Politecnico di Torino

III Facoltà di Ingegneria

Bioinformatics

Project #6:
Classification of gene expression data

Master degree in Electronic Engineering

Authors:
Burco Adriele, s235919
Burrello Alessio, s238495
Cesarano Giuseppe, s235845

July 5, 2018

Contents

CHAPTER 1

Introduction

The breast cancer is a popular disease that affects 0.4% of a random population; the risk factors include oral contraceptive pill use, a history of ovarian cancer of the person under study, first relatives with breast cancer and hormone therapy [1]. Parker in [2] proposes a new risk model of the breast cancer based on the 50-gene classification: this incorporates the gene expression based intrinsic subtypes luminal A, luminal B, HER2-enriched, and basal-like. In the next studies the class Normal-like has been added to the classification [3, 4].
Let's give a breaf description of the different subtypes:

- **Luminal A** cancers are low-grade, tend to grow slowly and have the best prognosis.
- **Luminal B** cancers generally grow slightly faster than luminal A cancers and their prognosis is slightly worse.
- **Triple-negative/basal-like breast**: this type of cancer is more common in women with BRCA1 gene mutations. Researchers are not really sure why, but this type of cancer is also more common among younger and African-American women.
- **HER2-enriched** breast cancer is hormone-receptor negative (estrogen-receptor and progesterone-receptor negative) and HER2 positive. HER2-enriched cancers tend to grow faster than luminal cancers and can have a worse prognosis, but they are often successfully treated with targeted therapies aimed at the HER2 protein.
- **Normal-like** breast cancer is similar to luminal A disease: still, while normal-like breast cancer has a good prognosis, its prognosis is slightly worse than luminal A cancers prognosis.

1.1 Goal and set up

Our goal is the classification of the different subtypes of **breast cancer** starting from the *transcriptoma* of each single patient. The dataset is composed by 1046 patients divided between the five subtypes of breast cancer. In the figure 1.1 we can see the distributions of the patients in the different classes, noting the strong imbalance between them.

The starting point is to collect data about transcriptoma in order to obtain a dataset with labels indicating some of the possible breast cancer subtypes. After that, data have to be analyzed and modified in order to handle some problems about class imbalance and huge number of features that this dataset contains.
Once the dataset is organized in the best possible way, classification has to be performed, considering

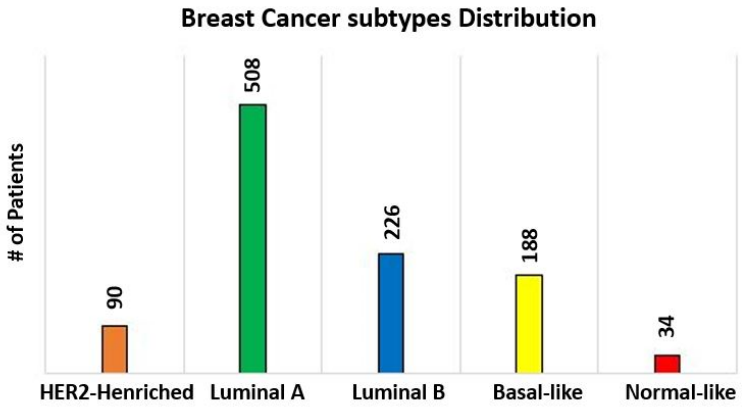

Figure 1.1: The distribution of the 1046 patients in the five different classes. It is clear from the graph that the distribution is not homogenous, with a class, i.e. Luminial A, with a very high diffusion (508) and a class, i.e. Normal-like, with only 34 cases.

both **supervised** and **unsupervised** machine learning techniques, in order to understand which approach can be the best one.

1.2 Methods

Python 2.7.14 has been used to handle all the analysis; MATLAB R2017b has been used to draw part of the graphs. Our algorithm is divided in 3 steps:

- Dimensionality reduction or feature selection: we implement two methods of feature reduction, taken from the **sklearn** library of python and we use the **genetic algorithm** as feature selector; feature reductions that use labels are applied only to supervised pipelines;
- Class imbalance management: here we take advantage of the **imblearn** library to handle classes with different number of samples. We mainly use techniques of oversampling, together with a cleaning of the dataset from outliers in one single case.
- Classification: again we construct a class that uses methods from sklearn. Here we implement both supervised methods and unsupervised methods, together with a cluster labeling to give a final accuracy to the classification task.

CHAPTER 2

Creation of the dataset

2.1 Download of the files composing the dataset

The starting point for the classification is the collection of data in order to create a meaningful dataset. In order to do this, we take as reference the indications given by the **Genomic Data Commons** (GDC) of the **National Cancer Institute** (NIH) [5].
The GDC provides a repository containing a large dataset about patients in order to share information useful to improve the cancer research from the genomic point of view.
The largest repository is the one about the breast cancer. In that section it is possible to observe thousands cases, with informations about the different subtypes of breast cancer depending on the genomic characteristics. As already said before, we are interested in the files indicating the transcriptoma profiling, the ones in the **FPKM-UQ** format. We downloaded all possible transcriptomas in this format, and we obtain 1222 of them collected in single different .txt files: each file contains a column with the gene's names and a column with the expression of each of them.

