: 0.66 (A), 0.62 (B), 0.51 (C) and 0.50 (D); B: 0.51 (A), 0.49 (B), 0.38 (C) and 0.44 (D); C: 0.72 (A), 0.78 (B), 0.79 (C) and 0.83 (D); D: 0.61 (A), 0.63 (B), 0.35 (C) and 0.39 (D). E: 0.56 (A), 0.44 (B), -0.19 (C) and -0.13 (D).

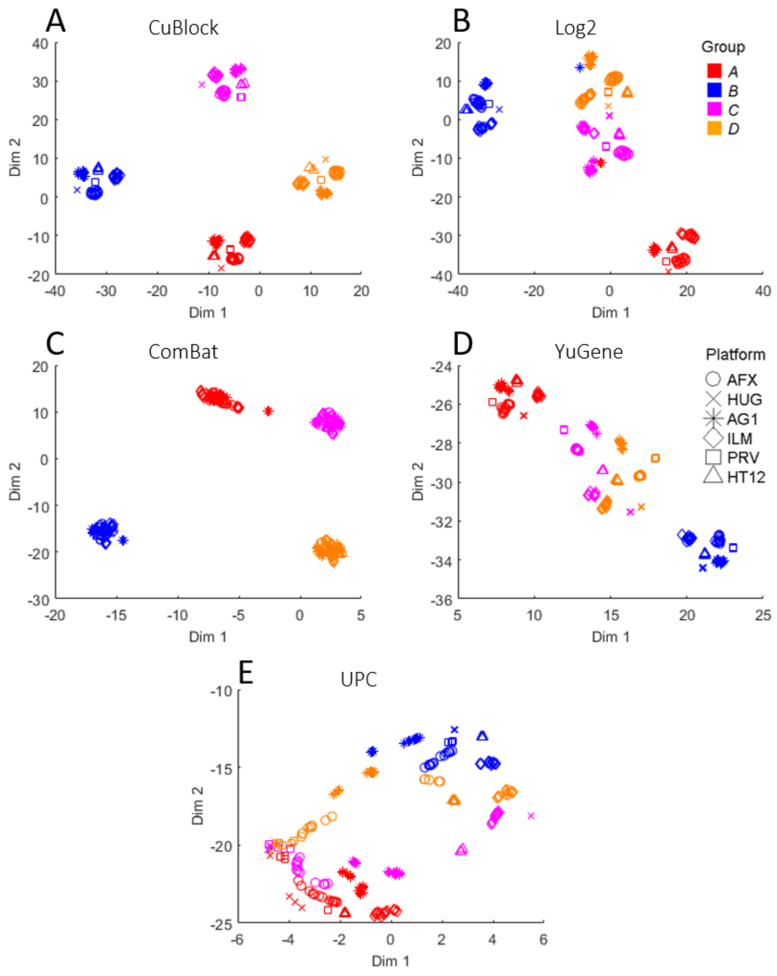


Figure A.4: *t*-SNE dimension reduction of the reference data set after normalization with CuBlock, $\log_2$, ComBat, YuGene and UPC. A: *t*-SNE for CuBlock normalized data; point color and shape indicate biological group and platform, respectively (right-hand legend); perplexity (Prp) and mean silhouette index (SI) values (see Section 1.3.3.1): Prp = 25, SI = 0.97. B: *t*-SNE for $\log_2$-normalized data; Prp = 20, SI = 0.74. C: *t*-SNE for ComBat-normalized data; Prp = 45, SI = 0.96. D: *t*-SNE for YuGene-normalized data; Prp = 78, SI = 0.68. E: *t*-SNE for UPC-normalized data; Prp = 78, SI = 0.20.

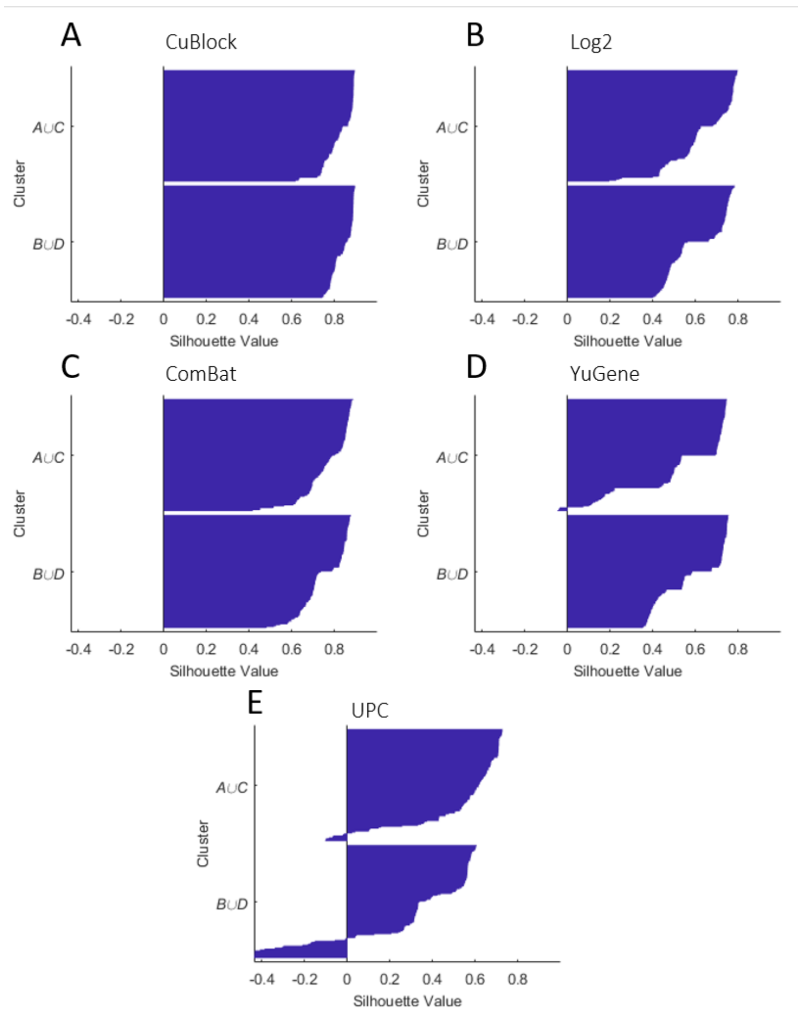


Figure A.5: Silhouette plots of the reference data set after normalization with CuBlock, $\log_2$, ComBat, YuGene and UPC. The given clusters are the groups $A \cup C$ and $B \cup D$. A: silhouette plot for CuBlock-normalized data; mean silhouette index (SI) values per group: 0.83 ($A \cup C$) and 0.84 ($B \cup D$). B: silhouette plot for $\log_2$-normalized data; SI values: 0.64 ($A \cup C$) and 0.61 ($B \cup D$). C: silhouette plot for ComBat-normalized data; SI values: 0.77 ($A \cup C$) and 0.75 ($B \cup D$). D: silhouette plot for YuGene-normalized data; SI values: 0.53 ($A \cup C$) and 0.59 ($B \cup D$). E: silhouette plot for UPC-normalized data; SI values: 0.53 ($A \cup C$) and 0.31 ($B \cup D$).

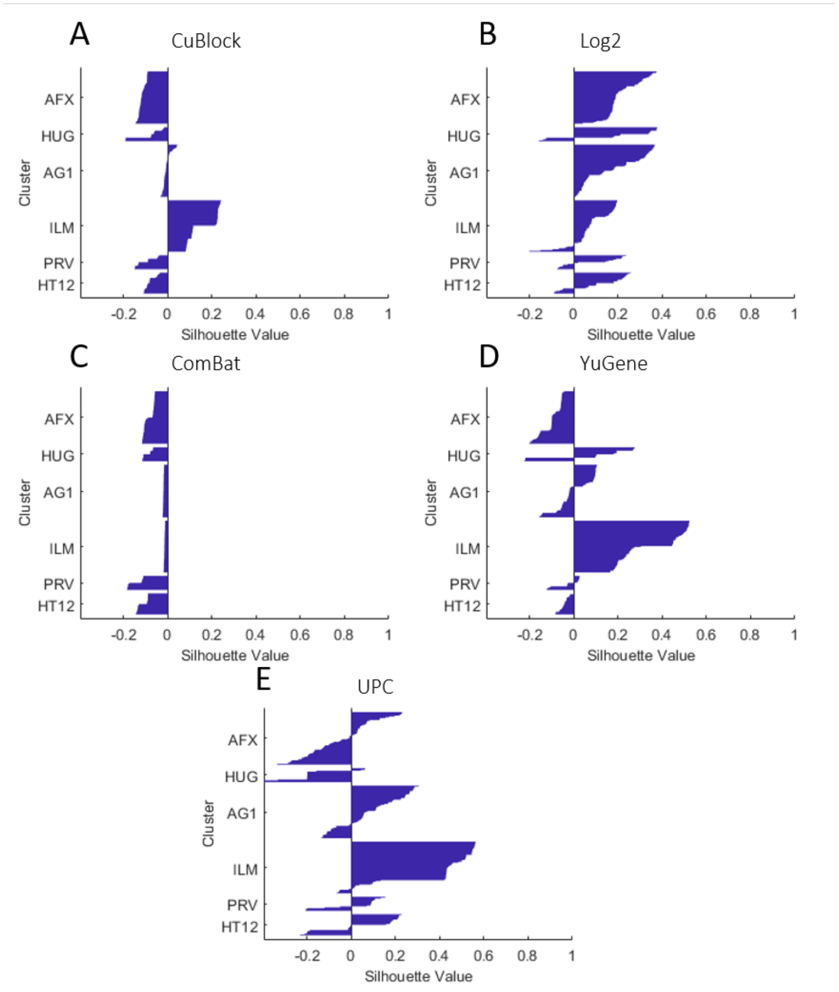


Figure A.6: Silhouette plots of the reference data set after normalization with CuBlock, $\log_2$, ComBat, YuGene and UPC. The given clusters are the platforms. A: silhouette plot for CuBlock-normalized data; mean silhouette index (SI) values per platform: -0.12 (AFX), -0.09 (HUG), 0.00 (AG1), 0.16 (ILM), -0.10 (PRV) and 0.08 (HT12). B: silhouette plot for $\log_2$-normalized data; SI values: 0.22 (AFX), 0.19 (HUG), 0.17 (AG1), 0.08 (ILM), 0.08 (PRV) and 0.11 (HT12). C: silhouette plot for ComBat-normalized data; SI values: -0.08 (AFX), -0.09 (HUG), -0.02 (AG1), -0.01 (ILM), -0.15 (PRV) and -0.11 (HT12). D: silhouette plot for YuGene-normalized data; SI values: -0.10 (AFX), 0.09 (HUG), 0.01 (AG1), 0.35 (ILM), -0.03 (PRV) and -0.04 (HT12). E: silhouette plot for UPC-normalized data; SI values: -0.04 (AFX), -0.18 (HUG), 0.09 (AG1), 0.37 (ILM), 0.02 (PRV) and 0.05 (HT12).

