Using gene expression data for brain region classification: Analysis of different classification techniques

Augusto Marques *IST-89789* | Laura Quintas *IST-92759* | Carolina Cruz *IST-92801* | Teresa Ribeiro *IST-92842*

Course: Machine Learning in Bioengineering – *Instituto Superior Técnico*

***Abstract*—Machine learning approaches have the potential to be very useful in studying and characterising transcriptomic data, namely from organoid samples. In this project we tested several different classifiers (k-nearest neighbors, decision trees, support vector machine and multi layer perceptrons) to determine the best algortihm to use when classifying gene expression data into a brain region. This data consisted of RNA-seqs obtained from six individuals. We compared raw and treated data, tested several hyperparameters for each classifier and compared the results. We were able to obtain good performances for all classifiers, especially in the log transformed datasets. Our results also suggest that SVM and DT are the best classifiers for this type of data, while KNN and MLP might be more suited for microarray data.**

***Index Terms*—Machine Learning, Gene expression data, RNA-seq, Classification, Brain Region**

I. Introduction

A. Motivation and Goals

The brain is an essential organ of the human body, and however there's a lack of understanding about its complex processes and about the disorders that affect it. Brain organoids are three dimensional, self-organized tissues, that mimic the main features of the human brain, both physiologically and histologically. [1]

To use this technology in the study of neurological diseases it's important that we're able to identify the brain region and maturity stage of samples. Usually, this is done using a few expression markers, which is a useful and cheap technique, but it doesn't fully capture the complex gene expression patterns that are present. Given that different protocols for organoid growth will have distinct differentiation pathways, a complete characterization of developing organoids is of great use for better understanding these mechanisms. [2]

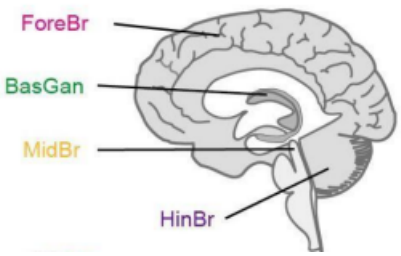


Fig. 1: Division of the brain into 4 main regions. Retrieved from *Dong et al.* [3]

.

Machine Learning approaches have been applied to study transcriptomic data from brain cells, in an effort to find better ways to understand and classify this data. [3] In the future, these computational techniques may be used for fast identification of brain regions and maturity stages of organoid samples.

Our goal in this project was to apply machine learning, namely supervised learning, to brain region classification. We compared several classifiers, and different hypermarameters in each classifier, to try to determine the best approach to brain region classification from RNA-seq data.

B. State of the Art

Gene-expression-based classification algorithms is an emerging powerful method for diagnosis, disease classification and potential markers identifier. The majority of statistical approaches developed for this method are either based on a continuous scale, such as microarray data which has been extensively used over the past decades, or rely on a normal distribution assumption. Because of benefits like delivering less noisy data, finding new transcripts and isoforms, and not needing predefined transcripts of interest, RNA-Seq has recently superseded microarrays as the tool of choice for assessing gene expression [4].

Regarding microarray data-based classification studies, the Support Vector Machine (SVM), the Multi Layer Perceptron (MLP) and K-Nearest Neighbor (KNN) showed similar tendencies in their results and outperformed the remaining approaches, Random Forest (RF) and Decision Trees (DT) [5]. Kernel- and ensemble-based algorithms, including the SVM and RF, demonstrated fast performance and strong prediction ability in a mixed data-based analysis that incorporates RNA-Seq data. Comparatively, the overall predictive ability of KNN and MLP was relatively moderate [6].

Microarray-based classifiers can still be effectively used to classify RNA-Seq data since those samples can resemble microarrays when processed using techniques like regularized logarithmic transformations [4]. Taking into consideration this information, we selected the top performing classifiers, including SVM, MLP, and KNN to categorize the RNA-Seq input into the main brain regions, as well as DT, to enable us to comprehend and interpret the classification using trees [7].

II. Methodology

In this section it is described the procedures and approaches taken regarding data acquisition and corresponding preparation (for classification), as well as a brief theoretical explanation of the 4 classifiers used.

A. Data Preparation

Initially, four datasets regarding gene expression data were provided, one of which was the original data taken from *Dong et al.* [3]. In these datasets, each row represents a gene and each column represents a sample of the brain. An example of the structure of these datasets can be seen in Appendix A.

After loading the four datasets, these were structurized to serve as input for the classifiers ahead. For this, a function called `structurize()` was made, that receives a given dataset and transposes it, transforming genes to columns and samples to rows. Additionally, it creates a variable with the samples' respective brain regions. An example of the structurized dataset can be seen in Appendix A of this paper. This variable was then altered by changing its value to the associated bigger region of the brain, out of 4 possible ones (instead of having 25 possible subregions), using the information in *Dong et al.* [3]. This represented the target variable of classification and has the following labels:

- `ForBr` (fore brain)
- `BasGan` (basal ganglia)
- `MidBr` (mid brain)
- `HinBr` (hind brain)

With these dataframes, four new datasets were made using feature scaling and feature selection (FS), thus having 8 different datasets to be used for classification, which are called:

1) **Raw**
2) **Raw_filtered**
3) Raw_filtered_FS
4) **logCPM**
5) logCPM_FS
6) **logCPM_centered**
7) logCPM_centered_FS
8) logCPM_zscore

To better understand the logic over the production and choice of these 8 datasets, we will now go through the preparation process of the datasets. The bold datasets are the datasets provided initially, and from which the other four in the list above were computed.

Firstly, the *Raw* dataset was obtained, containing the counts of the reads of RNA molecules for each gene in the different samples (265 rows/samples by 58930 columns[1]). This data is not treated and has a lot of low expressed genes, thus requiring further processing. Hence, a filter was applied, greatly reducing the number of genes, creating the *Raw_filtered* dataset (now with 33280 columns). After this, feature selection was applied using a variance threshold, which dropped all features with a variance lower than that threshold, creating the *Raw_filtered_FS* (442 columns). A threshold of 0.00005% of the maximum registered variance was set for the creation of this last dataset. Next, using the *Raw_filtered* dataset, the values of the read counts were altered. A normalization against the library size of each gene was performed, and the counts per million (CPM) were computed, creating the *logCPM* dataset (with 33280 columns). Again, for this last dataset, the same feature selection procedure was applied, reducing the number of features, generating the *logCPM_FS* dataset (836 columns). Note that for this last dataset the threshold defined was 20% of the maximum registered variance, since the values were subjected to the same scale. For the values of *logCPM*, these were also provided with mean centered to zero, allowing a more in-depth study (*logCPM_centered*). After this, feature selection was applied one last time, using again the 20% maximum variance threshold, creating *logCPM_centered_FS* (836 columns). Finally, a zscore scaling was performed, to see the results when imposing a standard deviation of one to the variables, as well as mean equal to zero (*logCPM_zscore*) (normally distributed variables).

B. Data Classification

1) Cross-validation

When using supervised learning to classify data, attention must be paid so that some common errors are not committed, one of the major ones being overfitting. Overfitting occurs when the predictive model fits too closely to a specific set of data, in a way that it's no longer able to generalize and predict the behaviour of data outside of that specific dataset.

To overcome this, cross-validation comes as a possible solution. It does so by partitioning the train dataset into smaller datasets so that the model is trained with separate datasets, instead of just one with all the data available. In our case, we used the function `StratifiedShuffleSplit()` from the `sklearn` Python package (Figure 2). This method, besides shuffling and splitting the train dataset, guarantees that the proportion of samples from each class is the same in each split (fold). This is specially important since our data is severely unbalanced [2], and in this way we assure the model includes samples from all 4 regions of the brain.

2) Performance Measures

To evaluate the performance of our classifiers, in order to later compare them, we decided to produce a confusion matrix for each dataset and to determine the accuracy, precision, recall and F-score for each one (Figure 3).

The confusion matrix consists of a table that displays the several possible combinations of true and predicted values

[1]For each dataset, the number of genes considered is always equal to the number of columns -1, i.e. not considering the target variable.

[2]In all 8 datasets we have 265 samples, from which 166 belong to ForBr, 59 to MidBr, 30 to BasGan and 10 to HinBr.

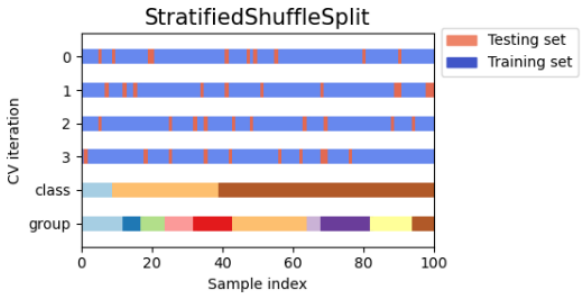


Fig. 2: Visualization of the `StratifiedShuffleSplit()` function behaviour. [8].

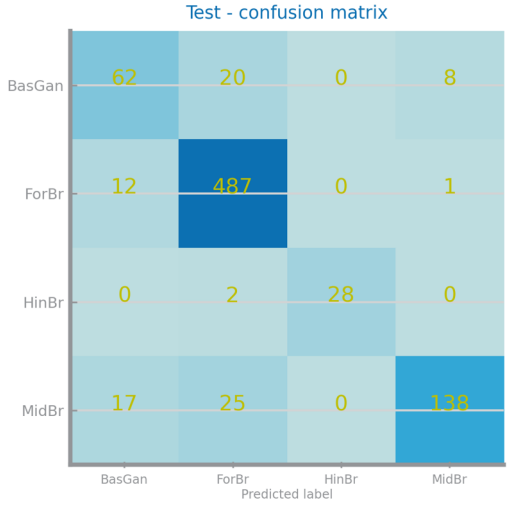


Fig. 3: Example of one of the obtained confusion matrices.

for all classes. The higher the values on the diagonal of this matrix, the most samples were correctly predicted, and the better the performance of the classifier. A perfect classifier would produce a diagonal matrix as its confusion matrix.