2.2 Classification of the different cancer subtypes

Each transcriptome corresponds to a patient with a particular cancer subtype, so the following step was to label them in order to associate each patient to one of the possible subtypes.
Looking at an article from Nature [3] and from other publications [4], [2], [6], as said before, the most popular classification is the *PM50* classification, referred to TCGA for breast cancer.
This type of classification has been created in 2009 and spread after the publication of [2]: is a 50-gene subtype predictor, developed using microarray.

2.2.1 Associate the labels to the patient

Once the classes have been identified, we have to label the different transcriptomas with the name of the classes, in order to assign each sample to a cancer subtype.
Labels are taken from articles [3] (from which we were able to identify 545 subtype labels) and from [6] (from which we were able to identify 1148 subtype labels). Merging the labels together we create a dataset with 1159/1222 labeled transcriptomes of patients. As already said in the previous chapter, 1046 of them are characterized by one of the five types of cancer that have been mentioned before. Moreover, while labeling, we considered other two classes:

- Not Present, refers to 63 labels which were not available;

- Healthy, refers to 113 transcriptomes of healthy tissue.

What we have after these steps is a set of 1222 files with transcriptomas, a file that associates the code of the file to the ID of the patient and different excel files with the label for each ID. Therefore, we merge all these file together with two different scripts:

- A *MATLAB script* that creates the dataset as .mat file, which contains a matrix with the transcriptoma of all patient and a vector with all the labels;
- A *Python script* that creates a .dat files which contains the same information and an additional vector with the names of the features (i.e. the genes).

The dataset used in the .mat format is available at *https://www.dropbox.com/s/g2sggr622t6agu4 /Dataset.mat?dl=0*. It can be easly used also in python through the library scipy, that offers the command *loadmat* to load .mat datasets.

CHAPTER 3

Algorithm specifications

The algorithm and a README well explaining how to use it is available at the following link:
https://github.com/ABurrello/Pam50Classification_algorithms
Before entering in details of tuning of parameters and of the different techniques, we give a rapid glance to all the possibilities of our algorithm and to the general structure. Our model is composed by three sections, and for each one of them we have different choises:

- **Feature reduction**:
  - **Principal component analysis** (the only unsupervised technique);
  - **Linear discriminant analysis**;
  - **Genetic algorithm**;
- **Class imbalance**:
  - **None** in the case no imbalance management is wanted;
  - **SMOTE** and **Random Oversampling** for the oversampling the smaller classes;
  - **SMOTE+ENN** that considers an oversampling technique together with a cleaning one.
- **Classification learners**:
  - **Support Vector Machine, k-Nearest Neighbors and Random Forest** for the supervised part;
  - **k-Means, Hierarchical Clustering** for the unsupervised part.

All these methods will be deeply discussed in the next sections. In the figure 3.1 the high level structure of the algorithm is shown, highlighting the three different main blocks.

The algorithm is composed by four different classes and a Main file:

- *Main.py*: it only manages the k-fold training/testing loop and calls all the classes main methods to execute the 3 main blocks of the algorithm;
- *GA.py*: contains the Genetic Algorithm Class for feature selection.
- *Imbalance_manager.py*: it contains the class that manages the class imbalance. Passing the string with the imbalance management technique it will transform the dataset and resamples it using the target technique;
- *DR.py*: it works exactly as the Imbalance_manager, but for the dimensionality reduction of the dataset;

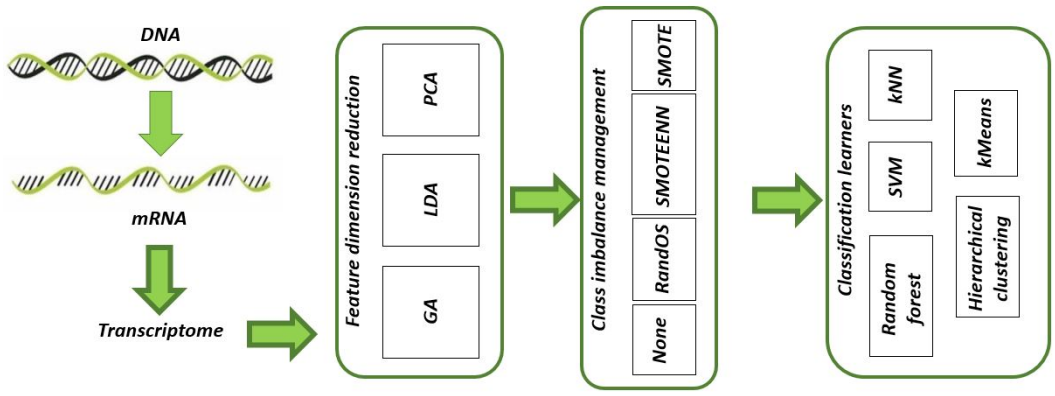

Figure 3.1: Shows the high level structure of the algorithm. It is very clear the division in the three stages, for which we give different options to be selected.

- *Classification.py*: This class implements all the learners. Also here we need only a string with the name to execute it. However, this class implements also the three scores used to evaluate the performances of the pipeline, the Accuracy, the F1-score and the BER (balanced error rate).
- *Opening_design.py*: it is a very simple class, which contains a methods called in the main to manage the user interface and the creation of the executed pipeline.

3.1 Feature reduction

Before applying the classification algorithm, a pre-processing phase is needed. This implies working on the dataset by applying some techniques for features extraction and selection. The main reasons behind this preprocessing are two:

- Working with so many features is very expensive and time consuming;
- We can risk an overfit (high variance) of the training dataset using a too high number of features: some techniques, like random forest, are very pruned to overfit with too many features and to underfit (high bias) with too few features.