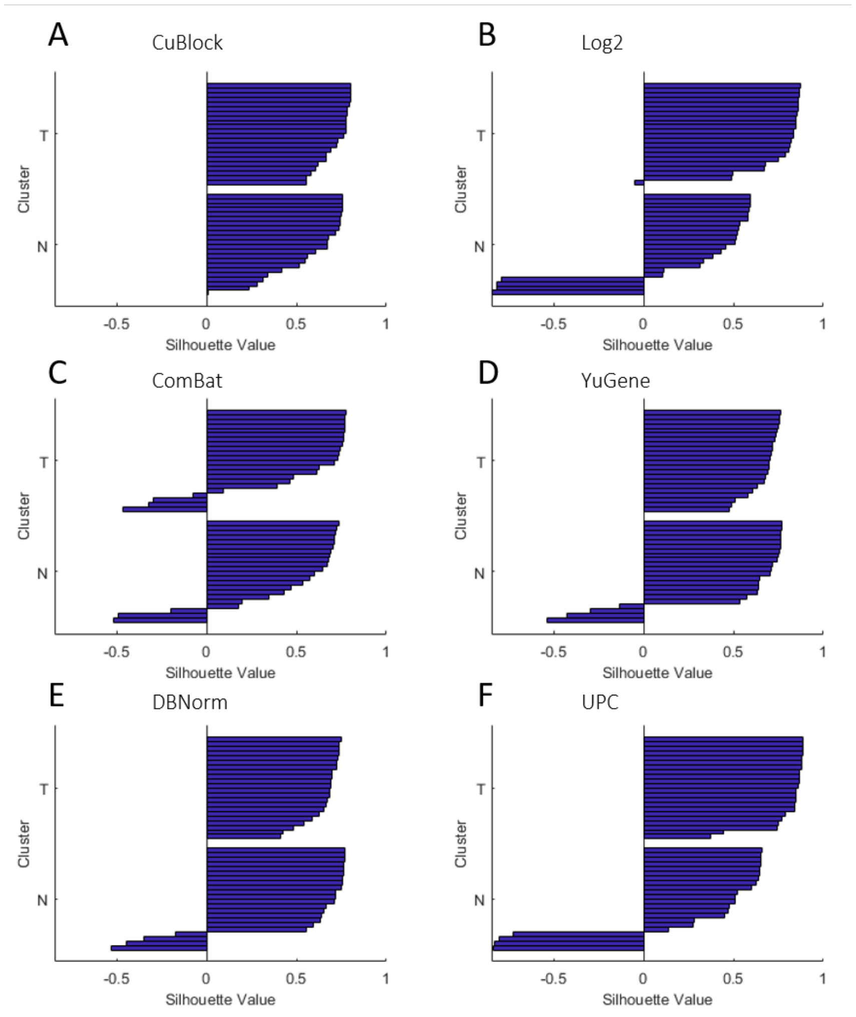


Figure A.7: Silhouette plots of the experimental data set after normalization with CuBlock (A), $\log_2$ (B), ComBat (C), YuGene (D), DBNorm (E) and UPC (F) using the groups T and N as given clusters. Mean silhouette index (SI) values: A: 0.72 (T), 0.57 (N); B: 0.75 (T), 0.23 (N); C: 0.48 (T), 0.45 (N); D: 0.67 (T), 0.51 (N); E: 0.65 (T), 0.51 (N); F: 0.81 (T), 0.28 (N).

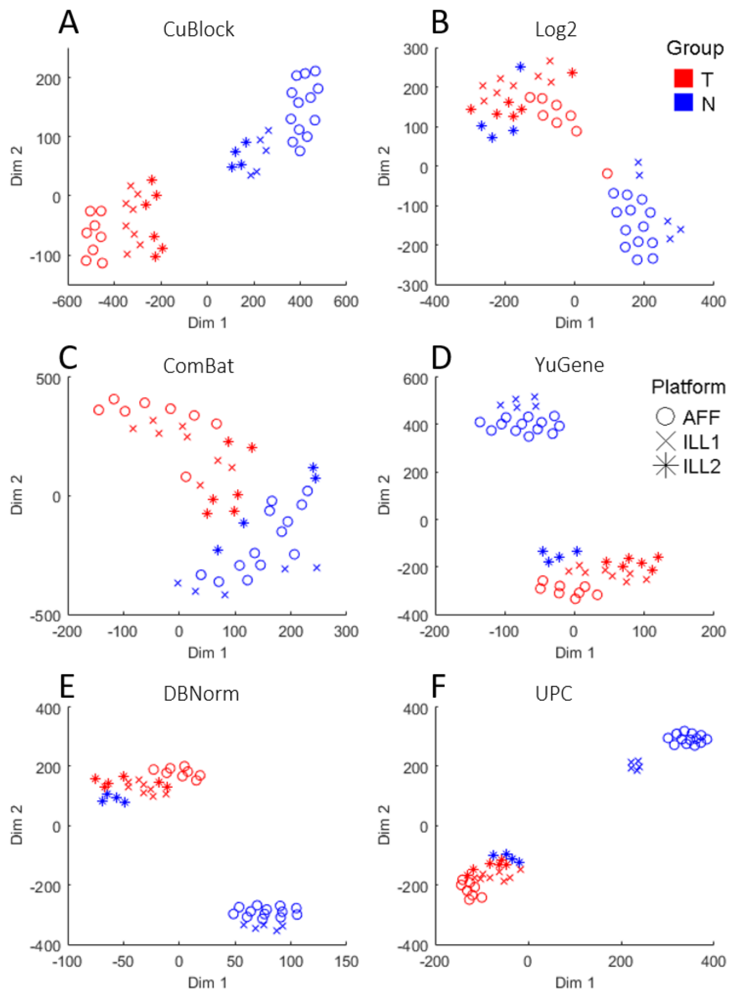


Figure A.8: t-SNE dimension reduction of the experimental data set after normalization with CuBlock, $\log_2$, ComBat, YuGene, DBNorm and UPC. A: t-SNE for CuBlock normalized data; point color and shape indicate biological group and platform, respectively (right-hand legend); perplexity (Prp) and mean silhouette index (SI) values (see Section 1.3.3.1): Prp = 10, SI = 0.93. B: t-SNE for $\log_2$-normalized data; Prp = 15, SI = 0.61. C: t-SNE for ComBat-normalized data; Prp = 10, SI = 0.62. D: t-SNE for YuGene-normalized data; Prp = 15, SI = 0.75. E: t-SNE for DBNorm-normalized data; Prp = 15, SI = 0.75. F: t-SNE for DBNorm-normalized data; Prp = 10, SI = 0.75.

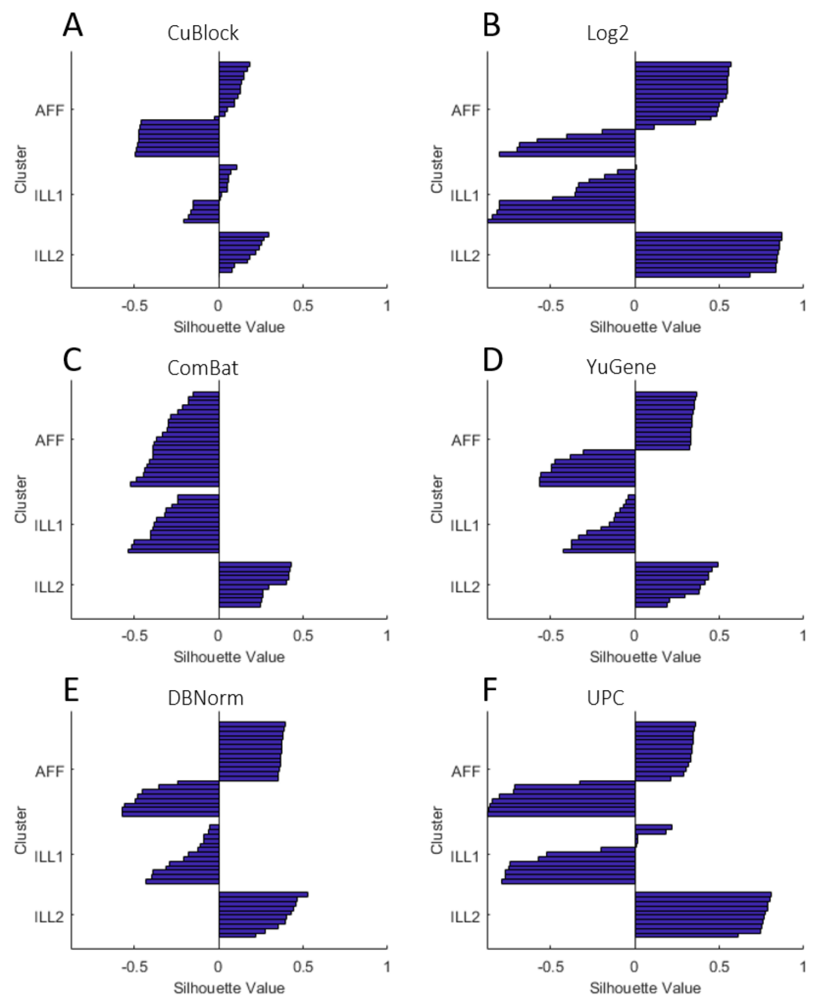


Figure A.9: Silhouette plots of the experimental data set after normalization with CuBlock, $\log_2$, ComBat, YuGene, DBNorm and UPC. The given clusters are the platforms. A: silhouette plot for CuBlock-normalized data; mean silhouette index (SI) values per platform: -0.12 (AFF), -0.03 (ILL1) and 0.18 (ILL2). B: silhouette plot for $\log_2$-normalized data; SI values: 0.19 (AFF), -0.48 (ILL1) and 0.83 (ILL2). C: silhouette plot for ComBat-normalized data; SI values: -0.34 (AFF), -0.38 (ILL1) and 0.34 (ILL2). D: silhouette plot for YuGene-normalized data; SI values: 0.03 (AFF), -0.20 (ILL1) and 0.37 (ILL2). E: silhouette plot for DBNorm-normalized data; SI values: 0.05 (AFF), -0.21 (ILL1) and 0.40 (ILL2). F: silhouette plot for UPC-normalized data; SI values: -0.09 (AFF), -0.36 (ILL1) and 0.76 (ILL2).

Appendix B: Supplementary information for Chapter 2

B.1 Installation

B.1.1 Installation with *conda*

In the folder *CNN-PepPred*, you will find environment files to create a python environment with all the required packages to run the model. There are two environment files, *model_environment_gpu.yml* and *model_environment_cpu.yml*, the first one will create an environment to work with GPUs and the second one with CPUs. GPU computations will usually be faster than CPU ones. The environment contains the following packages:

- python version 3.6.10
- numpy version 1.19.1
- tensorflow-gpu or tensorflow (for CPU environment) version 2.0.0
- keras-gpu or keras (for CPU environment) version 2.3.1
- pandas version 1.1.3
- pathlib
- biopython version 1.78
- logomaker
- scikit-learn version 0.23.2
- seaborn version 0.11.0

- pillow version 8.0.0

To create the environment, set the working directory to be the folder *CNN-PepPred* and type the following in your Anaconda terminal:

```
conda env create -f model_environment_gpu.yml
```

for the GPU environment and

```
conda env create -f model_environment_cpu.yml
```

for the CPU environment. This might take a few minutes.
Once the installation is finished, activate the environment using the command

```
conda activate CNNPepPred_Env_GPU
```

for GPU and

```
conda activate CNNPepPred_Env_CPU
```

for CPU.
At the end of the session, you can deactivate the environment using the command

```
conda deactivate
```

To remove the environment, use the command

```
conda remove --name CNNPepPred_Env_GPU --all
conda remove --name CNNPepPred_Env_CPU --all
```

B.1.2 Installation with *pip*

It is recommended to use the *conda* installation since the 3.6 version of python is required and creating an environment in Anaconda is more convenient and more uniform through different operating systems. However if you wish to do the installation using *pip*, make sure that you are using python 3.6 and create an environment and install the required packages following the instructions below.
Set the main folder *CNN-PepPred* as working directory and create a python environment called *CNNPepPred_Env_GPU* or *CNNPepPred_Env_CPU* using the lines

```
python -m venv CNNPepPred_Env_GPU
python -m venv CNNPepPred_Env_CPU
```

Activate the environment on Linux or MacOS with

```
source CNNPepPred_Env_GPU/bin/activate
source CNNPepPred_Env_CPU/bin/activate
```

and on Windows with

```
.\CNNPepPred_Env_GPU\Scripts\activate
.\CNNPepPred_Env_CPU\Scripts\activate
```

To install the required packages, as listed in the previous subsection use, for the GPU environment

```
pip install -r requirements_GPU.txt
```

and for the CPU environment

```
pip install -r requirements_CPU.txt
```

To deactive the environment, run

```
deactivate
```

B.1.3 Test

To test the installation, call the main function with the template *test_template.txt* in the *Test* folder. It will apply a pre-trained model to the sequences *test_seq.fasta*. The template contains pathways to the pre-trained model and to the data; you will need to modify these pathways in the template to be adapted to the operating system of your computer and replace *[your_working_path]* by the pathway of the folder *CNN-PepPred*. The result file *HLA_DRB1_08_01_predictedOutcome.txt* will be saved in the same folder. Check that they match the results in the file *HLA_DRB1_08_01_predictedOutcome_to_obtain.txt*.
To apply the main script, activate the previously installed environment and set the working directory to be the folder *CNN-PepPred*. If you are working from the python console, execute the lines

```
import sys
model_from_template = open("model_from_template.py").read()
sys.argv = ['model_from_template.py','test_template.txt']
exec(model_from_template)
```

Alternatively, you can run

```
import model_from_template
modelCNN = model_from_template.main('test_template.txt')
```

Or, if you are working from Spyder, you can execute the line

```
runfile('model_from_template.py',args='test_template.txt')
```

B.2 Description

The main folder *CNN-PepPred* contains two python scripts, *model_initializer.py* and *model_from_template.py*. The first contains the class *CNNPepPred*, where all the functions for training and applying allele-specific models are defined, the second launches the analysis following a user-filled template.