Accuracy, precision, recall and F-score can be obtained from the confusion matrix and are some of the most widely used measures when evaluating classifiers. Accuracy (Eq. 1) measures how many of the predictions made by the classifier are correct. Precision (Eq. 2) tells us how many of the samples classified as belonging to a certain class belong in fact to that class. Recall (Eq. 3), or specificity, measures how many of the samples that should have been classified as belonging to a certain class were correctly classified. Two models with low precision and high recall (or vice versa) are hard to compare using only these measures. Therefore, we also use F-score (Eq. 4), since it combines both of these measures, to make them comparable. F-score is the harmonic mean of recall and precision. The harmonic mean is used instead of the regular arithmetic mean to punish extreme values more. All of these metrics should be as high as possible.

$$Accuracy = \frac{\text{true positives} + \text{true negatives}}{\text{total number of samples}} \quad (1)$$

$$Precision = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \quad (2)$$

$$Recall = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \quad (3)$$

$$F\text{-}score = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} \quad (4)$$

All of these measures were computed in each dataset for each class (ForeBr, BasGan, MidBr, HinBr) and these were averaged to obtain the values for the dataset as a whole (Eq. 5). Note that this average is weighted, according to the number of samples considered for each class, during the classification task.

$$PM = \sum_{c} \frac{PM_c w_c}{4} \quad (5)$$

Where PM is a performance measure (accuracy, precision, F-score or recall), w the weight and c a given class (ForBr, HinBr, BasGan or MidBr). Note that the weight for a given class, w_c, is given by the number of samples evaluated for that class divided by the total number of samples evaluated.

C. Classification Algorithms

1) K-Nearest Neighbors

The k-nearest neighbors algorithm (or knn) is an algorithm with a geometrical approach. It will classify a new sample based on the distances between the new sample and the ones on the training set. The test sample is classified as belonging to the same class as the majority of the k nearest samples.

This is an easy to implement algorithm that doesn't require the tuning of many parameters. However, it gets significantly slower with the increase in the volume of data, as both its most efficient methods (ball tree and k-d tree) have a training time complexity of $O(d \cdot n \cdot log\ n)$, with d being the data dimensionality and n the number of samples in the training dataset. [9]

Regarding the k-nearest neighbors hyperparameters, the most relevant are weighting, distance and the number of neighbors. The most common distance measures are euclidean, manhattan and cosine. In this project we used euclidean distance, as according to Parry et al. (2010) [11] the differences between the results obtained by these three distance measures weren't relevant and euclidean is the standard distance measure.

Besides equal weighting, in which all samples weight the same, we can use distance based weighting, in which samples closer to the test data point are taken in more consideration than samples further away. Again, as according to Parry et al. (2010) the differences in results between these two weighting methods were negligent, we chose the standard option, that is, equal weighting.

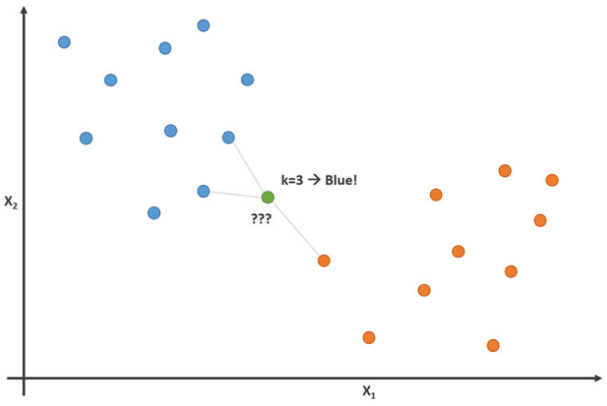


Fig. 4: Visual representation of the knn algorithm with $k = 3$. Image taken from [10].

The number of neighbors, on the other side, is a parameter that highly affects the performance of the classifier. The optimal k is dependent on the data, and can be very different depending on the type of data. In fact, sometimes a higher k improves performance [11] and in other cases the lower values are the ones that result in better performances [12]. Therefore, to obtain the best results, we decided to determine the optimal number of neighbors for each dataset and, from there, choose the best k for the classification task.

2) Support Vector Machine

Due to its solid theoretical foundations and strong generalization capabilities, Support Vector Machine (SVM) has emerged as one of the most widely used classification techniques in recent years. This capability is achieved by using an optimal $(N-1)$-hyperplane with the maximal margin, which divides a group of elements belonging to different classes, in an N-dimensional space, into subsets [5]. Although it has the potential to achieve a high performance, its implementation requires the optimization of the training process's hyperparameters, which has been shown to be essential for obtaining reliable predictions [13]. Among those are: the $kernel$ which is a method for using a linear classifier to solve a non-linear problem (fig. 5); the C variable that refers to the penalty parameter of the error term; It trades off maximization of the margin of the decision function against correctly classifying training sets; and the gamma parameter, γ, related only to the Radial Basis Function (RBF) which is the most popular kernel, determines the impact of a single training set in the hyperplane's class. [14] [15].

3) Decision Trees

One of the machine learning algorithms that has been gaining popularity in recent years for application in bioinformatics and genetics are tree based methods such as decision trees (DT). [17]

The fact that it is an extremely simple, interpretative, of fast-execution and visible model associated to its none-to-little data preparation requirement, and its capability of handling multiple outputs and high dimensional data with limited sample

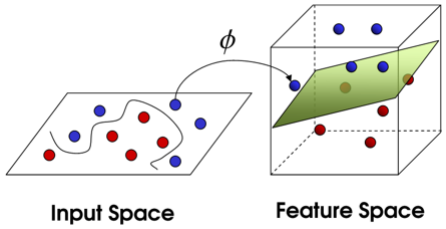


Fig. 5: Schematic representation of $(N-1)$-hyperplane creating subsets in a N-dimensional space using the kernel trick, ϕ [16]

sizes, justifies some apparent success of this type of classifier in recent years in the biomedical field. [7] [17]

Almost all classification tree building algorithms, such as C4.5 and CART (whose optimized version is used by `scikitlearn` Python package), employ a top-down heuristic search that recursively partitions the instance space using hyperplanes that are orthogonal to axes. [7] Starting from our training samples, the root node will represent the attribute or feature that for a given evaluation using a statistic best classifies these. The value of this feature is then used to split the training samples into descendant or child nodes, and this process is iteratively repeated until a pre-established stopping criterion is met, and the terminal nodes are assigned the class label that is associated with the largest number of samples contained in that node. [17] [18]

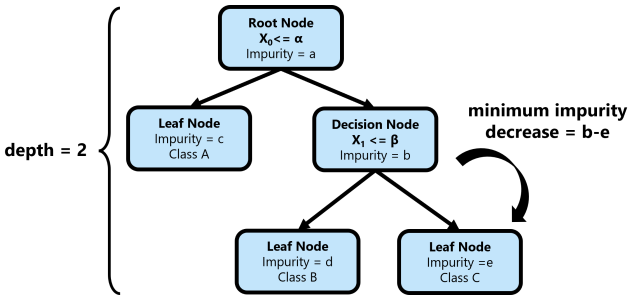


Fig. 6: Schematic representation of a decision tree with *depth* = 2.

Given the nature of this classifier, the critical step for the construction of DT is to select the best feature that will divide the space of instances, where usually the algorithms employ the concept of impurity (being that the lower impurity obtained by the split based on a given feature is considered better). In ML, it is common to use as measures of impurity of a given node t two criteria: the Entropy (where the reduction of the same is referred to as information gain) and the Gini index, whose formulas are presented in equations 6 and 7, respectively. [17]

Moreover, in the case of DTs it is also extremely important to define the criteria for stopping the splitting process because large tree models tend to cause overfitting of the model, especially in the case of data containing many features, as in the case of RNA-seq queries. So one of the essential processes for the use of DTs as classifiers is the pruning of the

$$I_E(t) = -\sum_{j=1}^{l} p_j(t) \log_2 \{p_j(t)\} \quad (6)$$

Equation 6: Mathematical expression to compute the Entropy of a node t where l represents the different classes depicted in the dataset and p_1, p_2, ...p_l the proportion of samples in the l classes, respectively. Retrieved from [17]

$$I_G(t) = \sum_{j=1}^{l} p_j(t) \{1 - p_j(t)\} \quad (7)$$

Equation 7: Mathematical expression to compute the Gini index of a node t where l represents the different classes depicted in the dataset and p_1, p_2, ...p_l the proportion of samples in the l classes, respectively. Retrieved from [17]

tree, which can be done in different approaches that explore different sets of hyperparameters, such as setting a maximum depth for the tree or the minimum impurity decrease between two nodes from different depths. In this way, it is possible to ensure that a new node/split of the tree presents a relevant and non-redundant information about the instance space.

4) Multi Layer Perceptrons

Multi Layer Perceptrons is a layered neural network with one or more hidden layers between the input and output layers. MLP is made of perceptrons. A perceptron, or single-layer perceptron, is composed of neurons, in comparison to what happens in our brain. The perceptron receives a certain data vector as input and attributes a weight to each entry. Then, it computes the sum of all pairs neuron times weight, producing an output. This output is then passed to an activation function, usually a step function, either returning 0 or 1.