Two different approaches are considered: the **Dimensionality reduction** and the **Feature selection**. To recall, in the first case the features are recomputed considering a linear combination of the initial features, while in the second case the new features are just a subset of the initial ones.
For the dimensionality reduction, we decide to follow the **Principal Component Analysis** (PCA) approach as unsupervised technique and the LDA technique for what concerns the supervised dimensionality reduction. Another algorithm, instead is used for the feature selection: a wrapper heuristic method, based on the Genetic Algorithm. We decided to used a wrapper method, rather than a filter method, because the dataset is not so big, there are not a lot of samples, so it does not need days to run.
From the results we got, the PCA seems to work better also because it takes into account all the features, differently from the Genetic Algorithm. The disadvantage is that in this way we lost all the direct references to the initial features: for example we cannot say anymore if a gene is more useful that another one with the PCA. With feature extraction, instead, initial data do not change.

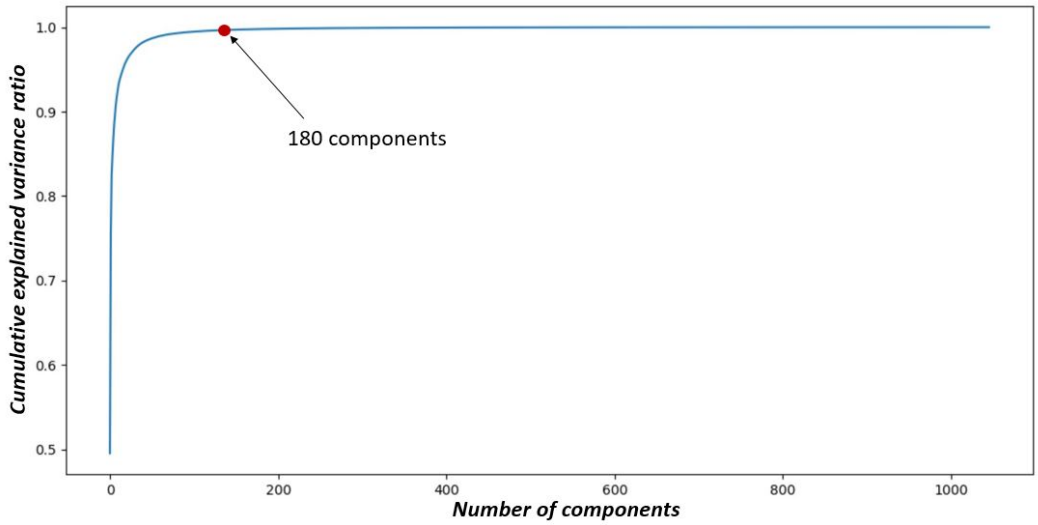

Figure 3.2: Shows the number of components corresponding to a target variance retained. On the x-axis there is the number of components, on the y-axis the cumulative variance.

3.1.1 Principal Component Analysis

The PCA is a well noted algorithm in the feature reduction domain: it takes into account all the features and gives back a number of components equal to $min\{\#samples, \#features\}$. The components are ordered for the explained variance ratio and a number of components has been chosen with the elbow rule. Another way is to calculate the variance retained with the formula below:

$$\frac{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)} - x_{approx}^{(i)}||^2}{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}||^2} \tag{3.1}$$

In our case, as shown in the figure 3.2 we take 180 components to run all our algorithms. In the section Results, we will see an overview of the results with different number of components, using the Support Vector Classifier as golden standard.

3.1.2 Linear Discriminant Analysis

The LDA is often known as a classification method. Nevertheless, it is composed by two stages: the first is the future reduction, that returns $min\{\#classes - 1, \#features\}$ number of features. Here, so we have only 4 significant features. The second is the classification stage.
We can understand that the number of features is very low with respect to the total one, but well suited for very low power methods as SVM or the same LDA, but not very suitable for Random Forest or Multilayer Perceptron Nets that are fed with many examples (and many features for example). Any particular choice of parameters has been done on the LDA.

3.1.3 Genetic Algorithm

Our dataset has a huge number of feature so it would be impossible to apply an exact method for feature selection. For this reason we decide to use an heuristic method which iteratively performs pseudo random operation on a vector of ones and zeros. The number of ones correspond to the final number of features that are chosen dependently on the position of ones in the array. We want to leave the number of feature as a settable parameter so we have to make attention on a couple of points.

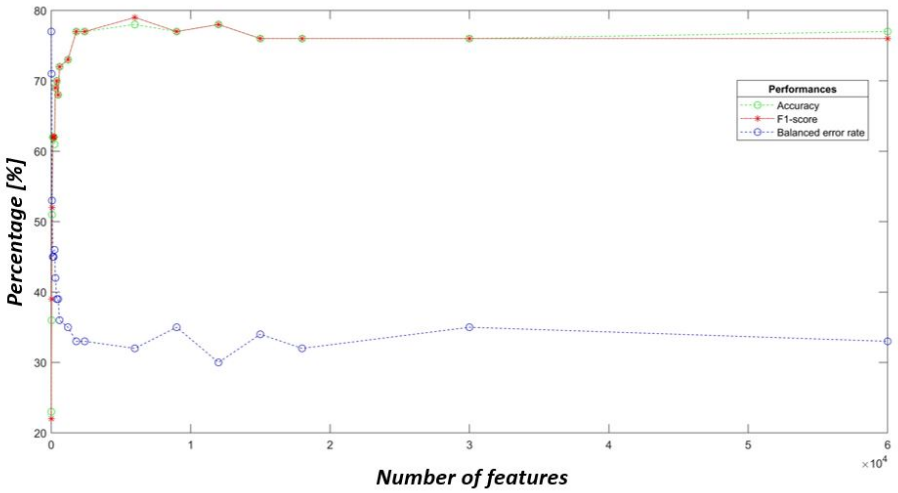

Figure 3.3: .