B.2.1 The class *CNNPepPred*

The class *CNNPepPred* is in the python script *model_initializer.py* and contains the following methods.

__init__

Description
Initialize the class. The input arguments can be read from the template.

Usage

```
CNNPepPred(allele='no_allele_name',savePath=Path(os.getcwd()),
   doTraining=False,trainingData=None,trainingOutcome=None,
   doLogoSeq=False,doCV=False,cvPart=None,kFold=5,doApplyData=
   False,trainedModelsFile=None,applyData=None,applyDataName=
   None,epitopesLength=15,parametersFile='parameters.txt')
```

Arguments

`allele`

The name of the allele.

`savePath`

The pathway where to save the results.

`doTraining`

Whether or not to do the training.

`trainingData`

The training sequences, in a list.

`trainingOutcome`

The training outcome corresponding to the training sequences.

`doLogoSeq`

Whether or not to plot (logo plot) the core binding pattern of the trained model.

`doCV`

Whether or not to perform a cross-validation.

`kFold`

The number of fold for the cross-validation.

`doApplyData`

Whether or not to apply the trained model to new sequences.

`trainedModelsFile`

The file containing the trained model. This option is only valid if no training is selected. The file is a pickle saved file from a previous training using this class.

`applyData`

The new sequences for the application of the trained model.

`applyDataName`

The name of the new sequences.

`epitopesLength`

The length of the epitopes on which the trained model will be applied. Each new sequence will be cut into all overlapping *epitopesLength*-mers and a prediction will be made for each of them.

`parametersFile`

The name with extension of the file containing the parameters of the model.

getParameters

Description
Get the parameters of the model as given by the parameter file of the template. The parameters will be saved as attributes. For more information about the parameters, see Appendix B.2.2.

Usage

`CNNPepPred.getParameters()`

aa2int

Description
Transform a sequence of amino acids to integers according to:

A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	-
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

where ”-” stands for the absence of amino acids. Any non-amino acid characters will be considered as ”-”.

Usage

`CNNPepPred.aa2int(s)`

Arguments

`s`

The amino-acid residue sequences in a list.

Value
Returns `sInt`, a list with the sequences as integers.

int2aa

Description
Transform a sequence of integers to amino acids according to the table in the description of *aa2int*.

Usage

`CNNPepPred.int2aa(sInt)`

Arguments

`sInt`

The integer sequences in a list of numpy arrays. If all sequences have the same length, it can be a numpy array of shape (N, L) where N is the number of sequences and L their length.

Value
Returns `s`, a list with the sequences as amino acid characters.

seqLength

Description
Compute the maximal length *maxL* in a set of sequences, the length *seqL* of each of them and the parameter *nMaxPool* determining the pooling size of the maxpooling layer in the model.
For more information on *nMaxPool*, see Section 2.4.2.

Usage

`CNNPepPred.seqLength(s,saveOutput=False)`

Arguments

`s`

The sequences, which can be either a list of amino-acid residue sequences or a list of integer sequences.

`saveOutput`

Whether or not to save the outputs as attributes.

Value
Returns `seqL`, a numpy array with the length of the sequences, `maxL`, the maximal length and `nMaxPool`, the pooling size of the maxpooling layer.

addEmptyPositions

Description
Add the integer value 20, standing for the absence of amino acid, to the

given sequences as needed so that they all have the same length, equal to the maximal length in the training set. In addition, it will add this value *nbPrev* times at the beginning of the sequences and *nbAfter* times at the end. The *nbPrev* and *nbAfter* parameters are set in the parameter file (Appendix B.2.2).
If the maximal length (the attribute *maxL* of the class) from a previously trained model is smaller than the maximal length in the training set (this can happen if the training is performed with transfer learning, see Section 2.4.5), the instances with length larger than *maxL* are removed from the training data.

Usage

`CNNPepPred.addEmptyPositions(sInt)`

Arguments

`sInt`

The integer sequences in a list.

Value
Returns `sIntNew`, a list of the integer sequences with the added absence-of-amino-acid values.

getImages

Description
Transform the sequences into images according to the given similarity matrix. For a given sequence, the height of the image corresponds to the residues of the sequence, the width corresponds to the 21 amino acids+absence of amino acids. The image is then filled with the similarity value between a residue of the sequence and an amino acid.
For more information on the peptide's encoding, see Section 2.4.1.

Usage

`CNNPepPred.getImages(sInt)`

Arguments

`sInt`

The integer sequences in a numpy array as given by the output of *addEmptyPositions*. All sequences must have the same length.

Value
Returns `IM`, a 4D numpy array with the images corresponding to the sequences. The first dimension corresponds to the number of sequences, the second to the height, the third to the width and the fourth to the channel (which is always 1 with this encoding).

trainCNN

Description
Train an ensemble convolutional neural network model. The base model consists of a *Conv2D* layer with *ReLu* activation, a *MaxPooling2D* layer and a *Dense* (or fully connected) layer. The parameters are defined in the parameter file (Appendix B.2.2).
For more information on the model's architecture, see Section 2.4.2.
If the class contains an attribute *trainedModels*, then these previously trained models will be used for transfer learning (see Section 2.4.5).

Usage

```
CNNPepPred.trainCNN(IM,out,saveModel=False)
```

Arguments

`IM`

The training images, as given by the output of *getImages*.

`out`

The training outcome.

`saveModels`

Whether or not to save the trained model as an attribute and in the saving pathway *savePath* of the class. If *saveModels* is true, the computation time of the training will be an attribute of the class called *timeTrain*. A folder called *model_[allele]* (where [allele] is the allele name of the class) will be created, it will contain the parameter file of the model and a folder called *nets* where the trained nets will be saved.

Value
Returns `models`, a list containing all the Keras trained models.

applyCNN

Description
Apply the trained model.

Usage

```
CNNPepPred.applyCNN(models,IM,saveOutcome=False)
```

Arguments

`models`

The ensemble model as given by the output of *trainCNN*.

`IM`

The images on which the trained model will be applied.

`saveOutcome`

Whether or not to save the predicted outcome as an attribute.

Value
Returns `yhat`, a numpy array with the predicted outcome of each sample.

crossValCNN

Description
Perform the training in a cross-validation set up. The computation time of the cross-validation will be an attribute of the class called *timeCV*.
If the training is performed with transfer learning (see Section 2.4.5), the peptides containing shared l-mers (where l is the parameter determining the length of the core binder, 9 by default) with the pre-trained model used for transfer learning are removed from the test set but are kept for training.

Usage

```
CNNPepPred.crossValCNN(IM,out)
```

Arguments

`IM`

The training images for the cross-validation.

`out`

The training outcome.

Value
Returns `yhatCV`, a numpy array containing the cross-validated predicted outcome of each sample and `modelCV`, a list containing the trained Keras models (as returned by *trainCNN* for each fold.

feedForwardAndGetScore

Description
Apply the trained model of the class to new sequences and get the score for each of the overlapping l-mers of a sequence, where l is the parameter determining the length of the core binder (9 by default).
To control the memory usage, the application of the sequences will be by batches of *maxNbSamples2apply*, which is a parameter (see Appendix B.2.2) with default value 50000.
For more information on the contribution score, see Section 2.4.4.

Usage

```
CNNPepPred.feedForwardAndGetScore(seq,saveOutcome=False)
```

Arguments

`seq`

The sequences on which the trained model will be applied as given by the output of *addEmptyPositions*.

`saveOutcome`

Whether or not to save the predicted outcome as an attribute. If *saveOutcome* is true, the computation time to apply the model on the data will be an attribute of the class called *timeApply*.

Value
Returns `contributionScore`, a numpy array with the constribution score of all the overlapping l-mers of each sequence and `yhat`, a numpy array with the predicted outcome of each sequence.

generateRandomSeq

Description
Generate integer random sequences. The number of random sequences to generate is set in the parameter file (Appendix B.2.2).

Usage

`CNNPepPred.generateRandomSeq(followLengthDistr=False)`

Arguments

`followLengthDistr`

If `False`, all the random sequences will have the same length *lengthRandSeq* as given in the parameter file (Appendix B.2.2). If `True`, the length distribution of the random sequences will follow the length distribution of the training data saved as an attribute called *seqL* with the function *seqLength*

Value
Returns `sR`, a list with the randomly generated integer sequences.

plotLogoSeq

Description
Generate a logo plot (using the package *logomaker*) of the highest scoring core binders. The plot will be saved in the pathway *savePath* of the class. The number of best scoring sequences used in the logo plot is set in the parameter file (Appendix B.2.2).

Usage

`CNNPepPred.plotLogoSeq(contributionScore,yhatR)`

Arguments

`contributionScore`

The contribution score of each overlapping l-mer in all of the sequences to which the trained model has been applied, as given by the output of *feedForwardAndGetScore*.

`yhatR`

The predicted score of each sequence.

Value
Returns `h`, the plot handle of the logo plot; `sBchar`, a list with the amino-acid sequences used to generate the plot and `pim`, the information matrix corresponding to the logo plot.

computationTime

Description
Save the computation time as an attribute called *timeTotal*.

Usage

```
CNNPepPred.computationTime(time_elapsed)
```

Arguments

`time_elapsed`

The elapsed time to save.

getCVresults

Description
Get the cross-validation results. The scores are: PC (Pearson correlation), AUC (area under the curve), RMSE (root mean square error), MCC (Matthews correlation coefficient), ACC (accuracy), BACC (balanced accuracy), F1 (F1-score). The result will be saved as a txt file *'cross_validation_results.txt* in the path *savePath*.

Usage

```
CNNPepPred.getCVresults()
```

printApplyOutcome

Description
Print the predicted outcome of the analysed sequences as a txt file *[allele]_predictedOutcome.txt* where [allele] is the allele name. The file will be saved in the path *savePath*. Note that only unique core binders will be

printed; if there are different peptides with the same core, the one with the highest predicted outcome will be printed.

Usage

`CNNPepPred.printApplyOutcome(saveTable = False)`

Arguments

`saveTable`

Whether or not to save the output table as an attribute.

Value
Returns `table`, a pandas data frame with the predicted outcome of the sequences on which a trained model was applied. The table only contains unique core binders.

seq2Lmer

Description
Cut sequences into all overlapping *epitopesLength*-mers, where *epitopesLength* is as given in the template (Appendix B.2.3).

Usage

```
CNNPepPred.seq2Lmer(seq,nameSeq=None,takeUniqueLmer=True,
   saveLmer=False)
```

Arguments

`seq`

The integer amino-acid sequences in a list of numpy arrays.

`nameSeq`

The name of the sequences.

`takeUniqueLmer`

Whether or not to select only the unique overlapping *epitopesLength*-mers.

`saveLmer`

Whether or not to save the output sequences as an attribute.

Value
Returns `sLmer`, a list with all overlapping *epitopesLength*-mers as integers; `nameSeqLmer`, the name of the sequences each element of `sLmer` belongs to and `indLmer`, the indices of the sequences each element of `sLmer` belongs to.

getCoreBinder

Description
Get the core binders of the sequences.

Usage

```
CNNPepPred.getCoreBinder(seq,contributionScore,applyDataName=
   None,saveCoreBinders=False)
```

Arguments

`seq`

The amino-acid sequences in a list. The sequences must all have the same length, i.e. use *int2aa* on the output of *addEmptyPositions*.

`contributionScore`

The contribution score of each overlapping *l*-mer in all of the sequences to which the trained model has been applied, as given by the output of *feedForwardAndGetScore*.

`applyDataName`

The name of the sequences.

`saveCoreBinders`

Whether or not to save the core binders as an attribute.

Value
Returns `sCore`, a numpy array with the core binder of each sequence (as amino acids).

save_object

Description
Save with *pickle* the object class. It will be saved in the path *savePath* with the file name given as argument or by default *[allele]_ModelCNN.pkl*, where [allele] is the allele name.
In order to avoid loading problems if the object is loaded from another OS, the attribute *savePath* is deleted upon saving.
If the class contains a list of trained Keras neural networks, it will be deleted as these nets are saved separately with the saving option of *trainCNN*.