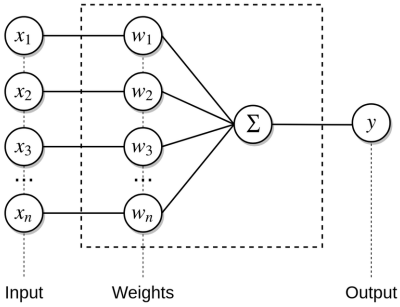


Fig. 7: Schematic representation of the perceptron model. Retrieved from Gratarrola, 2017 [19].

Most often, a single perceptron is not enough to give a good classification of the data, thus the need for more layers and the name multi layer perceptrons. In this case, learning occurs by changing the weights after each vector of data is processed, based on the amount of error in the predicted result compared to the true result.

III. Results and Discussion

In this section, the performance results for each classifier are presented, as well as their discussion regarding the parameters chosen and the biological context of the data provided.

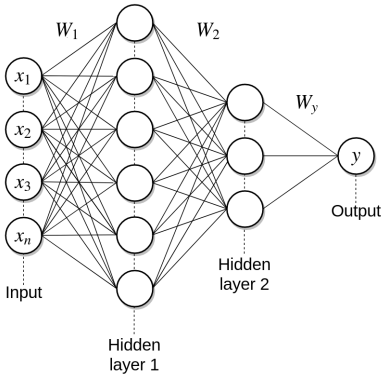


Fig. 8: Schematic representation of a two-layer perceptron model. Retrieved from Gratarrola, 2017 [19].

A. K-Nearest Neighbors

In order to determine the optimal number of neighbors for each dataset we performed the algorithm multiple times for each dataset, with the number of neighbors ranging from 1 to 30. The performance measures for each value of k were plotted for each dataset, for both the train and test datasets, as can be observed in figure 9. The remaining figures can be seen in Apeendix B.

Fig. 9: Performance measures plotted against the number of neighbors in the logCPM_centered_FS dataset

For each dataset, we determined the optimal k as the one that maximized the most parameters. In most cases the best number of neighbors was 1, with the exception of the logCPM_zscore dataset, for which the best results were obtained with $k = 3$ (see figure 10).

We then ran the algorithm for each dataset using the previously determined optimal value of k (Figure 11). The results obtained showed overall good results, with most datasets showing results above 80% for all measures. A stark difference can be observed between the treated and raw datasets. All three of the raw datasets have performance measures of 76% or lower. We can therefore conclude that the k-nearest neighbor algorithm isn't suitable for untreated data.

A second observation that can be made is that, for the logCPM datasets, the ones that underwent feature selection obtain

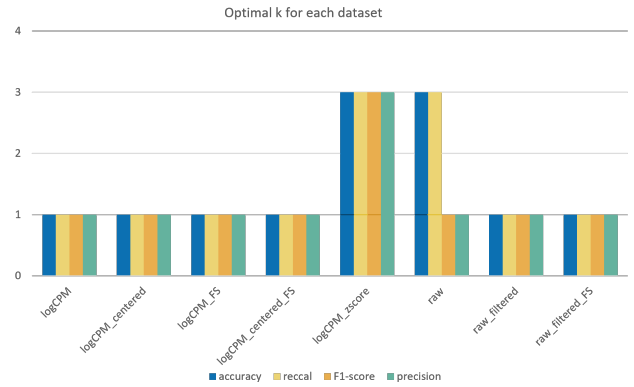


Fig. 10: The value of k that maximizes the performance measures for each dataset. For some parameters more than one k obtained the best results, in which cases the second value of k is represented as a thin line on the bar.

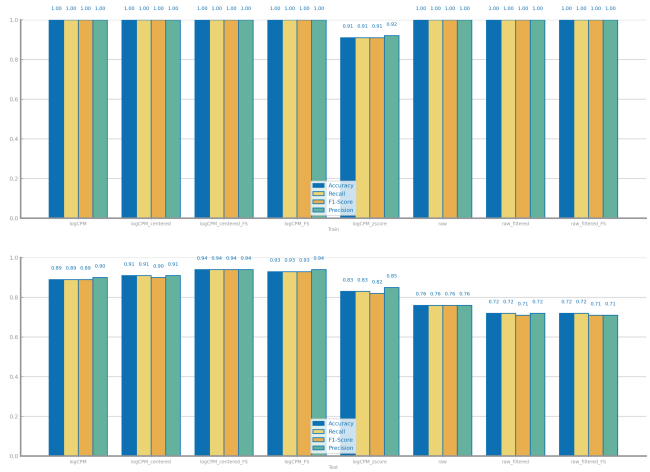


Fig. 11: Performance measures of the KNN classifier across all datasets (both train and test).

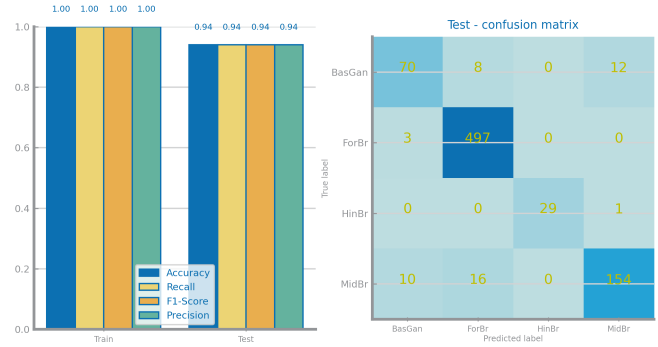


Fig. 12: Confusion matrix and performance measures for the logCPM_centered_FS dataset.

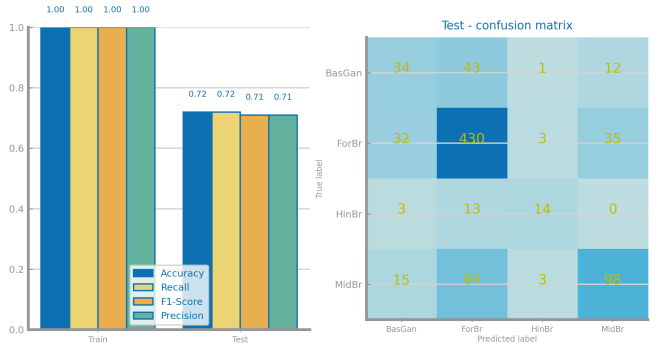


Fig. 13: Confusion matrix and performance measures for the raw_filtered_FS dataset.

Table 1: Hyperparameters evaluated for the SVM algorithm.

kernel	Linear		RBF	
C	0.1	1	10	100
γ	0.01	0.001	0.0001	0.00001
class_weight	None		Balanced	

the best the performances. Of these, the dataset with best results was the logCPM_centered_FS, with 94% score for all performance measures (fig 12).

In the untreated datasets, however, the opposite is observed, as the data that underwent feature selection results in the worst classifications. The raw_filtered_FS was the dataset with worst performance, with the performance measures averaging 71.5% (fig 13). The confusion matrixes and performance measures for the remaining datasets can be found at Appendix B.

B. Support Vector Machines

In order to test and evaluate the optimal hyperparameters, the performance of the combination of the parameters in table 1 was assessed and represented in a ranking graph (Figure 14), as well as in heatmaps.

The key difference between the two kernels is the raw data performance as it can be observed in the heatmaps (Figure 15), with the rbf kernel having an accuracy of 0.62 compared to 0.91 from the linear kernel. This was an expected result since the rbf kernel is not suitable when there's a high number of features, which is the case of the raw datasets [14].

As seen in the left heatmap (Figure 15), the accuracy was unaffected by any of the possible values of the C parameter, hence the value 1 was chosen as it is the default, as well as to minimize overfitting. Likewise the *class_weight parameter* values showed no difference, but because our data is unbalanced, the *balanced* option was selected. Lastly the γ parameter only applies to the rbf kernel.

To sum up, the chosen parameters were:

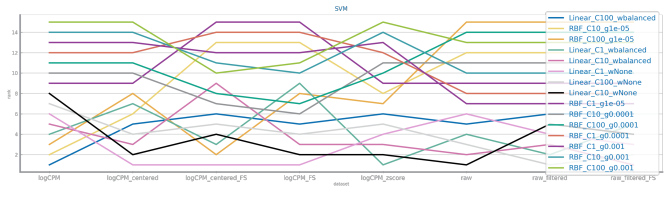


Fig. 14: Performance rank of each possible combination of hyperparameters in table 1 throughout the 8 different datasets.

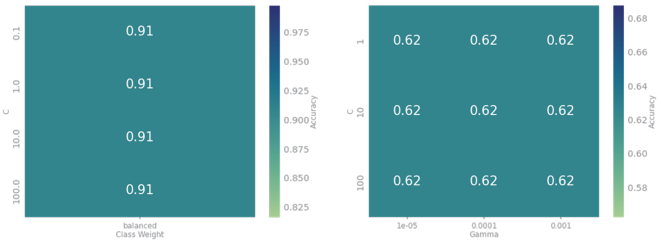


Fig. 15: Accuracy performance for the linear (left) and rbf kernel (right) in the raw dataset.

- Kernel: `Linear`
- C: `1`
- γ: `None`
- Class weight: `Balanced`

SVM performed exceptionally well overall, with all dataset performance metrics having a value higher than 0.9. The logCPM, logCPM_FS, logCPM_centered and logCPM_centered_FS datasets all showed the best (and equal) results. The remaining results of the confusion matrixes for all datasets can be seen in Appendix C.