The vectors representing the feature selection are randomly initialized but forcing the density of ones to the desirable value. Then two possible operation are available:

- Mutation: N bits are randomly flipped but in order to maintain the same density the same amount of 1s and 0s are flipped.
- Cross Over: a random interval of random size of bits is exchange between two vectors. This causes a final density not exactly fixed to the desirable value, but for probability it remains very close to it.

After this step the eliticism is performed and a new family of vectors is generated starting from the best vectors of the previous generation.
This is performed N iter times that in our case is set to 5. Increasing this number the accuracy will slightly grow but this can also lead to a much higher computational time. The number of flipping bits and the maximum and minimum size are tunable parameters. It is worth to change them during the training in order to reduce the number of changes when the number of iteration increase enough. In this way it is possible to fine grain the final result.

As classifier we chose a linear Support Vector Machine that evaluates the goodness of the gene vector (and so of the selected features) by using a cross validation dataset randomly extracted from the training set.
N.B The GA.py contains a parameter that is num_worker: changing it you can use more threads and so speed up the algorithm, that otherwise is wery slow for an high number of features.

3.2 Class unbalance organization

One of the problems that could be found during classification is the class unbalance, which consists in having classes which are represented by a huge number of samples, and some other classes which have just a few samples. In this case, classification methods, like nearest neighbor method, for example, will be more pruned to classify each sample as belonging to the biggest class.
For this reason, solving the problem of class unbalancing is fundamental in order to have a good classification with some methods: we will see in the paragraph, looking at the results, that some methods will benefit a lot from the balancing of the classes.

From [7], different methods are proposed; we tried lot of them, but we decide to use only methods of pure over sampling of the dataset or a mixture between cleaning techniques and over sampling. As it can be also seen from the article, this methods outperform the under sampling techniques. In this case we have not to tune any parameters, but only to have an accurate analysis of the applicable techniques in this specific classification task. Here, we list the methods used with a very brief discussion:

- **Random over-sampling** is a non-heuristic method that aims at balancing class distribution by replicating randomly the minority class samples. The limitation is given by the fact that over fitting can occur, because we consider exact copies of the minority class samples. However, the results in [7] show very nice results and the complexity of the method is the simplest possible;
- **SMOTE** [8]: its main idea is to form new minority class examples by interpolating minority class examples. Thus, the overfitting problem is avoided and we create classes more similar to the reality.
- **SMOTE with EEN**: this technique is based on the one before, with the addition of a cleaning technique based on the nearest neighbor; it has to be noticed that the cleaning is applied before to the over fitting and we clean all the classes, not only the ones with the higher number of samples.

In the Chapter *Results* we will talk better about the impact of these balancing techniques on the different learners.

3.3 Classification Learners

This is the main part of the algorithm: data arrive at this point already with the right number of features and with balanced classes (if requested by the user). The classifier works differently in the two phases of operation: during training, the classifier has to be tuned through several techniques through the so called **training set**. After this, we have a fitted algorithm that we can use to infer the label of new and unseen samples. What we have just described is a classifier, so a supervised method that assigns a label to the samples. However there is also one other set of techniques, identified as *'clustering'* techniques that try to create clusters in the unlabeled data: a cluster is a set of points that are closer to the other with respects to other points, in some predefined metric. For this class of algorithms we do not divide the dataset to train and test the algorithm, but we keep the total dataset and try to create clusters: the labels are used only in the end to understand if this clusters have some meaning. In the following sections we will describe all the methods used in our tool.

3.3.1 Supervised methods

The supervised methods, as already said, need a labeled training set to fit the model and then a test set to verify the performance of the technique. The models could be *linear* if the boundary between the classes are made by linear functions (hyperplanes for the multi-dimensional space) or *not linear* if the boundary is not a linear function: not knowing the distribution of the points and since it is very difficult to understand it in this case, we considered both cases to see which one is giving the best results.

k Nearest Neighbors

This is the simplest classifier in the machine learning field. We decided to apply it, because it is worth to try this method in every dataset, to see if the data are very well distributed. It is very difficult to predict the result of this method a priori, but most of the experiments start from here to build more complicated learners on top of it. The algorithm is very simple, since it calculates the nearest points