Usage

```
CNNPepPred.save_object(name=None)
```

Arguments

`name`

Name of the file.

load_object

Description
Load another object class. This is meant to load previously trained models. As the attribute *savePath* is deleted upon saving (see *save_object*), this function will reset it to be the parent directory of the argument *filename.*

Usage

```
CNNPepPred.load_object(filename)
```

Arguments

`filename`

Complete pathway to the object to load.

Value
Returns `obj`, the loaded object.

feedForwardVisualization

Description
Visualization of the feed-forward pass of the trained model on the set of se-

quences *s*. It will create a folder in the path *savePath* called *feed_forward_visualization* that will contain two folders: *nets* and *sequences*. The folder *nets* will contain a folder for each net of the trained model with each of its corresponding convolutional layer's filters and dense layer's weights represented as images. The folder *sequences* will contain one folder for each of the input argument sequences with an image of their encoding and a folder for each net containing the convolutional layer's output and the maxpooling layer's output represented as images.
For each input sequence, many images will be saved; it is therefore recommended to only run this function on a small pre-selected set of sequences.
For more information on the visualization of the feed-forward pass, see Section 2.4.3.

Usage

`CNNPepPred.feedForwardVisualization(s,fontSize=4,dpi=300)`

Arguments

`s`

The amino acid sequences in a list.

`fontSize`

The font size of the x and y tick labels. Default is 4.

`dpi`

The dpi of the images. Default is 300.

Value
Returns `yhat`, a numpy array with the predicted outcome of each sequence.

generateCVpartWithLeastLmerOverlap

Description
Generate a cross-validation partition for the training data such that the number of shared *l*-mers between folds is reduced, where *l* is the length of the core binders as given in the parameter file (Appendix B.2.2).
For more information on the way the partition is generated, see Section 2.5.1.2.

Usage

CNNPepPred.`generateCVpartWithLeastLmerOverlap(kFold,saveCVPart=False)`

Arguments

`kFold`

The number of folds, as an integer.

`saveCVPart`

Whether or not to save the cross-validation partition as an attribute of the class called *cvPart*. If true, the average number of shared l-mers between each of the `kFold` train/test partitions (within each positive and negative class) will also be saved as an attribute of the class called *averageLmersOverlappingCV*.

Value
Returns `cvPart`, a numpy array with the cross-validation partition and `averageLmersOverlappingCV`, the average number of shared l-mers between each of the `kFold` train/test partitions (within each positive and negative class).

B.2.2 The parameter file

When initializing the class, the parameters will be set from the file given with full path in the template or, by default, the file in the working directory called *parameters.txt*. This file consists of two columns (separated by a comma), one with the name of the parameter and one with the value of the parameter. Only the parameter values can be changed if needed. If a parameter value is left empty, the default value will be set (if left empty, check that the comma separating the columns is still there). The parameters are the following.

- *bindingThr*. Default: 0.5.
  The binding threshold for the predicted values.

- *similarityMat*. Default: blosum62.txt
  The similarity matrix to use for the sequence encoding. It must be symmetric and be of the same format, with the same amino-acid order, as the default file.

- *l*. Default: 9
  The length of the core binder.

- *maxNbSamples2apply*. Default: 50000
  The maximum number of sequences on which a trained model can be applied in one batch. This is only for the application of the model through the function *feedForwardAndGetScore*. Increase if you have enough memory and decrease if you don't have enough memory.

- *nbPrev*. Default: 2
  The number of empty positions (corresponding to the absence of amino acids) to add at the beginning of a sequence.

- *nbAfter*. Default: 2
  The number of empty positions (corresponding to the absence of amino acids) to add at the end of a sequence.

- *F*. Default: 5/10/20/30
  The number of filters of the convolutional layer. Different number of filters can be given, separated by a slash "/". In that case the final model will be an -equally weighted- ensemble of models with different number of filters.

- *rep*. Default: 10
  The number of models to train with different initial weights per number of filters. For each number of filters given in the parameter *F*, *rep* number of models will be trained. The final model will be an equally weighted ensemble of *rep* times the number of different number of filters, i.e. $40 = 10 \cdot 4$ with the default parameters.

- *nMaxPool*. Default: see Section 2.4.2.
  The pooling size of the Maxpooling layer will be $nMaxPool \times 1$. The default value is set by a formula given in the Section 2.4.2 and will be such that the output layer has size $L_{freq} \times F$ where L_{freq} is the most frequent sequence length in the training data set and F is the number of filters.

- *initializeStd*. Default: 0.01
  The standard deviation of the initial weights (randomly generated from the normal distribution with zero mean). The same value will be used for the convolutional and the dense layers.

- *alpha*. Default: 0.005
  The learning rate of the stochastic gradient descent.

If transfer learning is used for training, two different values can be given (one for each optimization step of the training, see Section 2.4.5). The two values must be separated by a slash "/", for example "0.005/0.001". If only one value is given, it will be used for both optimization steps.

- *gamma*. Default: 0.9
  The momentum of the stochastic gradient descent.
  If transfer learning is used for training, two different values can be given (one for each optimization step of the training, see Section 2.4.5). The two values must be separated by a slash "/", for example "0.9/0.5". If only one value is given, it will be used for both optimization steps.

- *l2_fact*. Default: 0.0001
  The L2 regularization factor. The same value will be used for the convolutional and the dense layers.

- *maxEpochs*. Default: 30
  The number of epochs.
  If transfer learning is used for training, two different values can be given (one for each optimization step of the training, see Section 2.4.5). The two values must be separated by a slash "/", for example "20/10". If only one value is given, it will be used for both optimization steps. If 0 is given for the second value (i.e. "20/0"), the second optimization will be skipped.

- *miniBatchSize*. Default: 128
  The size of the mini batch for the stochastic gradient descent.
  If transfer learning is used for training, two different values can be given (one for each optimization step of the training, see Section 2.4.5). The two values must be separated by a slash "/", for example "128/56". If only one value is given, it will be used for both optimization steps.

- *useBias*. Default: 1
  Whether or not to use bias. The same value will be used for the convolutional and the dense layers.

- *activationFctDenseLayer*. Default: linear
  The activation function of the last layer (the *Dense* layer). Possible values are to choose among *keras*'s activation functions.

- *lossFct*. Default: mean_squared_error
  The loss function. Be aware that if changed, some parameter tuning might be needed. For example if for a classification problem you would rather use the *binary_crossentropy* loss function, you should change the activation function of the *Dense* layer to be the *sigmoid* function. Possible values are to choose among *keras*'s loss functions.

- *nbRandSeq*. Default: 200000
  The number of random sequences to be generated in the function *generateRandomSeq* (Appendix B.2.6).

- *nbBest*. Default: 2000
  The number of best scoring sequences to select for the generation of the logo plot with *plotLogoSeq*

- *lengthRandSeq*. Default: 15
  The length of the random sequences generated in the function *generateRandomSeq* (Appendix B.2.6).

B.2.3 The template file

Fill the template file given in the main folder *CNN-PepPred* according to the desired analysis. This template consists of two columns (separated by a comma), one with the name of the template's inputs and one with their values. Only the input values can be changed if needed. If an input value is left empty, the default value will be set (if left empty, check that the comma separating the columns is still there). The inputs are the following.

- *allele*.
  The name of the allele. This name can be thought of as a job name for the run. If the training option is not selected and no trained model is given as input, then *allele* corresponds to the name of a pre-trained model (Appendix B.2.5).

- *savePath*. Default: os.getcwd()
  The pathway where to save the results.

- *doTraining*. Default: 0
  Whether or not to do the training.

- *trainingDataPath*. Default: None
  The file with the training data. It must be a *.txt* file, with at least two columns (with headers) separated by a comma. The first column contains the sequences and the second the outcome. For regression, the outcome must be already normalized. A third column containing a cross-validation partition can be added. If the cross-validation option is selected and no partition is given here, it will be generated following the function *generateCVpartWithLeastLmerOverlap*.
  If training data are given and a previously trained model is given in the template as *trainedModelsFile*, transfer learning will be used for training (see Section 2.4.5).

- *doLogoSeq*. Default: 0
  Whether or not to plot (logo plot) the core binding pattern of the trained model.

- *doCV*. Default: 0
  Whether or not to do the cross-validation.

- *kFold*. Default: 5
  The number of folds for the cross-validation. If a partition is given in the training data file, this input will be ignored and the *kFold* value will be the number of partitions.

- *doApplyData*. Default: 0
  Whether or not to apply the trained model to new sequences.

- *trainedModelsFile*. Default: None
  Either the file containing the trained model (a *.pkl* file) or the pathway of the folder containing the *parameters* file and the *nets* folder with the trained nets (as saved with the function *trainCNN*. If the input is a *.pkl* file, the parent folder must contain the *nets* folder. This option is only valid if no training is selected.
  If the apply or the logoseq option are selected with no training and *trainedModelsFile* is left empty, then a pre-trained model will be selected based on the *allele*. For available alleles, see Appendix B.2.5.
  If a previously trained model is given and training data are given in the template as *trainingDataPath*, transfer learning will be used for training (see Section 2.4.5).
  If a previously trained model is given as input, only the values of

nbRandSeq, *nbBest*, *lengthRandSeq* and *maxNbSamples2apply* of the parameter file *parametersFile* given in the template will be considered. If transfer learning is used for training, the values of *alpha*, *gamma*, *maxEpochs*, *miniBatchSize*, *activationFctDenseLayer*, *lossFct*, *initializeStd* will also be considered. The remaining parameter's values will be taken from the parameter file of the previously trained model.

- *applyDataPath*. Default: None
  The file containing the data on which the trained model will be applied. It must be a FASTA file.

- *epitopesLength*. Default: 15
  The length of the epitopes on which the trained model will be applied. Each new sequence will be cut into all overlapping *epitopesLength*-mers and a prediction will be made for each of them.

- *parametersFile*. Default: parameters.txt (in the working directory)
  The full path for the file containing the parameters of the model. Parts of this file are ignored if a trained model is given as input in *trainedModelsFile*.

- *saveClassObject*. Default: 0
  Whether or not to save the class generated following the template in *savePath*. If the class contains a list of trained Keras neural networks, it will be deleted as these nets are saved separately with the saving option of *trainCNN*.

B.2.4 The script *model_from_template.py*

The argument of the script *model_from_template.py* is the template file. By default this file is called *template.txt* and is located in the working directory, the name and pathway can be modified but need to be given with full path as a system argument.
The script will first read the system argument to obtain the name of the template and call the main function with this template as an argument.

```
tmplName = sys.argv
if len(tmplName)==1:
   tmplName = 'template.txt'
else:
    tmplName = tmplName[1]
main(tmplName)
```

The *main* function will run the desired analysis following the template. First, the start time is recorded and the template is read,

```
time_start = time.perf_counter()
file = Path(tmplName)
allele,savePath,doTraining,trainingData,trainingOutcome,
    doLogoSeq,doCV,cvPart,kFold,doApplyData,trainedModelsFile,
    applyData,applyDataName,epitopesLength,parametersFile,
    saveClassObject = readTemplate(file)
```

then, the class *CNNPepPred* is initialized

```
modelCNN = CNNPepPred(allele,savePath,doTraining,trainingData,
    trainingOutcome,doLogoSeq,doCV,cvPart,kFold,doApplyData,
    trainedModelsFile,applyData,applyDataName,epitopesLength,
    parametersFile)
```

and the desired analysis will be performed following the template. If the training option is selected, the images *IM* encoding the sequences and training outcome *out* are first retrieved.

```
sInt = modelCNN.aa2int(modelCNN.trainingData)
modelCNN.seqLength(sInt,saveOutput=True)
sInt = modelCNN.addEmptyPositions(sInt)
IM = modelCNN.getImages(sInt)
out = modelCNN.trainingOutcome
```

Cross-validation with the training data is performed as follows:

```
modelCNN.crossValCNN(IM,out)
modelCNN.getCVresults()
```

The final model, to be saved in the object *modelCNN*, will be trained with all of the training data.

```
modelCNN.trainCNN(IM,out,saveModel=True)
```

To obtain the logoplot with the binding core, the script generates random sequences,

```
sR = modelCNN.generateRandomSeq()
```

applies the model to obtain the predicted outcomes and contribution scores of the random sequences' overlapping *modelCNN.l*-mers

```
contributionScore,yhatR = modelCNN.feedForwardAndGetScore(sR)
```

and finally generates the logoplot.

```
modelCNN.plotLogoSeq(contributionScore,yhatR)
```