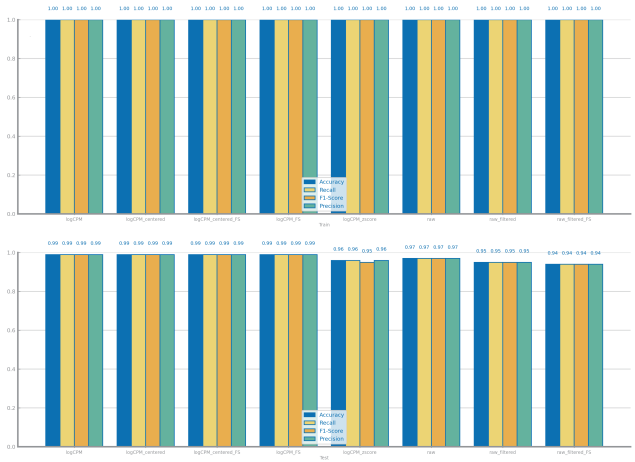


Fig. 16: Average performance measures (across all 10 folds) of the SVM classifier in train (up) and test (down) sets, in the 8 datasets used.

C. Decision Trees

As already mentioned in section II-C3, choosing the correct hyperparameters for the models built by the classifier is crucial to avoid overfitting the tree to the data and get the best performance. Thus, combinations of the hyperparameters presented in Table 2 had their performance evaluated, obtaining the results of the presented ranking graph (Figure 17) and in the heatmap (Figure18). The max depth selected was the one recommended by Kim *et al.* [5] and the two standard criteria and minimum impurity decrease values selected are the ones commonly used.

As it is shown in the results it is not possible to identify a set of hyperparameters that systematically performs best

Table 2: Hyperparameters evaluated for the DT algorithm.

max depth	3		
min impurity decrease	0.0001	0.0025	0.0005
criterion	entropy	gini	

Fig. 17: Performance rank of each possible combination of hyperparameters in Table 2 throughout the 8 different datasets.

in the rank graph, and in the heatmap shown, for a max depth of 3 the accuracy variation between the different sets of hyperparameters does not exceed 0.02, suggesting that the tuning of these hyperparameters does not produce a relevant impact in the performance of this classifier. Nevertheless, to proceed with the analysis the following set of hyperparameters was selected to test this classifier, represented in the rank graph (Fig.17) by the yellow line:

- max depth: `3`
- criterion: `'entropy'`
- minimum impurity decrease: `0.001`

Regarding the results for the classification task, as it is possible to observe in Figure 19, in overall the results obtained for this classifier are quite satisfactory with the majority of the datasets performing near or above 90% except in the case of the raw filtered dataset that registered the lowest values of performance, with an average of 77% in the score metrics.

It must be highlighted the performance of DT for the logCPM datasets where FS was performed, where the average performance for the 4 score measures reached 95% in the test set. The confusion matrix and isolate performance results for the best (log_CPM_FS) and worst (raw_filtered_FS) performance datasets are shown as well (Figure 20 and Figure 21, respectively). The remaining results for the other datasets can be found in Appendix D.

In addition, because this classifier allows the visualization of

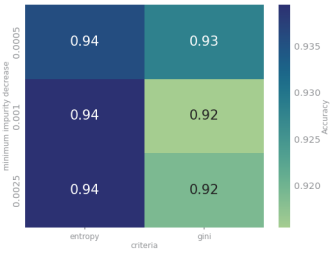


Fig. 18: Accuracy performance for maximum depth of 3in the logCPM centered FS dataset.

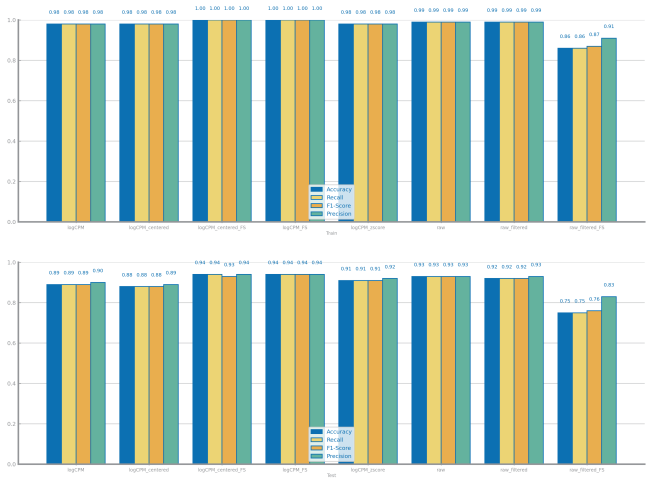


Fig. 19: Average performance measures (across all 10 folds) of the DT classifier in train (up) and test (down) sets, in the 8 datasets used.

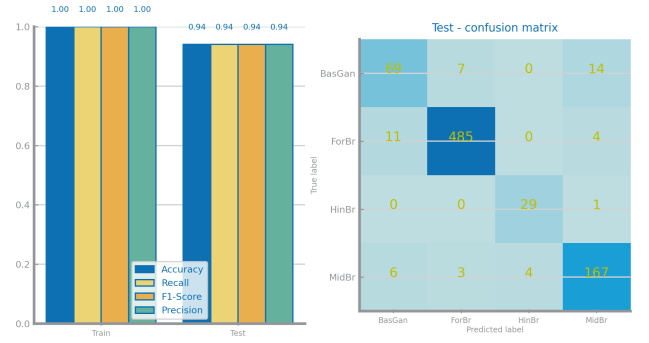


Fig. 20: Average performance measures (across all 10 folds) of the DT classifier in the test set (left) and corresponding confusion matrix (right), for the logCP_FS dataset.

its learned model, the diagram representing the trees built for each dataset were analyzed, with the tree for the log_CPM_FS dataset shown in Figure 22 and for the raw_filtered_FS shown in Figure 23. The remaining trees can also be found in Appendix D.

By comparing the best and worst performing datasets, log_CPM FS and raw_filtered_FS, it was possible to suppose an explanation for the worse performance of the latter.

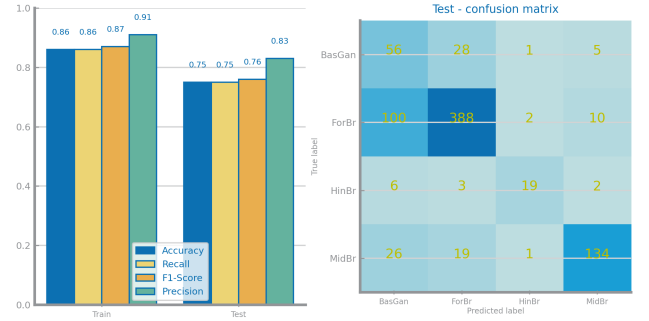


Fig. 21: Average performance measures (across all 10 folds) of the DT classifier in the test set (left) and corresponding confusion matrix (right), for the raw_filtered_FS.

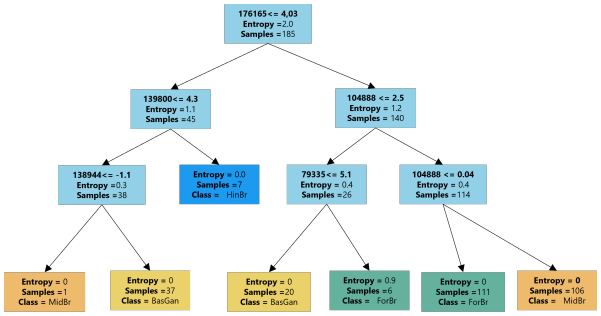


Fig. 22: Tree model obtained by the algorithm for the logCPM_FS dataset presenting the features selected to perform the split, the amount of impurity, and number of samples contained in each node for the decision nodes (light blue nodes) and the impurity, number of samples and class most prevalent in the leaf nodes (colored nodes)

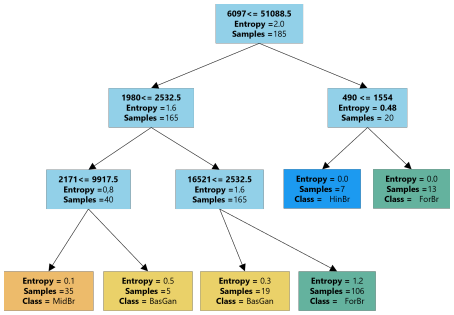


Fig. 23: Tree model obtained by the algorithm for the raw_filtered_FS dataset presenting the features selected to perform the split, the amount of impurity, and number of samples contained in each node for the decision nodes (light blue nodes) and the impurity, number of samples and class most prevalent in the leaf nodes (colored nodes)

Observing the model tree built for LogCPM_FS, it is possible to observe that genes **176165** and **144888** are used as attributes on which the instance space split is based in the first node and in the second level node, as it happens in the majority of the other datasets (7 out of 8 trees and 6 out of 8 trees, respectively), which may indicate some biological relevance of these genes in the task of distinguish the different brain regions. However, as it is possible to observe, in raw_filtered_FS dataset these genes are not included in the model tree since during the FS task they were removed from the dataset. This fact may explain the worst performance of DT on this particular dataset.

D. Multi Layer Perceptrons

Regarding the classification task, the MLP classifier parameters were defined according to *Kim et al.* [5]. The parameters chosen were:

- Activation function: `ReLU`
- Solver: `Adam`
- Initial learning rate: `0.01`
- Maximum number of iterations: `500`

This choice tends to avoid under-fitting and high-memory capability. According to Figure 24 the best performance was registered for the logCPM dataset using centering and feature selection, with performance values equal or higher than 95%; then a good and bad performance for the logCPM and raw

family datasets was achieved, respectively, and finally the worst performance for the logCPM dataset, with performance values below 60%. The confusion matrix for the best performance is shown as well (Figure 25), with the number of corrected and uncorrected predictions for each brain region, for a clearer view of the classification task.