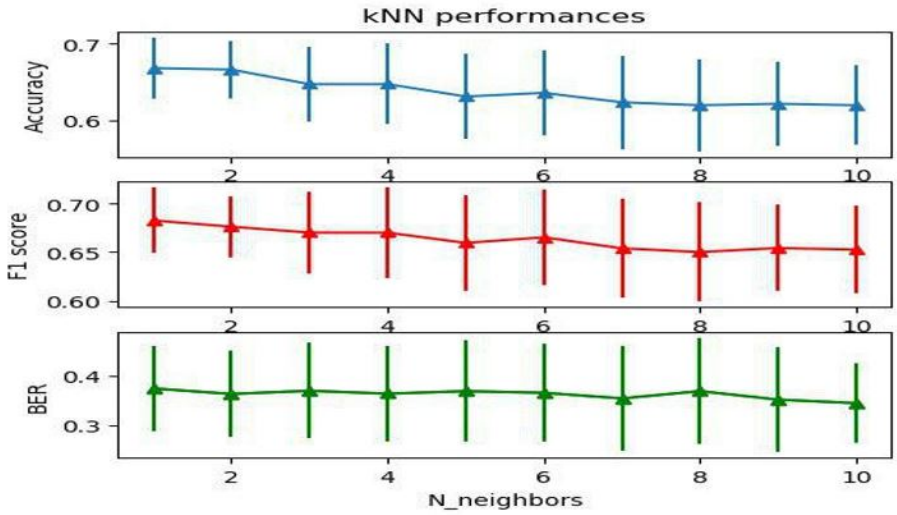

Figure 3.4: The accuracy of the kNN has been plotted for a cross validation set to decide the number of neighbors. We repeated the experiment ten times and plotted the three index used (section Results for a clear explanation). We finally decided to use neighbors = 1 since it gives the higher performances.

to the test sample and it assign the labels more present in that points. The only parameter to set is the number of neighbors, set in the way explained in figure 3.4

Support Vector Classifier

The SVC is a very common method and even using a linear kernel it is able to perform very well in a huge number of tasks. It divides the features space with a sufficient number of hyperplanes. The main parameter is "C" which regularize the algorithm avoiding a strong overfit of the data in the training set, that is very common when the number of features is high. Also for this reason a dimensionality reduction algorithm is strongly suggested for this method. Indeed the pipeline with PCA and SVC gave us the best accuracy.

Random Forest

The random forest is an ensamble of classifier based on the classification tree as base element. It has to be notice that the random forest is a non-linear classifier, so it divides the space in a different way with respect to the SVM. The random forest makes decision with a series of *if then else* rules. There are no tricky steps in the building of this random forest, only 3 parameters to set. As we can see in figure 3.5, this parameters have been set using a cross validation set to validate the model constructed. The final parameters are 40 trees, depth = 8 and 25 leaves.

3.3.2 Unsupervised methods

As already explained, the unsupervised methods do not use the labels to determine the classification, but try to create cluster in the dataset between similar points.// The only relevant thing before entering in the details of the methods is a choice that we did in the pipeline: since these are unsupervised methods, we decide use none of the balancing techniques (even if they could be helpful) and any preprocessing technique that uses labels. For this reason, when running an unsupervised method, one can only use PCA as preprocessing. The labels in this case are only used at the end to give a general idea about the meaning of the cluster created: to each cluster we assign the label of the most present

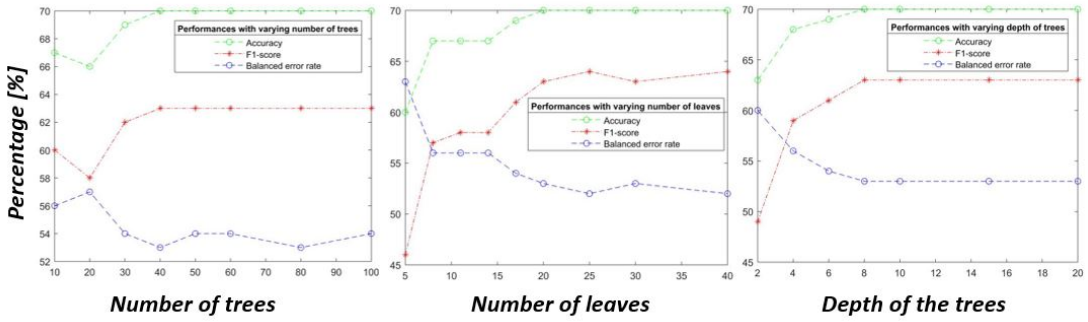

Figure 3.5: The figures represent the choice of the three most important parameters inside a random forest, made using a cross validation set: from the left, we can see the number of trees, the number of leaves and the maximum depth of each tree inside the random forest. The parameters have been chosen basing on the three indeces that you can see in the legend (for a detailed explanation of the index go the result chapter). In the algorithm we set 40 trees, max depth = 8 and 25 leaves. Each experiment to fix this parameters has been done keeping the other 2 fixed.

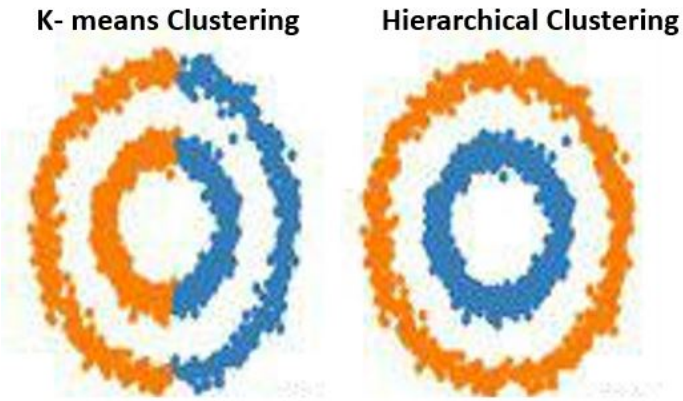

Figure 3.6: The image represents the different type of clusters created by the Hierarchical cluster and the kMeans.