The sequences on which the trained model must be applied are first cut into all the overlapping *epitopesLength*-mers.

```
sIntApply,sApplyName = modelCNN.seq2Lmer(modelCNN.aa2int(
   modelCNN.applyData),L=None,nameSeq=modelCNN.applyDataName,
   saveLmer = True)[0:2]
```

Then the amino-acid sequences are prepared in the required format for the application of the trained model.

```
sIntApply = modelCNN.addEmptyPositions(sIntApply)
```

The trained model is then applied to obtain the predicted outcomes and the contribution scores, which are used to find the binding cores, and the results are printed in the saving pathway.

```
modelCNN.feedForwardAndGetScore(sIntApply,saveOutcome = True)
modelCNN.getCoreBinder(modelCNN.int2aa(sIntApply),modelCNN.
   contributionScore,sApplyName,saveCoreBinders = True)
modelCNN.printApplyOutcome()
```

Finally the computation time is saved in the object and the object is saved in the saving pathway if selected in the template.

```
time_elapsed = (time.perf_counter() - time_start)
modelCNN.computationTime(time_elapsed)
if saveClassObject:
   modelCNN.save_object()
```

B.2.5 The pre-trained models

The user can use models available for some alleles which were trained with IEDB data (Section 2.5.1.1). The models are in a folder called *trainedIEDB-models* of the main directory *CNN-PepPred*. In this case, the template must contain the name of the allele and the data to apply the model to; no training

must be selected and the trained model file (*trainedModelsFile*) must be left empty. An example template called *template_pretrained_model_example.txt* in the main directory has been pre-filled for the allele *HLA_DRB1_01_01*. The location where to save the results and the fasta file on which to apply the pre-trained model must be filled (*[your_path_to_save_the_results]* and *[fasta_file_for_prediction]* in the example template).
Available alleles are:

HLA_DPA1_01_03_DPB1_02_01,	HLA_DPA1_01_03_DPB1_03_01,	HLA_DPA1_01_03_DPB1_04_01,
HLA_DPA1_01_03_DPB1_04_02,	HLA_DPA1_01_03_DPB1_06_01,	HLA_DPA1_01_03_DPB1_104_01,
HLA_DPA1_02_01_DPB1_01_01,	HLA_DPA1_02_01_DPB1_09_01,	HLA_DPA1_02_01_DPB1_10_01,
HLA_DPA1_02_01_DPB1_14_01,	HLA_DPA1_02_01_DPB1_17_01,	HLA_DPA1_02_01__DPB1_13_01,
HLA_DPA1_02_02_DPB1_05_01,	HLA_DQA1_01_01_DQB1_05_01,	HLA_DQA1_01_02_DQB1_05_01,
HLA_DQA1_01_02_DQB1_06_02,	HLA_DQA1_02_01_DQB1_02_02,	HLA_DQA1_02_01_DQB1_03_01,
HLA_DQA1_03_01_DQB1_03_02,	HLA_DQA1_03_02_DQB1_04_01,	HLA_DQA1_05_01_DQB1_02_01,
HLA_DQA1_05_01_DQB1_03_01,	HLA_DQA1_05_05_DQB1_03_01,	HLA_DRB1_01_01,
HLA_DRB1_03_01,	HLA_DRB1_04_01,	HLA_DRB1_04_02,
HLA_DRB1_04_04,	HLA_DRB1_04_05,	HLA_DRB1_07_01,
HLA_DRB1_08_01,	HLA_DRB1_08_02,	HLA_DRB1_09_01,
HLA_DRB1_10_01,	HLA_DRB1_11_01,	HLA_DRB1_11_03,
HLA_DRB1_12_01,	HLA_DRB1_13_01,	HLA_DRB1_13_02,
HLA_DRB1_13_03,	HLA_DRB1_14_01,	HLA_DRB1_14_54,
HLA_DRB1_15_01,	HLA_DRB1_16_01,	HLA_DRB3_01_01,
HLA_DRB3_02_02,	HLA_DRB3_03_01,	HLA_DRB4_01_01,
HLA_DRB4_01_03,	HLA_DRB5_01_01,	HLA_DRB5_02_02.

B.2.6 Random generation of non-binders

The majority of experimental results only report binding peptides, so that most sets are too imbalanced to properly train a model. Therefore, we provide a separate script for the generation of randomly selected peptides that act as non-binders.
The script will simply select peptides at random from a user given folder containing fasta files, respecting the length distribution of the binders in the training set. These files should contain enough natural random sequences so that there shouldn't be any patterns that would relate them to one another (e.g. a full proteome).
The script is in the main folder *CNN-PepPred*, it is called *generateRandom-*

NonBinders.py and contain a unique function with the same name. Therefore, to import it, use

```
from generateRandomNonBinders import generateRandomNonBinders
```

The function is

```
generateRandomNonBinders(fastaSeqLoc,seqL=None,seq=None,prop=1,
   N=None,maxFiles=None)
```

with arguments:

`fastaSeqLoc`

The location of the folder containing the fasta files to select from.

`seqL`

A numpy array with the lengths of the binding peptides in the training set.

`seq`

A list of amino-acid sequences corresponding to the binding peptides in the training set. If `seqL` is not given, it will be computed from this list. If `seqL` is given, this argument is ignored.

`prop`

The proportion of peptides to select. The number of selected peptides will be around `prop·N` where `N` is either the number of binding peptides or the argument `N`.
`prop` is 1 by default.

`N`

The number of peptides to select. The final number will be `prop·N`. Note that due to the nature of the algorithm, it is possible that the number of peptides in the output differs slightly from this number.
If no sequences or length of sequences is given, `N` will be 2000 by default.

`maxFiles`

The maximum number of files to read in the given folder `fastaSeqLoc`.
We recommend dividing the sequences to select from into many files in `fastaSeqLoc` and using the parameter `maxFiles` instead of having one big file. In this way, the computational time will be lower since the algorithm will only read few smaller files rather than a big one and there won't be any memory issues.

The function will return `seqNeg`, a list of amino-acid sequences respecting the number of sequences and their length distribution according to the given input arguments.
The function can be called as follows, with `myfolderwithsequences` being the pathway to the folder containing the sequences to select from.

```
seqNeg = generateRandomNonBinders(myfolderwithsequencesaSeqLoc,
   seqL=bindersLength,prop=1.3,maxFiles=3)
```

In this case the output `seqNeg` will have around 1.3 times the number of elements in `bindersLength`, with lengths ditributed like in `bindersLength` and selected from 3 randomly selected fasta files in the folder `myfolderwithsequences` . On the other hand

```
seqNeg = generateRandomNonBinders(myfolderwithsequencesaSeqLoc,
   seq=bindersSeq,N=2500,maxFiles=1)
```

will return around 2500 peptides respecting the length distribution of the sequences in `bindersSeq` and randomly selected from 1 file in the folder `myfolderwithsequences`.

B.3 Examples

Three different templates were prepared as examples in the main folder *ModelCNN*. To use them, you will need to change the pathways in the templates adapting them to the operating system of your computer and replace *[your_working_path]* by the pathway of the folder *ModelCNN*.
To run the template files, set your working directory to *ModelCNN* and type in your console

```
import sys
model_from_template = open("model_from_template.py").read()
sys.argv = ['model_from_template.py','full_path_to_any_template
   .txt']
exec(model_from_template)
```

Or, alternatively,

```
import model_from_template
modelCNN = model_from_template.main('full_path_to_any_template.
   txt')
```

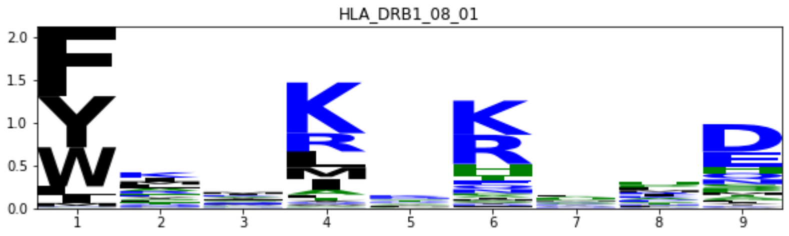


Figure B.1: Logo plot of the first template.

Template 1: Train+CV+logoPlot+Apply

The first template, *template1_Train_CV_logoPlot_Apply.txt*, will perform cross-validation and train a model using the example training data set of allele *HLA_DRB1_08_01* in the folder *Example*. It will also generate the logo plot representing the binding characteristics of the trained model and apply it to new sequences *uniprot-proteome_UP000000605_100.fasta* in the same folder. The results will be saved in the folder *Template1_results* of the *Example* folder.
Here are the cross-validation results obtained after runing this template (note that there might be small differences between runs):

```
Allele,#Peptide,#Binder,PC,AUC,RMSE,MCC,ACC,BACC,F1
HLA_DRB1_08_01
    ,1118,559,0.783,0.962,0.312,0.834,0.917,0.917,0.917
```

The different scores are: PC (Pearson correlation), AUC (area under the curve), RMSE (root mean square error), MCC (Matthews correlation coefficient), ACC (accuracy), BACC (balanced accuracy), F1 (F1-score).
Figure 1 contains the logo plot of the trained model:
Here is a list of some of the highest predicted binders:

```
Peptide_Source,Start,End,Peptide,Binding_Core,Predicted_Outcome
spQ63PT2SAHH_BURPS,168,182,EVALFKSIERHLEID,FKSIERHLE,1.454
spQ63Q03RPOB_BURPS,1069,1083,VKVYLAVKRRLQPGD,YLAVKRRLQ,1.392
spQ63UT2SYH_BURPS,346,360,REQAFIVAERLRDTG,FIVAERLRD,1.375
spQ63PT2SAHH_BURPS,103,117,GTPVFAFKGESLDEY,FAFKGESLD,1.371
spQ63NC4ACSA_BURPS,572,586,VVAFVVLKRSRPEGE,FVVLKRSRP,1.327
spQ63Y06SYR_BURPS,440,454,AVRFFLISRKADTEF,FFLISRKAD,1.303
spQ63WM0RS20_BURPS,28,42,FRTAIKAVRKAIDAG,IKAVRKAID,1.289
spQ63WM0RS20_BURPS,47,61,AAELFKAATKTIDTI,FKAATKTID,1.278
```

```
spQ63TM2SYT_BURPS,575,589,EKISYKIREHTLEKV,YKIREHTLE,1.236
spQ63UY6RS6_BURPS,85,99,LRHLIVKMKKAETGP,LIVKMKKAE,1.232
```

The first column is the name of the sequence, as written in the FASTA file. The second and third columns are respectively the start and end position of the peptide in the sequence. The fourth column is the peptide and the fifth column its binding core. The sixth column is the model's predicted outcome.

Template 2: Train

The second template, *template2_Train.txt*, will train a model using the example training data set of allele *HLA_DRB1_08_01* in the folder *Example*. The results will be saved in the folder *Template2_results* of the *Example* folder.

Template 3: Apply with template 2 trained model

The third template, *template3_Apply.txt*, applies the pre-trained model of *HLA_DRB1_08_01* to new sequences *uniprot-proteome_UP000000605_100seq.fasta* in the *Example* folder. The results will be saved in the folder *Template3_results* of the *Example* folder. Here is a list of some of the highest predicted binders:

```
Peptide_Source,Start,End,Peptide,Binding_Core,Predicted_Outcome
spQ63PT2SAHH_BURPS,168,182,EVALFKSIERHLEID,FKSIERHLE,1.449
spQ63UT2SYH_BURPS,346,360,REQAFIVAERLRDTG,FIVAERLRD,1.391
spQ63Q03RPOB_BURPS,1069,1083,VKVYLAVKRRLQPGD,LAVKRRLQP,1.369
spQ63PT2SAHH_BURPS,103,117,GTPVFAFKGESLDEY,FAFKGESLD,1.351
spQ63Y06SYR_BURPS,440,454,AVRFFLISRKADTEF,FFLISRKAD,1.327
spQ63WM0RS20_BURPS,29,43,RTAIKAVRKAIDAGD,IKAVRKAID,1.309
spQ63NC4ACSA_BURPS,572,586,VVAFVVLKRSRPEGE,FVVLKRSRP,1.305
spQ63T53ALLC1_BURPS,34,48,DDFFAPKERMLNPEP,FAPKERMLN,1.301
spQ63WM0RS20_BURPS,47,61,AAELFKAATKTIDTI,FKAATKTID,1.272
spQ63TM2SYT_BURPS,575,589,EKISYKIREHTLEKV,YKIREHTLE,1.269
```

Appendix C: Supplementary information for Chapter 3

C.1 List of variables extracted from the CRF

Table C.1 contains the variables extracted from the Case Report From (CRF) of the SICPAC study.