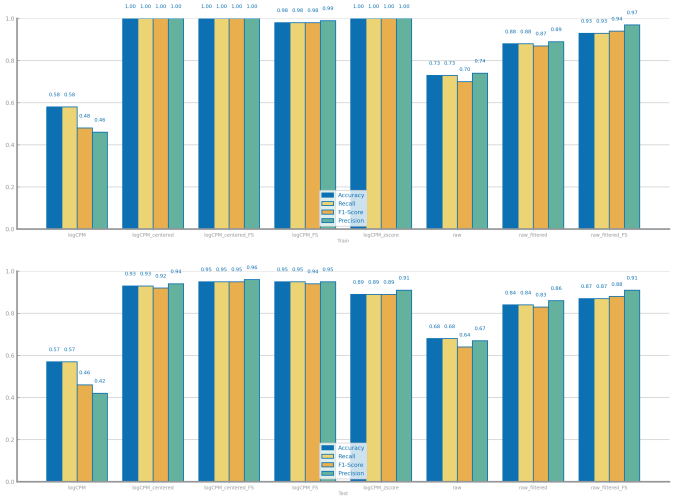


Fig. 24: Average performance measures (across all 10 folds) of the MLP classifier in train (up) and test (down) sets, in the 8 datasets used.

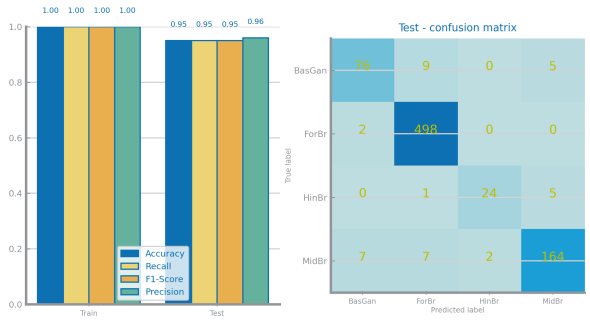


Fig. 25: Average performance measures (across all 10 folds) of the MLP classifier in the test set (left) and corresponding confusion matrix (right), for the logCPM_centered_FS dataset.

The classification results (isolated performance measures + confusion matrix) for the remaining datasets can be seen in Appendix E of this paper.

E. Overall Results

In the final phase of this project, the performance of the 4 classifiers regarding the task of classifying the 4 brain regions was compared with each other. A table with the results for the 4 score measures can be seen in Appendix F. The following plot (Fig. 26) presents a chart that allows to compare the performance of the four classifiers in each of the datasets used.

Indeed, all classifiers had in general a good performance, with SVM presenting the best results with an average performance around or above 0.97. DTs also obtained very consistent results throughout the datasets, with an average performance score of

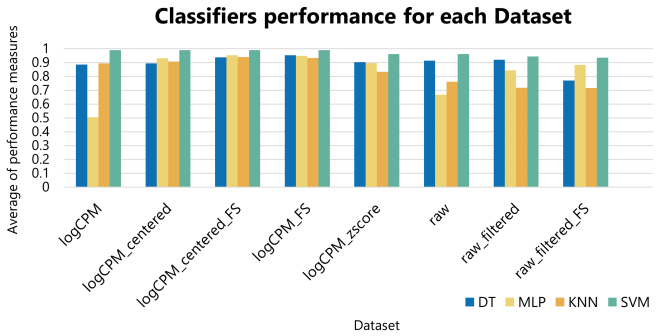


Fig. 26: Average of performance scores (across all 10 folds) of the four classifiers in the 8 datasets used.

0.90, except in the raw_filtered_fs dataset. When compared to DTs and in datasets with some data processing, namely normalization and filtering, MLP and KNN had better global performance with overall average scores around 0.83 and 0.84 but presented a significant decrease in the average scores for the raw datasets.

Regarding the comparison of the accuracy results with the study done by *Kim et al.* who used these classifiers to classify microarray data, it is possible, in the table below, to compare the average accuracy values of the classifiers for all the datasets studied in our work, and the average accuracy obtained by *Kim et al.* for their six datasets [5].

Table 3: Comparison of the average accuracy for all datasets of the four classifiers vs Kim et.al average accuracy results

Classifier	Average Accuracy	Average Accuracy (Kim et al.)
DT	0.89	0.66
MLP	0.84	0.74
KNN	0.84	0.79
SVM	0.97	0.79

Observing the table above, it is clear that all classifiers obtained a better accuracy than those obtained by the authors of the article, especially the DT and the SVM whose accuracy increased by 20%. These results may be explained by the fact that RNA-seq provides data that is more 'friendly' to classifiers instead of microarrays, as well as the data preparation and FS techniques behind said type of data. This results in a greater difference in accuracy compared to the values obtained by *Kim et al.*

In fact, in the two graphs below, it can even be seen that the overall performance of the classifiers has improved with the implementation of data processing. Both MLP and KNN performance results show improvements (of 20% and 12% on average, respectively) on datasets that have been filtered, normalized and centered in relation to its performance for raw data (see Figure 27). In the case of SVM and DT, processing alone does not seem to produce a significant performance improvement, but the same does not happen when feature selection is executed (see Figure 28). In the case of

DTs there is an improvement around 5% in the performance of this classifier when a simple threshold variance feature selection method is applied. It is important to highlight that this FS method also leads to significant improvements in MLP performance, especially in the logCPM_FS dataset.

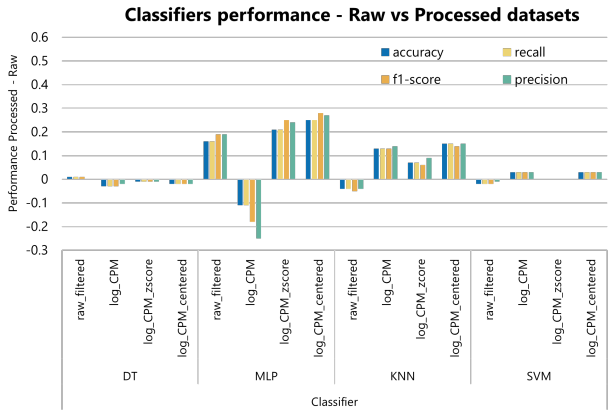


Fig. 27: Differences in the performance scores of the four classifiers between raw and processed datasets (the y-axis represents the difference between the performance score obtain for a processed dataset vs the performance score of the raw dataset)

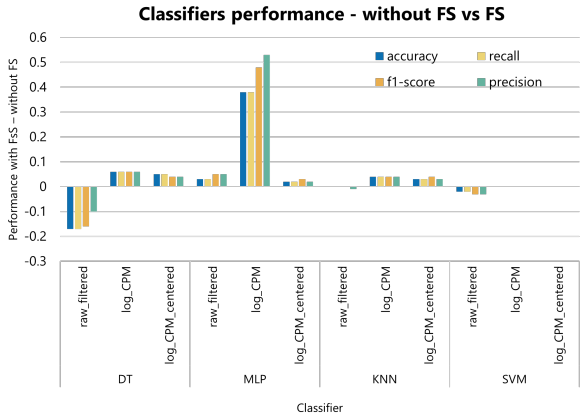


Fig. 28: Differences in the performance scores of the four classifiers between FS datasets and non FS datasets (the y-axis represents the difference between the performance score obtain for a FS dataset vs the performance score of the same dataset without the performance of FS)

IV. Conclusions

RNA-seq data proved to be a good source for classifying brain regions and demonstrates its potential in the bioinformatics field. MLP and KNN are better for microarray data, hence only having a good performance in the RNA-Seq log transformed data sets. In all datasets, SVM and DT showed strong performance, distinctively in the raw data, and in particular in the first classifier, which had all performance metrics over 0.90 and an average of 0.97.

As expected, our limitations include the unbalanced data, the small number of samples relative to the number of genes, and the use of only four classification algorithms. Future research might beneficially explore other types of SVM with further optimization, since it was the best performing algorithm in our study. Random forest, another algorithm which was frequently discussed in the literature, but that wasn't employed, also has the potential to be a strong classifier in this context. Additionally, classification of the samples into 25 sub brain regions instead of the 4 bigger ones (as it was in the original data) might bring as well more interesting results in the biological context that the data is inserted into, despite possibly leading to worsen results, since one has more unique values regarding the class. Finally, despite not being the focus of this project, other feature selection methods could be used, such as correlation, fisher ratio or even wrapper and embedded methods, which could lead to the discovery of subsets of genes with a significant role in classification.

V. Acknowledgments

The group elements thank Sofia Agostinho for all the support given throughout the project, and professor Ana Fred for the constructive critics when discussing the work.

References

[1] H. Zheng, Y. Feng, J. Tang, and S. Ma, "Interfacing brain organoids with precision medicine and machine learning," *Cell Reports Physical Science*, vol. 3, p. 100974, 07 2022.

[2] Y. Tanaka, B. Cakir, Y. Xiang, G. J. Sullivan, and I.-H. Park, "Synthetic analyses of single-cell transcriptomes from multiple brain organoids and fetal brain," *Cell Reports*, vol. 30, pp. 1682–1689.e3, 02 2020.

[3] P. Dong, J. Bendl, R. Misir, Z. Shao, J. Edelstien, D. A. Davis, V. Haroutunian, W. K. Scott, S. Acker, N. Lawless, G. E. Hoffman, J. F. Fullard, and P. Roussos, "Transcriptome and chromatin accessibility landscapes across 25 distinct human brain regions expand the susceptibility gene set for neuropsychiatric disorders," *bioRxiv*, 2022. [Online]. Available: https://www.biorxiv.org/content/early/2022/09/04/2022.09.02.506419

[4] G. Zararsız, D. Goksuluk, S. Korkmaz, V. Eldem, G. E. Zararsiz, I. P. Duru, and A. Ozturk, "A comprehensive simulation study on classification of rna-seq data," *PLOS ONE*, vol. 12, p. e0182507, 8 2017.