class. We know that the class imbalance could be a problem in this case: in this way however we only have a worse accuracy, but we do not change the cluster that could be created on a real dataset without any adding of artificial data. Let us enter now in the details of the techniques.

kMeans and Hierarchical Clustering

The most popular unsupervised method is the kMeans algorithm. It separates the samples in K different groups of equal variance, minimizing the inertia, also known as within-cluster sum-of-square: $I = \sum_i (d(i, cr))$ where cr is the centroid of the assigned cluster and d is the squared distance. "K" is of course a parameter to be set and the most common way is to find the "elbow" in the inertia-ncluster curve that is plotted in fig 3.7. Even if there is not a clear elbow, we can consider the point where the graph starts decreasing linearly, this point is shown in red in the figure. In our case k=20 is the chosen number of clusters.
This parameter tuning is done using all the samples of the dataset, without using any cross validation set, because our aim using this unsupervised method is just to check if the generated clusters are also clinically meaningful.

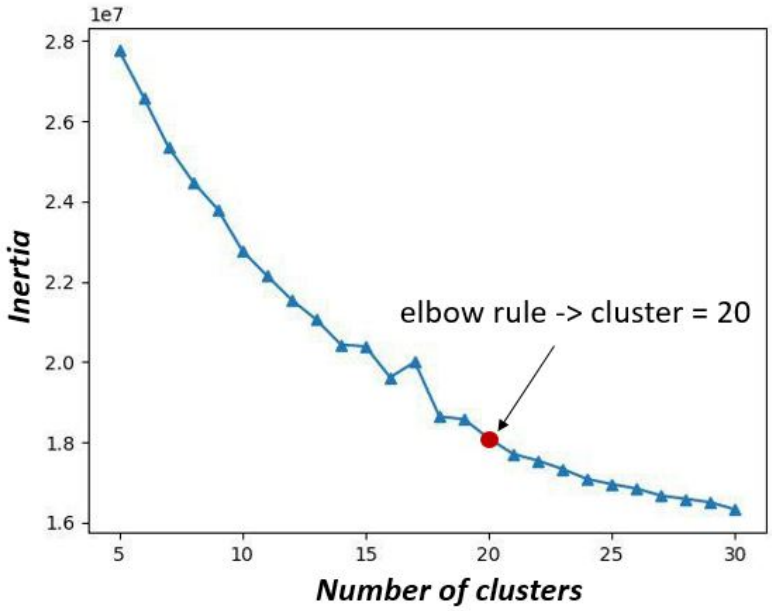

Figure 3.7: This graph has been useful to choose the number of cluster to use for the kMeans algorithm. We plotted the mean square distance of the points from the belonging centroids as explained in the text. Than we use the "elbow rule" to take the number of clusters for our analysis.

Why using more than 5 clusters? Does it make sense? Yes, of course, it depends on how the data are distributed in the space. There could be two clearly separated groups belonging to the same class, so two different clusters for the same class are necessary in that case.
How can we compare the so obtained 20 clusters with our 5 classes? The used method was to assign a label (between 1 and 5) to each cluster depending on the majority of the labels of the points belonging to that cluster. This means that if a cluster is composed by two samples of class A and three samples of class B than the label of this cluster would be B and the number of mispredictions would be two.

In addiction to the kmeans clustering we decided to apply also the Hierarchical Clustering because it can classify different distribution of data as we can see in fig 3.6. The results show the kmeans works better in our data distribution. The Hierarchical generates one big cluster including all the Luminal A samples and many other samples of other classes, and we can clearly detect it looking at the high BER.

This unbalancing in cluster size could strictly depend on the unbalance of our classes. Managing the imbalance ends up in a clear improvement in accuracy which will increase up to 75%. The test has been done only the original points, also if we use an "artificial dataset" to train. Nevertheless we decided to avoid these methods from the pipeline because they use the labels and so the pipeline would not be totally unsupervised. An alternative is to use the DBSCAN that automatically keeps in consideration the density of the data.

DBSCAN

The DBSCAN (Density-Based Spatial Clustering) was initially used to eventually fix the unbalancing issue. Supposing that each class is more dense in a specific area of the space, taking care of the density of the data during the cluster selection should improve the classification result. The main parameter to be set is the epsilon: the maximum distance between two samples for them to be considered as in the same neighborhood. This value has been chosen plotting the graph 3.8. This method is used in