CRF variables

Patients' clinical data	**Co-treatment**	**AES**
Time (number of days between consecutive visits), Gender, BILITO, Basophils, Creatinine, Age, GOT, GPT, Metastasis, Neutrophiles, Potassium, ProtTot, Gender, Sodium, Lymphocyte, Red blood cells, Hemoglobin, Leukocytes, Eosinophil, Monocytes, Platelet	DB00016, DB00030, DB00085, DB00099, DB00158, DB00178, DB00186, DB00193, DB00207, DB00222, DB00230, DB00275, DB00295, DB00316, DB00331, DB00335, DB00338, DB00370, DB00391, DB00421, DB00425, DB00448, DB00451, DB00502, DB00563, DB00581, DB00584, DB00641, DB00678, DB00695, DB00706, DB00788, DB00813, DB00836, DB00863, DB00904, DB00945, DB00966, DB00999, DB01050, DB01076, DB01118, DB01183, DB01184, DB01225, DB01233, DB01234, DB01261, DB01306, DB01558, DB01591, DB01592, DB04817, DB04861, DB04876, DB06204, DB06723, DB06791, DB08810, DB09154, DB09300, DB11057, DB11742, DB11921, DB13157, DB13257, DB13679	ABDOMINAL_DISTENSION, ALOPECIA, ANEMIA, ANOREXIA, ANXIETY, ARTHRITIS, ASCITES, ASTHENIA, CACHEXIA, COLIC, CONFUSION, CONJUNCTIVITIS, CONSTIPATION, DIARRHEA, DRUG_HYPERSENSITIVITY, DYSGEUSIA, EDEMA, EXFOLIATIVE_DERMATITIS, FEVER, FLACCIDITY, HALLUCINATIONS, HEPATITIS, INFECTION, INFILTRATION, JAUNDICE, LEUKOPENIA, METASTASES, MUCOSITIS, NAUSEA, NEUROPATHY, NEUTROPENIA, NOTOXICIDAD, ONICOLISIS, ORAL_CANDIDIASIS, PAIN, PARESTHESIA, PLAQUETOPENIA, PLEURA, PNEUMONIA, PRURITUS, SEPSIS, SKIN_RASH, SOMNOLENCE, STEATORRHOEA, THROMBCYTOPENIA, THROMBOCYTOPENIA, THROMBOSIS, UTI, VOMITING, WEIGHT_LOSS

Table C.1: List of variables extracted from the CRF and used in the analysis. Variables are grouped by their origin in this table for the sake of simplicity. The drugs used as co-treatments are identified with their drug bank ids (DB prefix). The column AES corresponds to the adverse events recorded in at least one visit.

C.2 How to read the regression trees

The analysis performed in this study uses regression trees to determine the effect of all the variables measured, both numeric and categorical, on the outcome (counts of monocytes, eosinophils, red blood cells, platelets and leukocytes and concentration of hemoglobin). Decision trees allow us to select which variables have the greatest effect on the variable value and the criteria followed to calculate this value.
In Figure C.1, an example of a regression tree for the prediction of the upcoming visit's eosinophil count is shown. At the bottom level of the tree the final values of the variable analysed are displayed, while the rest of the nodes represent a variable and a cut-off value. In future predictions, this cut-off value will determine which way the prediction process will go through, depending on the variable found in the tree. This route is done from top to bottom, starting from the primary node at the top level of the tree. Taking Figure C.1, the first variable found is Neutropenia_2, which corresponds to the diagnosis of neutropenia at the penultimate visit, i.e., 2 visits before the objective upcoming visit. Being a categorical variable, it can only take values 0 and 1. As seen in the tree, if the patient suffers from neutropenic condition, prediction will continue on the right side, while the left side will be used if the patient does not show any neutropenic condition. The same procedure is followed at every node level on the tree. When a numeric variable is found in the tree, its cut-off value is used the same way as any categorical values. All numerical variables are normalized by subtracting the median of all variable values up to the last available visit of the patient. Note that the outcome (in this example, the eosinophil count) is also normalized, i.e., the values at the bottom of the trees are normalized. To obtain the unnormalized value, one needs to add the variable's median value which is patient and visit dependent. In Figure C.1 several time periods between visits are considered as significant to predict positive and negative trends with respect to the median of the eosinophil counts. Taking the variable Time_1 (time period between the actual and a future visit normalized by subtracting the median of all time periods), we can see that different trend predictions can be done. Values shown at the bottom level of the tree correspond to the increasing or decreasing tendencies compared to the median of the patient's outcome values. A negative/positive value will indicate a decrease/increase compared to the median value. Taking the same Time_1, if the normalized time period is less than 2.5 days, a slightly decreasing trend with respect to

the median on eosinophils count will be observed in this patient, based on the small negative value found at the final node (value=-0.03).

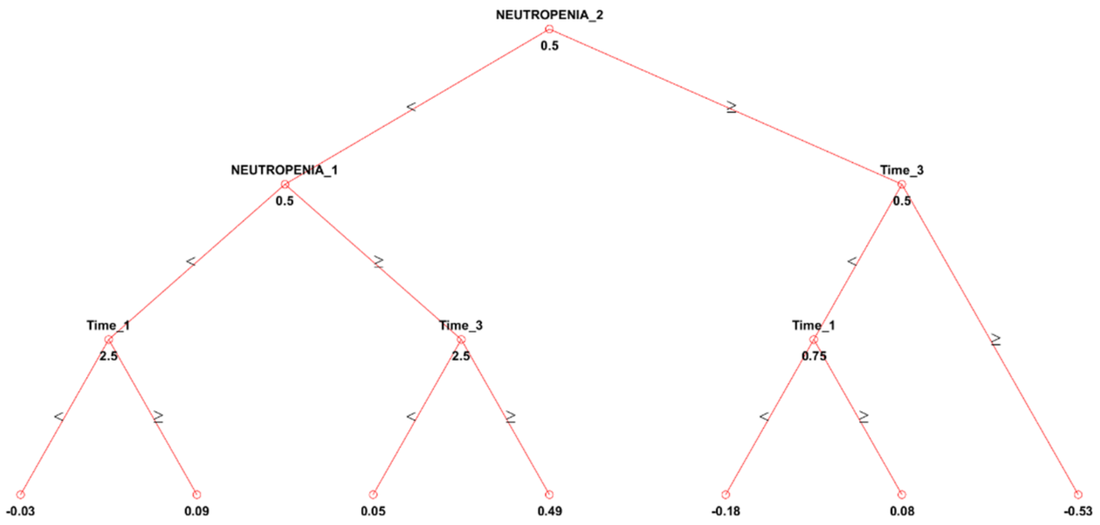


Figure C.1: Example of a regression tree, obtained for eosinophils using data from the CRF. The numerical values (as opposed to categorical 0/1 values) are normalized and therefore this tree is valid for all patients provided the patients' values are normalized too. In order to obtain the patient specific tree at a specific visit, one needs to compute the outcome and numerical variables' median values for the patient using all the values up to this specific visit and add these medians to the appropriate values displayed in the tree. For example, if the patient's median eosinophil count up to the last available visit is c, then a predicted normalized value of -0.03 corresponds to an unnormalized value of (-0.03)+c.

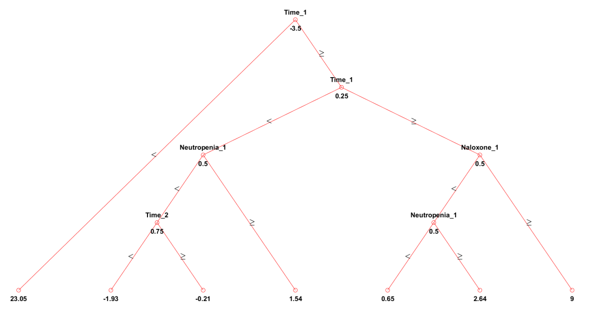
C.3 Trees from the models

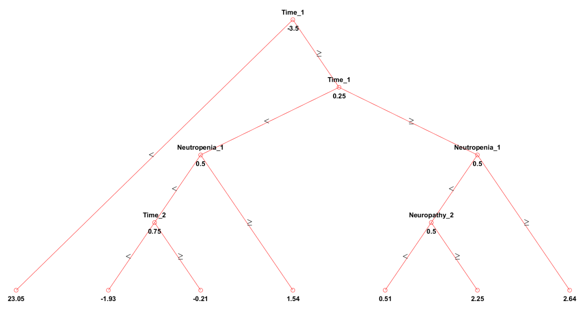
This section contains the regression trees from the models for each of the 6 outcomes trained with the full training data. It is divided in two parts; the first corresponds to the models trained only with the CRF variables while the second corresponds to the models trained with the CRF and the TPMS variables.
Note that the values of numerical (non-categorical) variables appearing in the trees are normalized and that this normalization depends on the patient and on the visit of this patient. As a recall, the normalization consists of

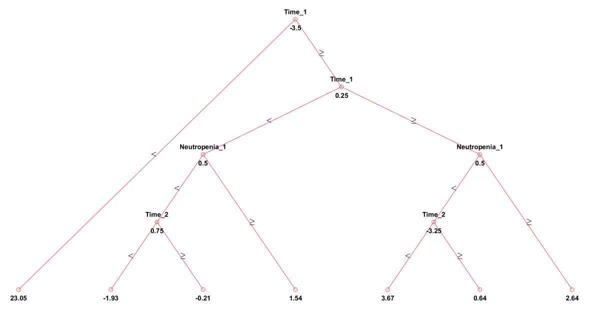
subtracting the patient's median value to the variables (where the median is computed using all of the patient's available visit values).
For each outcome, the final prediction is the equally weighted average of all of the model's single-tree predictions. It is possible for two or more trees from the same outcome to be the exact same. This means that the model has converged to the same solution multiple times. In such case, the repeated tree will weight more in the final ensemble.

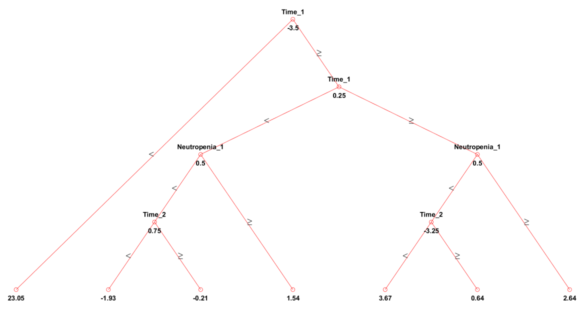
C.3.1 CRF variables

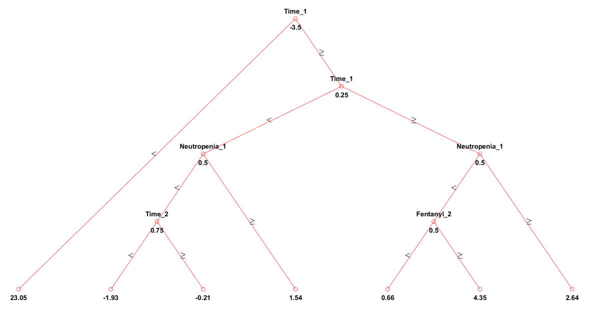
Leukocytes



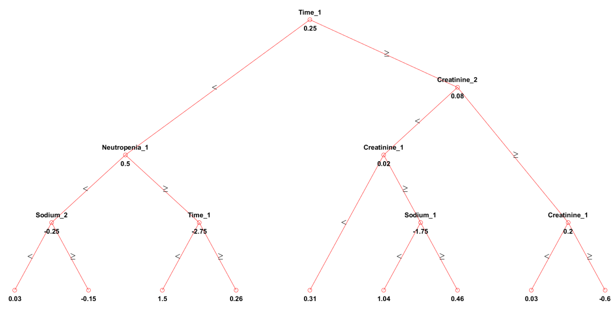


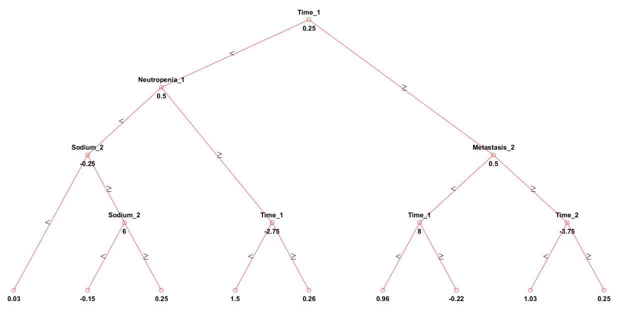




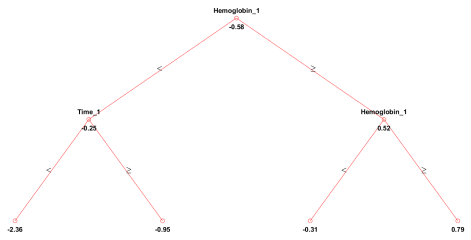


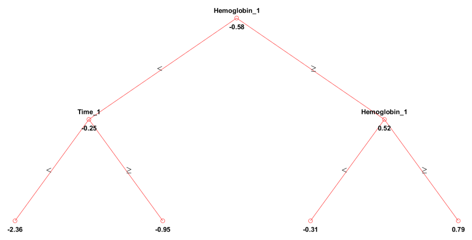
Monocytes

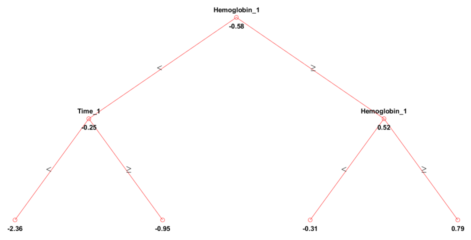

Time_1
0.25
Creatinine_2
0.08
Neutropenia_1
0.5
Creatinine_1
0.02
Sodium_2
-0.25
Time_1
-2.75
Sodium_1
-1.75
Creatinine_1
0.2
0.03
-0.15
1.5
0.26
0.31
1.04
0.46
0.03
-0.6