[5] J. Kim, Y. Yoon, H.-J. Park, and Y.-H. Kim, "Comparative study of classification algorithms for various dna microarray data," *Genes*, vol. 13, no. 3, 2022. [Online]. Available: https://www.mdpi.com/2073-4425/13/3/494

[6] S. R. Piccolo, A. Mecham, N. P. Golightly, J. L. Johnson, and D. B. Miller, "The ability to classify patients based on gene-expression data varies by algorithm and performance metric," *PLOS Computational Biology*, vol. 18, no. 3, pp. 1–34, 03 2022. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1009926

[7] "1.10. decision trees." [Online]. Available: https://scikit-learn.org/stable/modules/tree.html

[8] S. learn, "3.1. cross-validation: evaluating estimator performance — scikit-learn 0.22.2 documentation," Scikit-learn.org, 2013. [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

[9] J. Adamczyk, "k nearest neighbors computational complexity," Medium, 09 2020. [Online]. Available: https://towardsdatascience.com/k-nearest-neighbors-computational-complexity-502d2c440d5

[10] I. Mierswa, "K-nearest neighbors: A simple machine learning algorithm," RapidMiner, 05 2017. [Online]. Available: https://rapidminer.com/blog/k-nearest-neighbors-laziest-machine-learning-technique/

[11] R. M. Parry, W. Jones, T. H. Stokes, J. H. Phan, R. A. Moffitt, H. Fang, L. Shi, A. Oberthuer, M. Fischer, W. Tong, and M. D. Wang, "k-nearest neighbor models for microarray gene expression analysis and clinical outcome prediction," *The Pharmacogenomics Journal*, vol. 10, pp. 292–309, 07 2010.

[12] H. M. d. Silva and A. S. Perera, "Evolutionary k- nearest neighbor imputation algorithm for gene expression data," *The International Journal on Advances in ICT for Emerging Regions*, vol. 10, 06 2017. [Online]. Available: https://journal.icter.org/index.php/ICTer/article/view/224

[13] W. M. Czarnecki, S. Podlewska, and A. J. Bojarski, "Robust optimization of svm hyperparameters in the classification of bioactive compounds," *Journal of Cheminformatics*, vol. 7, p. 38, 12 2015.

[14] C.-w. Hsu, C.-c. Chang, and C.-J. Lin, "A practical guide to support vector classification," 11 2003.

[15] "Rbf svm parameters." [Online]. Available: https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[16] "The kernel trick in support vector classification." [Online]. Available: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

[17] X. Chen, M. Wang, and H. Zhang, "The use of classification trees for bioinformatics," *Wiley interdisciplinary reviews. Data mining and knowledge discovery*, vol. 1, pp. 55–63, 01 2011.

[18] I. Polaka, I. Tom, and A. Borisov, "Decision tree classifiers in bioinformatics," *J. Riga Technical University*, vol. 42, pp. 118–123, 01 2010.

[19] D. Grattarola, "Deep feature extraction for sample-efficient reinforcement learning," Ph.D. dissertation, POLITECNICO DI MILANO, 10 2017.

APPENDIX

A. Data Preparation

	GeneID	BB3_ARC_G_RNA	BB5_PVC_G_RNA	BB5_PVC_N_RNA	BB3_PAC_G_RNA	BB3_PAC_N_RNA	BB4_EC_G_RNA	BB4_EC_N_RNA	BB6_MDT_G_RNA	BB6_MDT_N_RNA	...
0	ENSG00000000003	173.0	70.0	15.0	213.0	44.0	132.0	22.0	20.0	3.0	...
1	ENSG00000000005	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...
2	ENSG00000000419	33.0	105.0	114.0	71.0	161.0	52.0	127.0	21.0	37.0	...
3	ENSG00000000457	182.0	250.0	273.0	336.0	327.0	242.0	208.0	194.0	102.0	...
4	ENSG00000000460	79.0	147.0	74.0	98.0	52.0	134.0	33.0	49.0	17.0	...

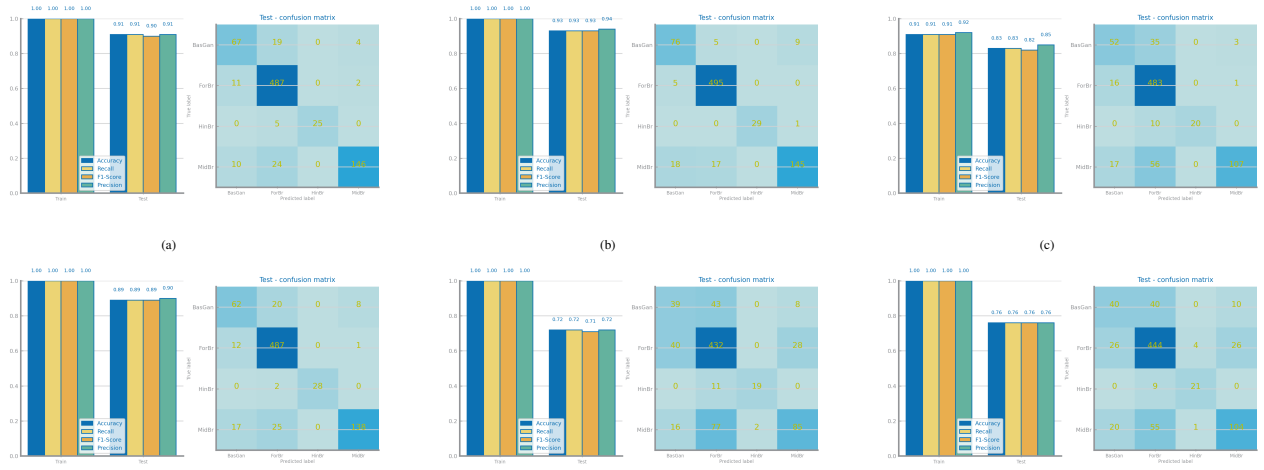
(a)

	ENSG00000000003	ENSG00000000005	ENSG00000000419	ENSG00000000457	ENSG00000000460	ENSG00000000938	ENSG00000000971	ENSG00000001036	...	BRAIN REGION
0	3.974968	-2.068913	1.674641	4.047052	2.869907	1.082836	2.555303	3.334169	...	ARC
1	2.192139	-2.068913	2.7518	3.973422	3.222567	0.561601	2.973943	2.71148	...	PVC
2	0.480329	-2.068913	3.174636	4.412151	2.571638	-2.068913	1.628349	1.591906	...	PVC
3	3.680954	-2.068913	2.148638	4.328714	2.592076	-2.068913	1.648004	1.919929	...	PAC
4	1.612079	-2.068913	3.399415	4.405006	1.835677	0.099095	-2.068913	2.21983	...	PAC

(b)

Fig. 29: Example of the original (*Raw*) (a) and structurized (b) datasets structures (head of the dataframes only). Values in the dataframes' cells are not to be considered. In (a), regarding column names, `BB` stands for brain bank, followed by the number of the individual from where the sample was taken, ranging from 1 to 6; then the brain region is identified, out of 25 possible regions, followed by the type of sample - `G` if it was taken from the glia cells and `N` if it was taken from the neuron cells - and the type of nucleic acid - DNA or RNA (in our case RNA, since the data represent RNA-seq counts). A detailed methodology regarding the acquisition of this data can be seen in *Dong et al.* [3].