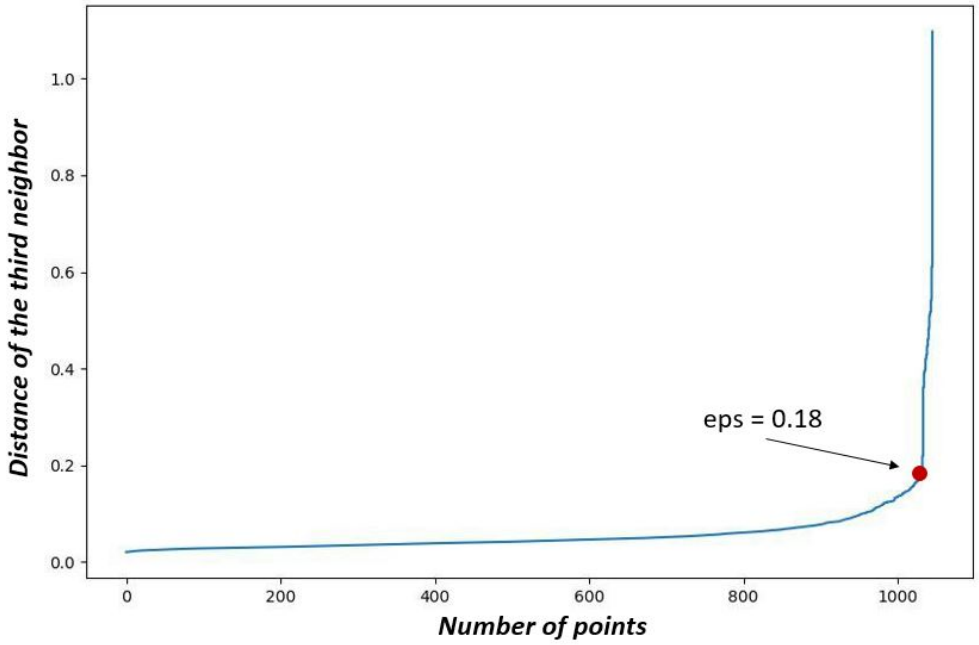

Figure 3.8: The graph is used to decide the value of eps for the DBSCAN algorithm. On the x-axis we have the number of points and on the y-axis the value of eps. The graph is drawn calculating for each point the distance of the third neighbor (the choice of eps is not so much biased by the decision of this number) and seeing how many points have the third-neighbor closer to the target value of y.

[9] and it works as follow:

- For each point the distance from all the other points is computed and only the third nearest distance is stored.
- The X axis indicates how many points have a 3rd nearest distance lower than epsilon.

With the right number of epsilon all the samples are not classified as noise. Nevertheless we got unexpected result: a unique cluster includes all the samples. This happens when the data are equally distributed over the space, so our initial assumption was wrong and this method was discarded.

CHAPTER 4

Results

In this section we go further in the details of the results for which it concerns the different techniques.

4.1 Performances evaluation

To evaluate the performances of our algorithms we decide to avoid to use multi index evaluation like Specificity+Sensitivity or Recall+Precision or Accuracy for individual classes since it is very difficult to use them to fast compare algorithms. So, basing on [10], we use three indeces that take into account all the performance statistics aforementioned.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

The accuracy is the simplest one and it can be directly applied to a multi-label dataset, by dividing the correct predicted samples by the total number of them; TP stands for true positive, TN true negative, FP false positive and FN false negative. However, this index could be biased by the presence of big classes, because it does not take into account the difference in the number of samples between them.

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} \tag{4.2}$$

The $F_1 score$ is the harmonic mean between the precision and the recall criteria. We have to notice that to apply this method to multi-label classes we have to do the $F_1 score$ for each individual class putting all the others in the other class. Than we rotate this experiment among all the labels and we do a final mean.

$$BER = 1 - \frac{1}{2}(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}) \tag{4.3}$$

The Balanced Error Rate (BER) score is the most representative for which it concerns the individual classes. In fact it mainly calculates the error done in the prediction of each class and do the mean between them. This score is very good to understand if also the small classes are well predicted.

4.2 Performances and conclusions

We here report all the results in two different tables for supervised (Table 4.1) and unsupervised pipelines (Table 4.3).

4.2.1 Supervised results

In the table 4.1 we can see all the results for the methods applied; we want to briefly discuss about some interesting points on the results achieved:

- the best method among the used ones is the SVC without any additional imbalance technique and with the PCA as preprocessing; moreover, the SVC for two of the three preprocessing techniques, doesn't suffer from any problem of class imbalance. This can be observed by the constant BER value among all the experiments using different class imbalance management.
- there is not a single best preprocessing technique, but, based on the classifier, the best one changes: it has to be noticed that all the number of features extracted has been trained by the SVC model or by the unsupervised technique and not adapted to other classifiers.
- we created a dedicated table (Table 4.2) to try to better understand the problem of imbalance of the classes. We take some particular examples from the 4.1 highlighted in the table: for each classifier we take an example of classification without balancing of the classes and one with the classes balanced and we compare not only the BER (that is always higher in the experiments with the balanced classes), but also the specificity of each individual class. It is very important to notice how the Sensitivity of the two classes, Normal-like and HER2, with the lower number of samples, achieved very higher sensitivity using one of the Balance technique for the samples. In particular, we can see that for the *Random forest*, without the embedding of the imbalance management the Sensitivity of this 2 classes is near to 0 with respect to the case with the SMOTE where they takes $\tilde{7}0\%$ Sensitivity: this could be caused by the minimum number of samples needed by both sides of the split, that avoid the tree from creating the particular split (for classes with low number of samples we can not reach this number). Finally, looking to the Table 4.2, we can understand that that both the two small classes are very well managed by the balance management with all the classifiers.

4.2.2 Unsupervised results

Both the unsupervised methods gives very poor results as reported in 4.3. The high BER suggest that almost all the sample are collapsed in a single cluster. Managing the imbalance of the classes will obviously help but this will not be unsupervised anymore.

Table 4.1: Summary of the performances of all the possible supervised pipelines.