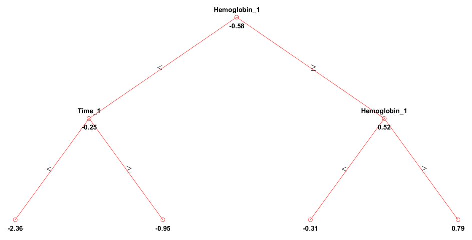

Time_1
0.25
Neutropenia_1
0.5
Metastasis_2
0.5
Sodium_2
-0.25
Sodium_2
6
Time_1
-2.75
Time_1
8
Time_2
-3.75
0.03
-0.15
0.25
1.5
0.26
0.96
-0.22
1.03
0.25

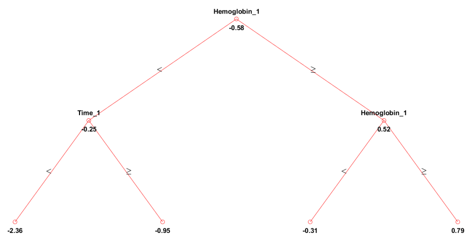
Hemoglobin


Hemoglobin_1
-0.58
Time_1
-0.25
Hemoglobin_1
0.52
-2.36
-0.95
-0.31
0.79

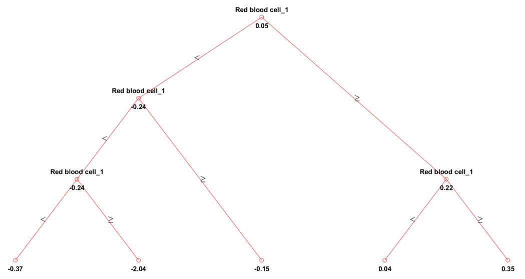

Hemoglobin_1
-0.58
Time_1
-0.25
Hemoglobin_1
0.52
-2.36
-0.95
-0.31
0.79

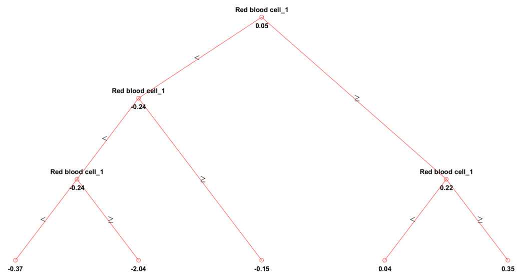

Hemoglobin_1
-0.58
Time_1
-0.25
Hemoglobin_1
0.52
-2.36
-0.95
-0.31
0.79

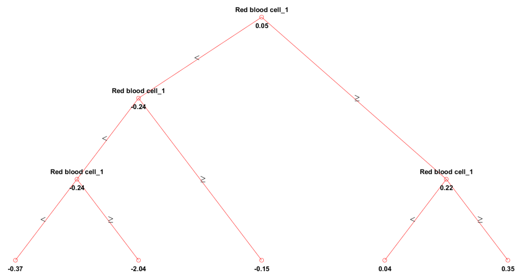

Hemoglobin_1
-0.58
Time_1
-0.25
Hemoglobin_1
0.52
-2.36
-0.95
-0.31
0.79

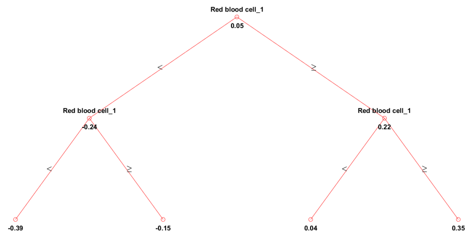

Hemoglobin_1
-0.58
Time_1
-0.25
Hemoglobin_1
0.52
-2.36
-0.95
-0.31
0.79

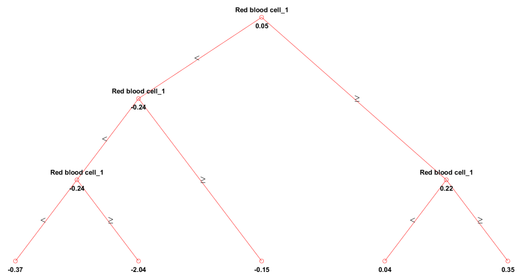
Red blood cells



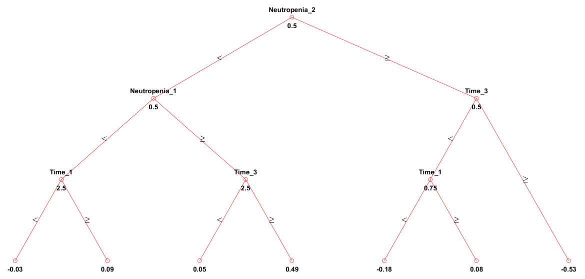




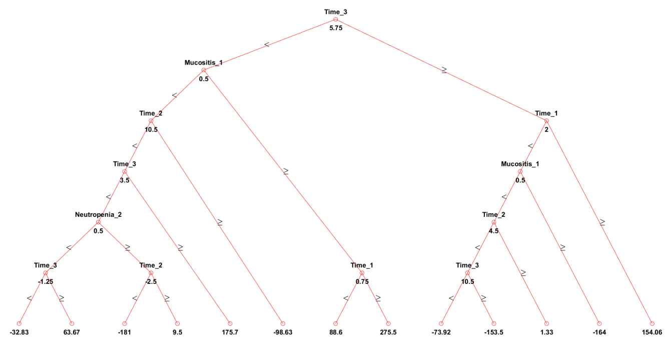


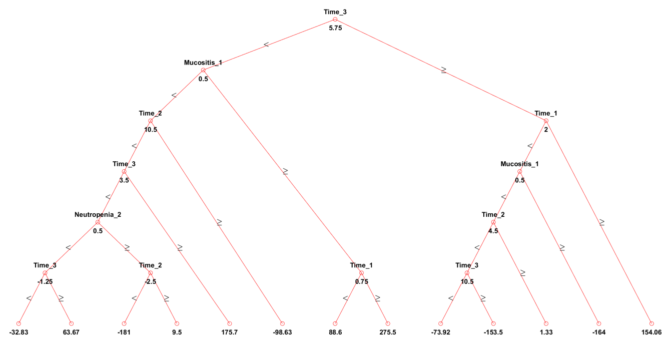


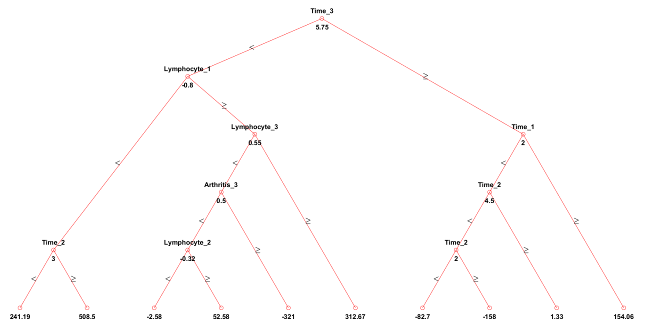
Eosinophils

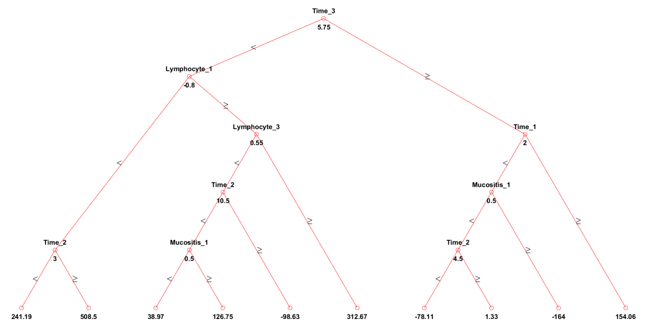


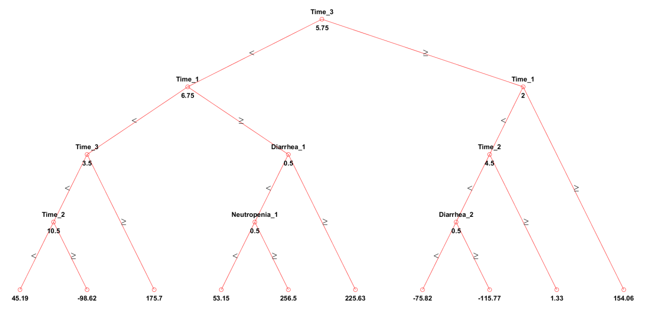
Platelets


Time_3
5.75
Mucositis_1
0.5
Time_1
2
Time_2
10.5
Mucositis_1
0.5
Time_3
3.5
Time_2
4.5
Neutropenia_2
0.5
Time_3
-1.25
Time_2
-2.5
Time_1
0.75
Time_3
10.5
-32.83
63.67
-181
9.5
175.7
-98.63
88.6
275.5
-73.92
-153.5
1.33
-164
154.06


Time_3
5.75
Mucositis_1
0.5
Time_1
2
Time_2
10.5
Mucositis_1
0.5
Time_3
3.5
Time_2
4.5
Neutropenia_2
0.5
Time_3
-1.25
Time_2
-2.5
Time_1
0.75
Time_3
10.5
-32.83
63.67
-181
9.5
175.7
-98.63
88.6
275.5
-73.92
-153.5
1.33
-164
154.06

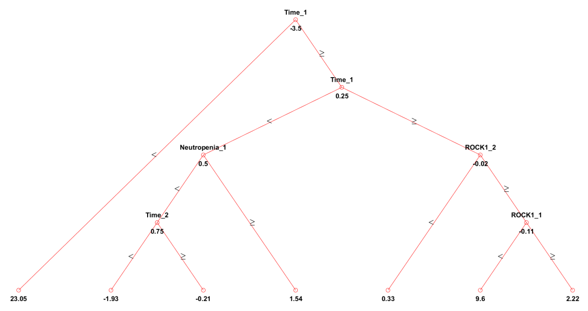

Time_3
5.75
Lymphocyte_1
-0.8
Time_1
2
Lymphocyte_3
0.55
Arthritis_3
0.5
Time_2
4.5
Time_2
3
Lymphocyte_2
-0.32
Time_2
2
241.19
508.5
-2.58
52.58
-321
312.67
-82.7
-158
1.33
154.06