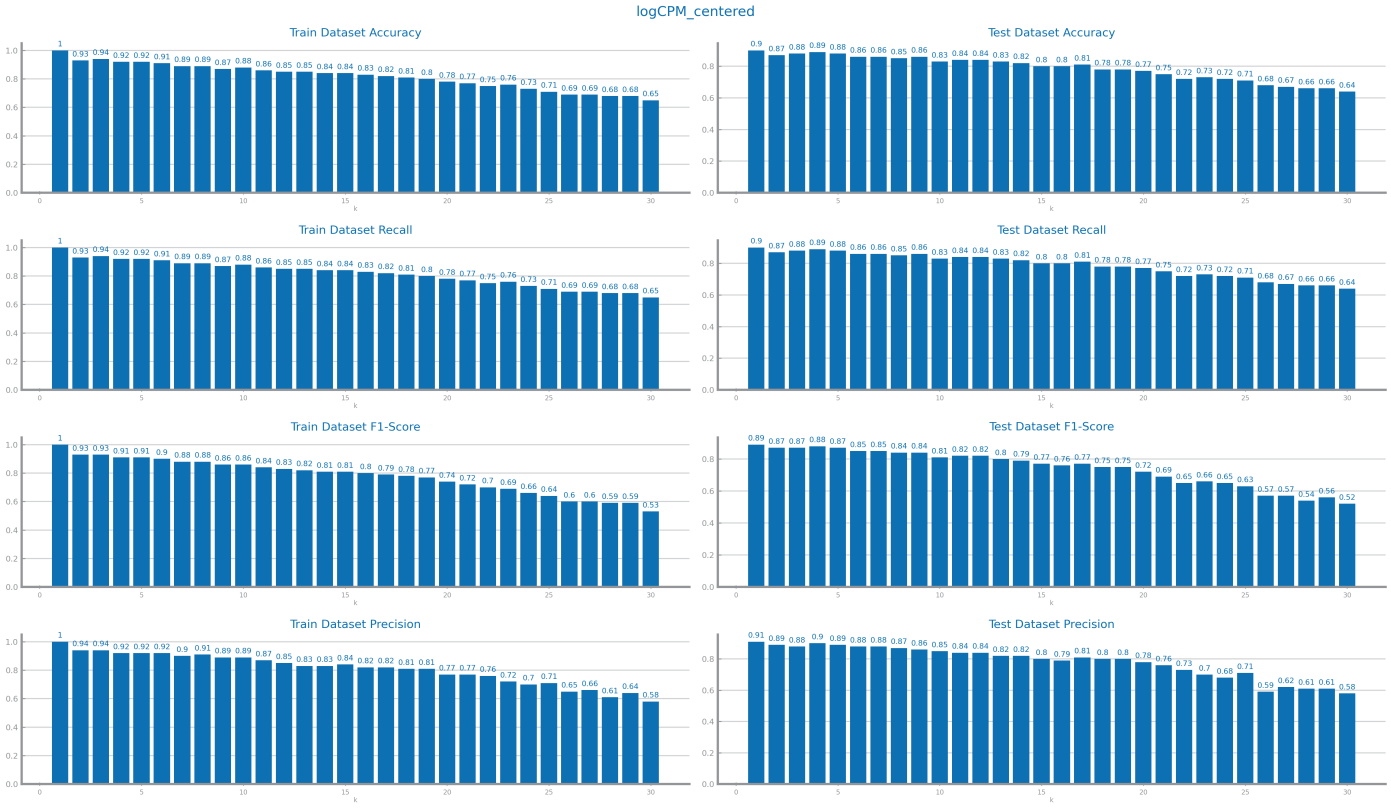
B. K-Nearest Neighbors



(a) (b) (c)

(d) (e) (f)

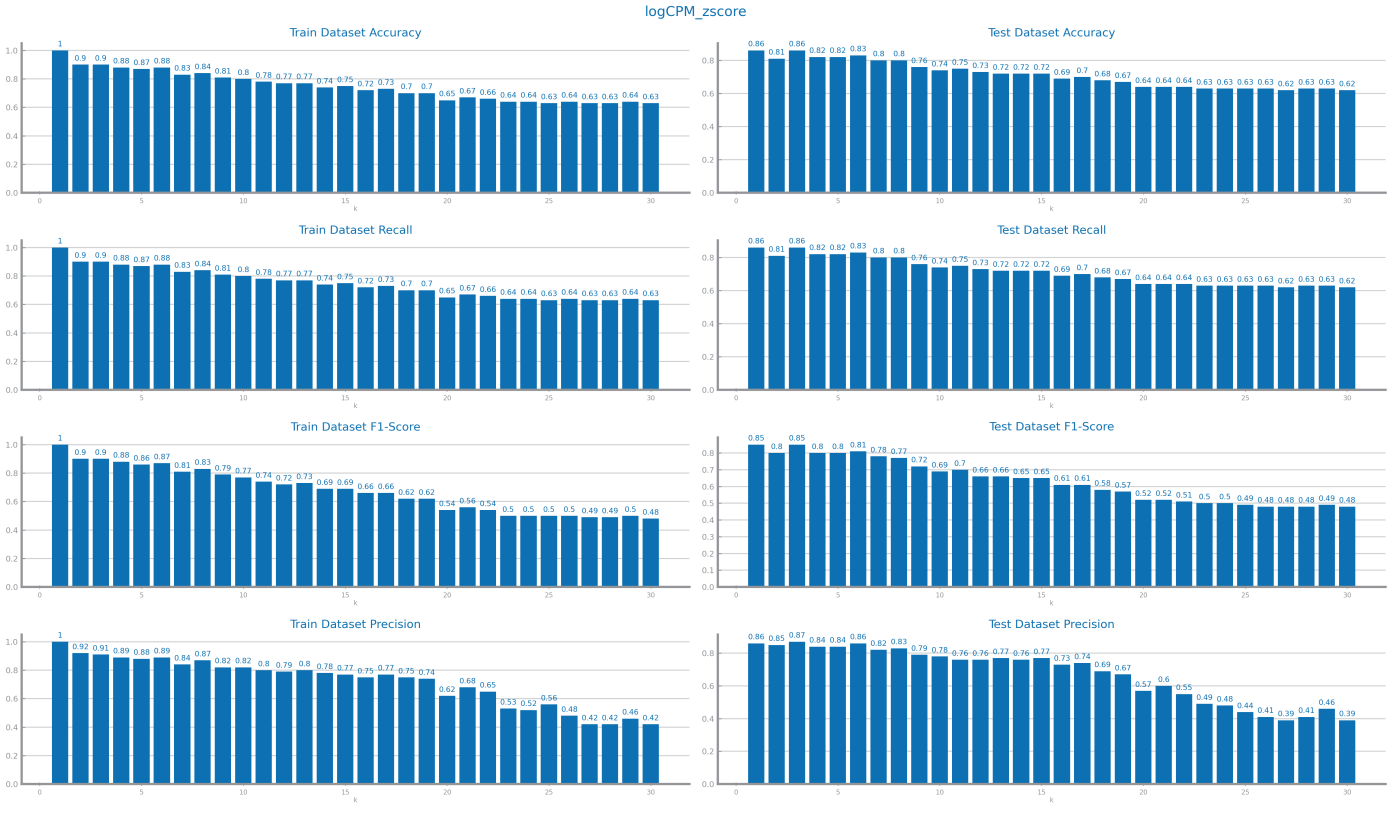
Fig. 30: Average performance measures (across all 10 folds) of the KNN classifier in the test set and corresponding confusion matrix, for the (a) logCPM_centered (b) logCPM_centered_FS (c) logCPM_zscore (d) logCPM (e) raw_filtered (f) raw.



(a)

(b)

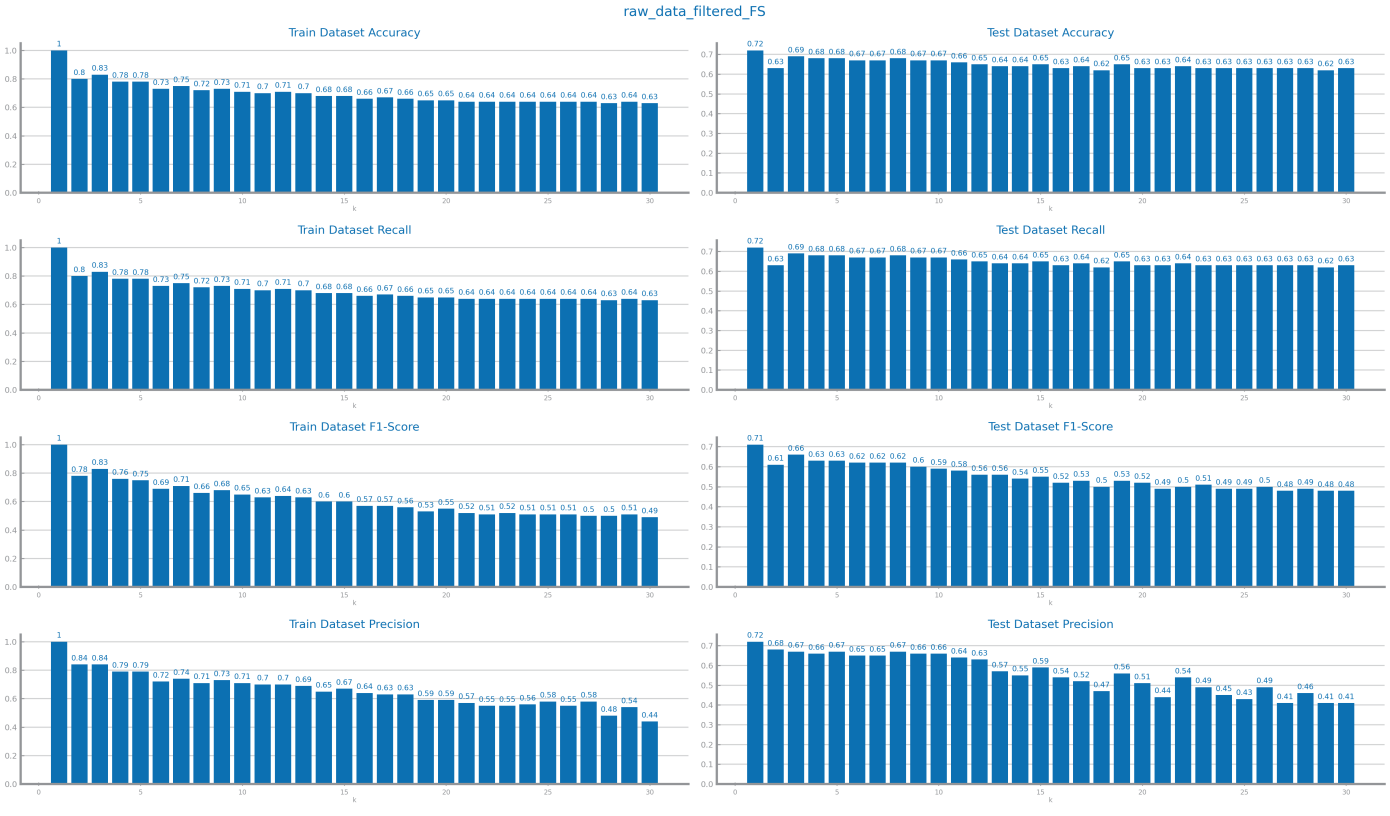
Fig. 31: Performance measures across k values from 1 to 30, for the (a) logCPM_centered and (b) logCPM_FS datasets.