	Accuracy	**F1 score**	**BER**
PCA + SMOTE + SVC	0.85	0.85	0.18
PCA + SMOTEENN + SVC	0.79	0.80	0.19
PCA + Random Over Sampling + SVC	0.85	0.85	0.18
PCA + None + SVC	**0.85**	**0.85**	**0.18**
LDA + SMOTE + SVC	*0.81*	*0.82*	*0.20*
LDA + SMOTEENN + SVC	0.80	0.81	0.20
LDA + Random Over Sampling + SVC	0.80	0.81	0.23
LDA + None + SVC	*0.80*	*0.80*	*0.32*
GA + SMOTE + SVC	0.78	0.78	0.32
GA + SMOTEENN + SVC	0.68	0.70	0.33
GA + Random Over Sampling + SVC	0.77	0.77	0.33
GA + None + SVC	0.77	0.77	0.33
PCA + SMOTE + kNN	0.70	0.71	0.30
PCA + SMOTEENN + kNN	*0.68*	*0.70*	*0.26*
PCA + Random Over Sampling + kNN	0.71	0.69	0.42
PCA + None + kNN	*0.71*	*0.69*	*0.42*
LDA + SMOTE + kNN	0.74	0.75	0.28
LDA + SMOTEENN + kNN	0.78	0.78	0.23
LDA + Random Over Sampling + kNN	0.77	0.77	0.30
LDA + None + kNN	0.77	0.77	0.30
GA + SMOTE + kNN	0.50	0.51	0.59
GA + SMOTEENN + kNN	0.42	0.45	0.54
GA + Random Over Sampling + kNN	0.54	0.52	0.63
GA + None + kNN	0.54	0.52	0.63
PCA + SMOTE + Random forest	0.74	0.75	0.29
PCA + SMOTEENN + Random forest	0.60	0.61	0.31
PCA + Random Over Sampling + Random forest	0.76	0.77	0.29
PCA + None + Random forest	0.72	0.67	0.52
LDA + SMOTE + Random forest	0.79	0.80	0.26
LDA + SMOTEENN + Random forest	0.78	0.79	0.27
LDA + Random Over Sampling + Random forest	0.81	0.81	0.27
LDA + None + Random forest	0.80	0.79	0.33
GA + SMOTE + Random forest	*0.82*	*0.82*	*0.21*
GA + SMOTEENN + Random forest	0.59	0.57	0.27
GA + Random Over Sampling + Random forest	0.81	0.81	0.23
GA + None + Random forest	*0.78*	*0.75*	*0.46*

Table 4.2: This table reports an additional metric used to evaluate the capability of the balancing techniques to increase the accuracy of the class with low samples. In this direction we use the *Sensitivity*, the number of samples correctly classified of a class over the total number of samples of that class.

Sensitivities	**Basal like**	**Normal like**	**Lum. A**	**Lum. B**	**HER2**
LDA + SMOTE + SVC	0.85	0.81	0.82	0.75	0.77
LDA + None + SVC	0.86	0.33	0.92	0.62	0.64
PCA + SMOTEENN + kNN	0.89	0.82	0.61	0.66	0.69
PCA + None + kNN	0.87	0.37	0.88	0.42	0.33
GA + SMOTE + Random forest	0.95	**0.61**	0.81	0.77	**0.82**
GA + None + Random forest	0.97	**0**	0.96	0.61	**0.15**

Table 4.3: Summary of the unsupervised learning performances. The performances have been calculated giving a final label to each cluster, based on the majority of the labels inside that cluster.

	Accuracy	**F1 score**	**BER**
PCA + None + kMeans	0.68	0.65	0.53
PCA + None + Hierarchical Clustering	0.63	0.51	0.63

Bibliography

[1] N. Zare, E. Haem, K. B. Lankarani, S. T. Heydari, and E. Barooti, "Breast cancer risk factors in a defined population: weighted logistic regression approach for rare events," *Journal of breast cancer*, vol. 16, no. 2, pp. 214–219, 2013.

[2] J. S. Parker, M. Mullins, M. C. Cheang, S. Leung, D. Voduc, T. Vickery, S. Davies, C. Fauron, X. He, Z. Hu *et al.*, "Supervised risk predictor of breast cancer based on intrinsic subtypes," *Journal of clinical oncology*, vol. 27, no. 8, p. 1160, 2009.

[3] C. G. A. Network, "Comprehensive molecular portraits of human breast tumors," *Nature*, vol. 490, pp. 61–70, 2012.

[4] S. E. Clare and P. L. Shaw, "big data for breast cancer: where to look and what you will find," *NPJ breast cancer*, vol. 2, p. 16031, 2016.

[5] Gdc.cancer.gov, "Home nci genomic data commons." [Online]. Available: https://gdc.cancer.gov/

[6] D. Netanely, A. Avraham, A. Ben-Baruch, E. Evron, and R. Shamir, "Expression and methylation patterns partition luminal-a breast tumors into distinct prognostic subgroups," *Breast Cancer Research*, vol. 18, no. 1, p. 74, 2016.

[7] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[9] M. T. Elbatta and W. M. Ashour, "A dynamic method for discovering density varied clusters," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 6, no. 1, pp. 123–134, 2013.

[10] N.-Y. Nair-Benrekia, P. Kuntz, and F. Meyer, "Learning from multi-label data with interactivity constraints: an extensive experimental study," *Expert Systems with Applications*, vol. 42, no. 13, pp. 5723–5736, 2015.