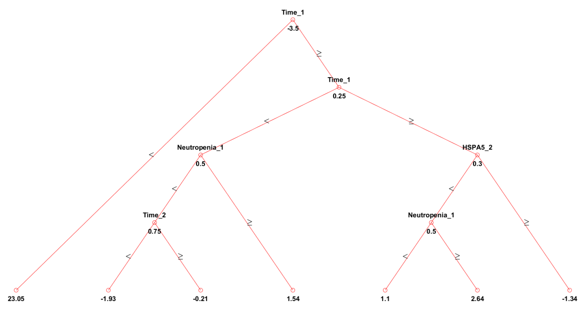

Time_3
5.75
Lymphocyte_1
-0.8
Time_1
2
Lymphocyte_3
0.55
Time_2
10.5
Mucositis_1
0.5
Time_2
3
Mucositis_1
0.5
Time_2
4.5
241.19
508.5
38.97
126.75
-98.63
312.67
-78.11
1.33
-164
154.06

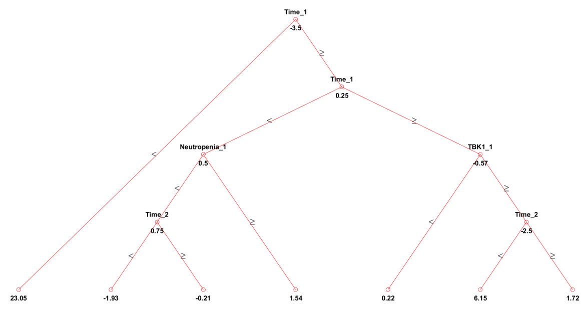

Time_3
5.75
Time_1
6.75
Time_1
2
Time_3
3.5
Diarrhea_1
0.5
Time_2
4.5
Time_2
10.5
Neutropenia_1
0.5
Diarrhea_2
0.5
45.19
-98.62
175.7
53.15
256.5
225.63
-75.82
-115.77
1.33
154.06

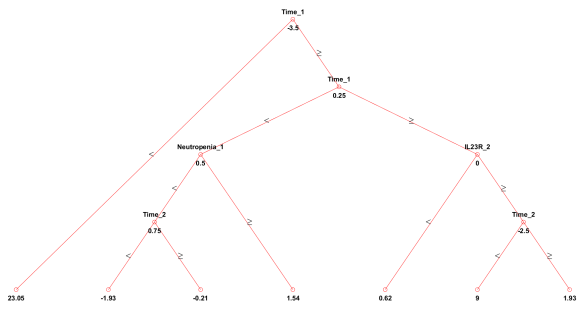
C.3.2 CRF and TPMS variables

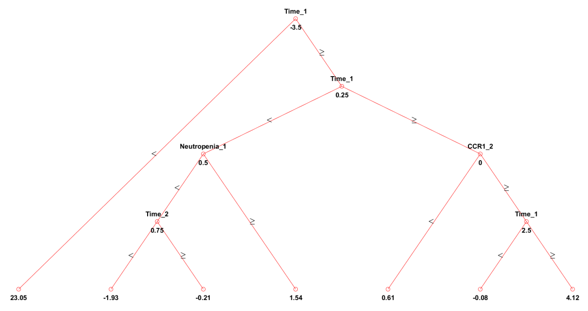
Leukocytes



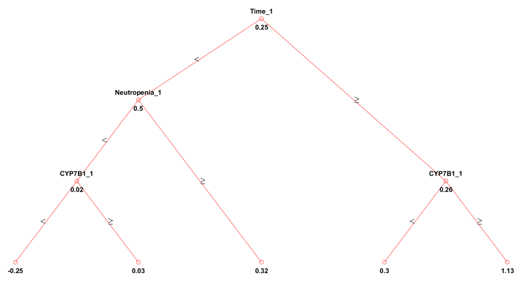


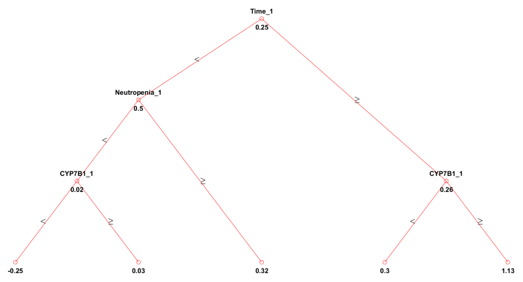




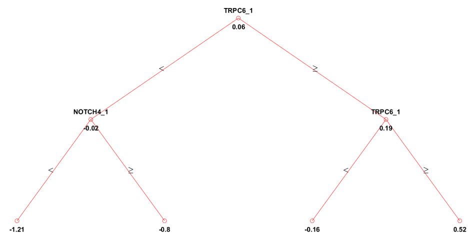


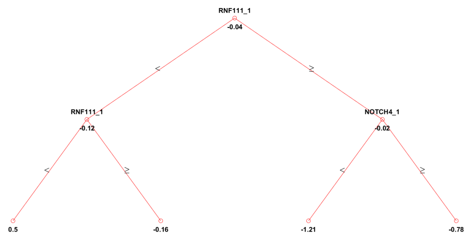
Monocytes

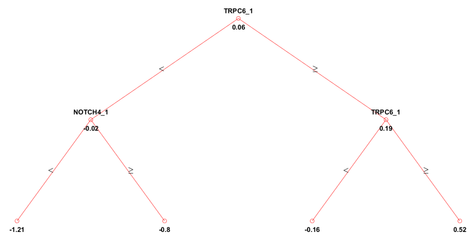

Time_1
0.25
<
>=
Neutropenia_1
0.5
CYP7B1_1
0.02
CYP7B1_1
0.26
-0.25
0.03
0.32
0.3
1.13

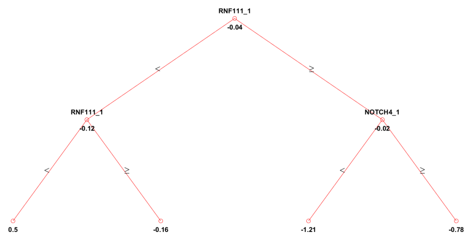

Time_1
0.25
<
>=
Neutropenia_1
0.5
CYP7B1_1
0.02
CYP7B1_1
0.26
-0.25
0.03
0.32
0.3
1.13

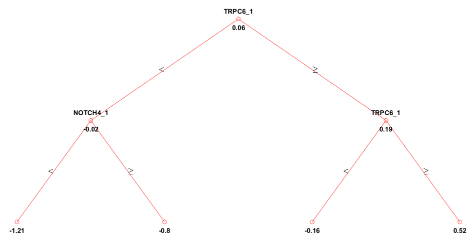
Hemoglobin


TRPC6_1
0.06
<
>=
NOTCH4_1
-0.02
TRPC6_1
0.19
-1.21
-0.8
-0.16
0.52

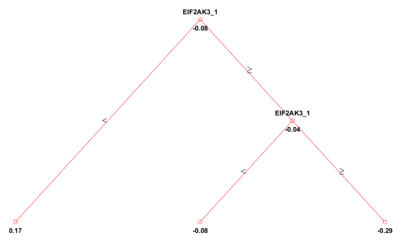

RNF111_1
-0.04
<
>=
RNF111_1
-0.12
NOTCH4_1
-0.02
0.5
-0.16
-1.21
-0.78

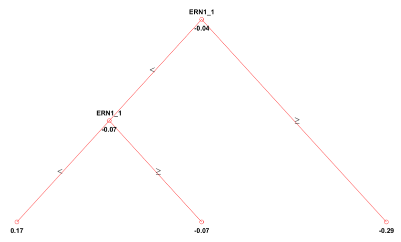

TRPC6_1
0.06
<
>=
NOTCH4_1
-0.02
TRPC6_1
0.19
-1.21
-0.8
-0.16
0.52


RNF111_1
-0.04
<
>=
RNF111_1
-0.12
NOTCH4_1
-0.02
0.5
-0.16
-1.21
-0.78

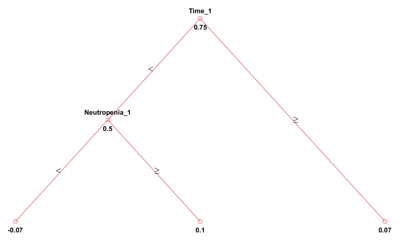

TRPC6_1
0.06
<
>=
NOTCH4_1
-0.02
TRPC6_1
0.19
-1.21
-0.8
-0.16
0.52

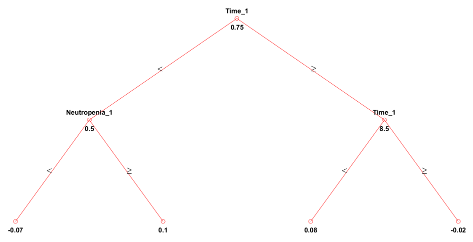
Red blood cells



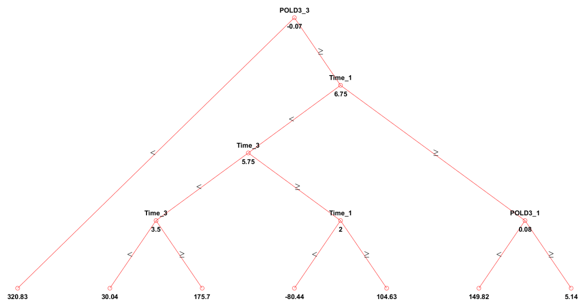


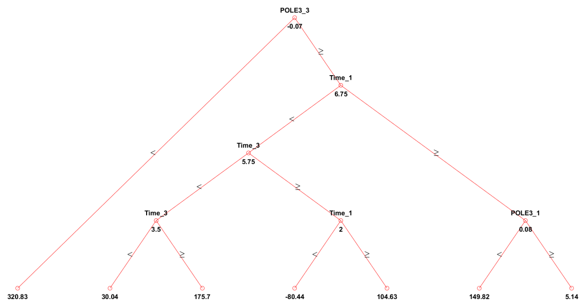
Eosinophils

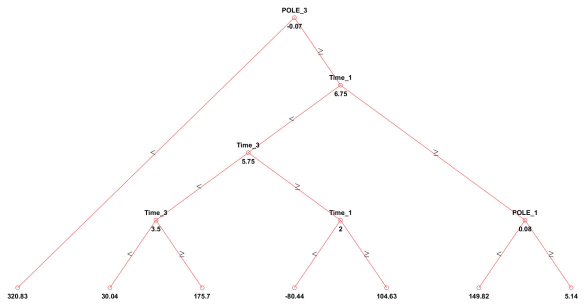


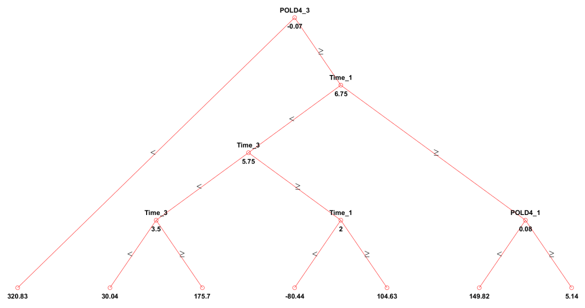


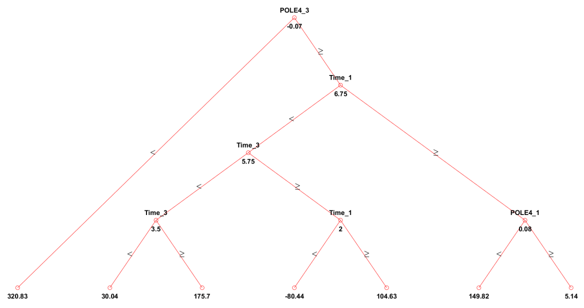
Platelets


POLD3_3
-0.07
Time_1
6.75
Time_3
5.75
Time_3
3.5
Time_1
2
POLD3_1
0.08
<
≥
320.83
30.04
175.7
-80.44
104.63
149.82
5.14


POLE3_3
-0.07
Time_1
6.75
Time_3
5.75
Time_3
3.5
Time_1
2
POLE3_1
0.08
<
≥
320.83
30.04
175.7
-80.44
104.63
149.82
5.14


POLE_3
-0.07
Time_1
6.75
Time_3
5.75
Time_3
3.5
Time_1
2
POLE_1
0.08
<
≥
320.83
30.04
175.7
-80.44
104.63
149.82
5.14


POLD4_3
-0.07
Time_1
6.75
Time_3
5.75
Time_3
3.5
Time_1
2
POLD4_1
0.08
<
≥
320.83
30.04
175.7
-80.44
104.63
149.82
5.14


POLE4_3
-0.07
Time_1
6.75
Time_3
5.75
Time_3
3.5
Time_1
2
POLE4_1
0.08
<
≥
320.83
30.04
175.7
-80.44
104.63
149.82
5.14