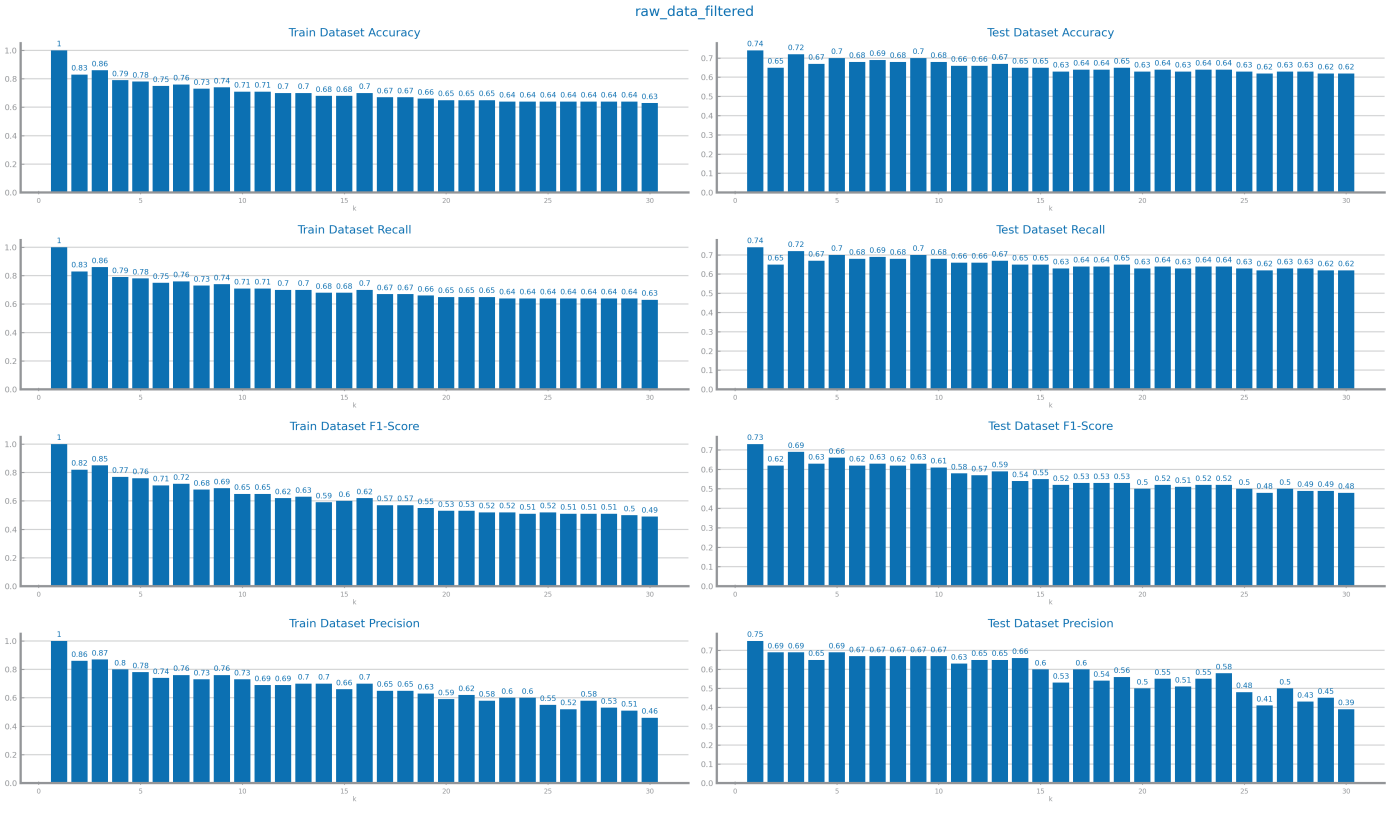
(a)

(b)

Fig. 32: Performance measures across k values from 1 to 30, for the (a) logCPM_zscore and (b) logCPM datasets.



(a)

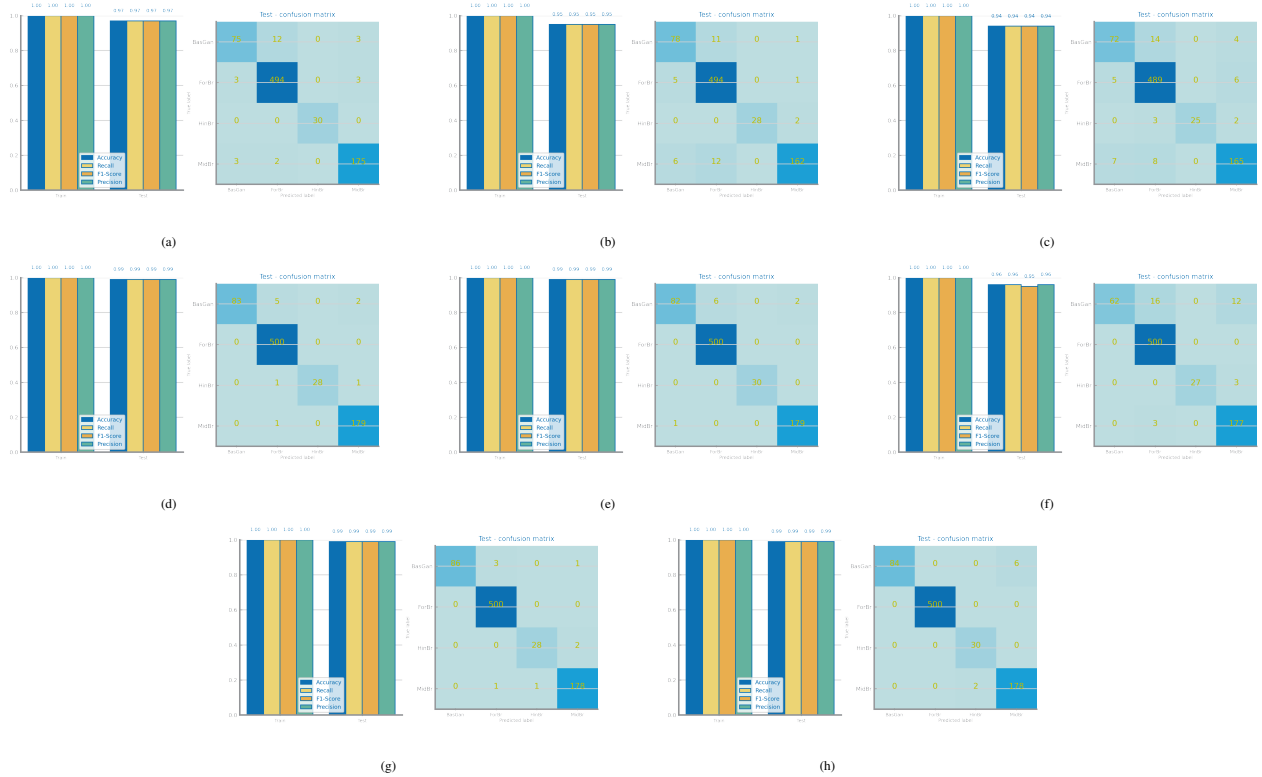


(b)

Fig. 33: Performance measures across k values from 1 to 30, for the (a) raw_data_centered and (b) raw_data_filtered.

Fig. 34: Performance measures across k values from 1 to 30, for the raw_data dataset.

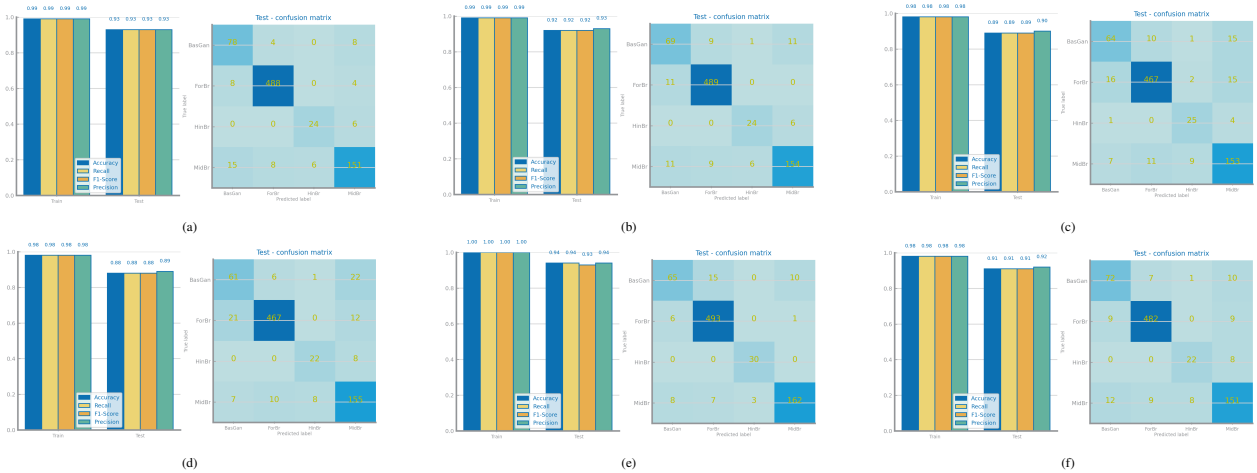
C. Support Vector Machines



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. 35: Average performance measures (across all 10 folds) of the SVM classifier in the test set and corresponding confusion matrix, for the (a) raw (b) raw_filtered (c) raw_filtered_FS (d) logCPM (e) logCPM_centered (f) logCPM_zscore (g) logCPM_FS (f) logCPM_centered_FS .

D. Decision Trees



(a) (b) (c) (d) (e) (f)

Fig. 36: Average performance measures (across all 10 folds) of the DT classifier in the test set and corresponding confusion matrix, for the (a) raw (b) raw_filtered (c) logCPM (d) logCPM_centered (e) logCPM_centered_FS (f) logCPM_zscore.

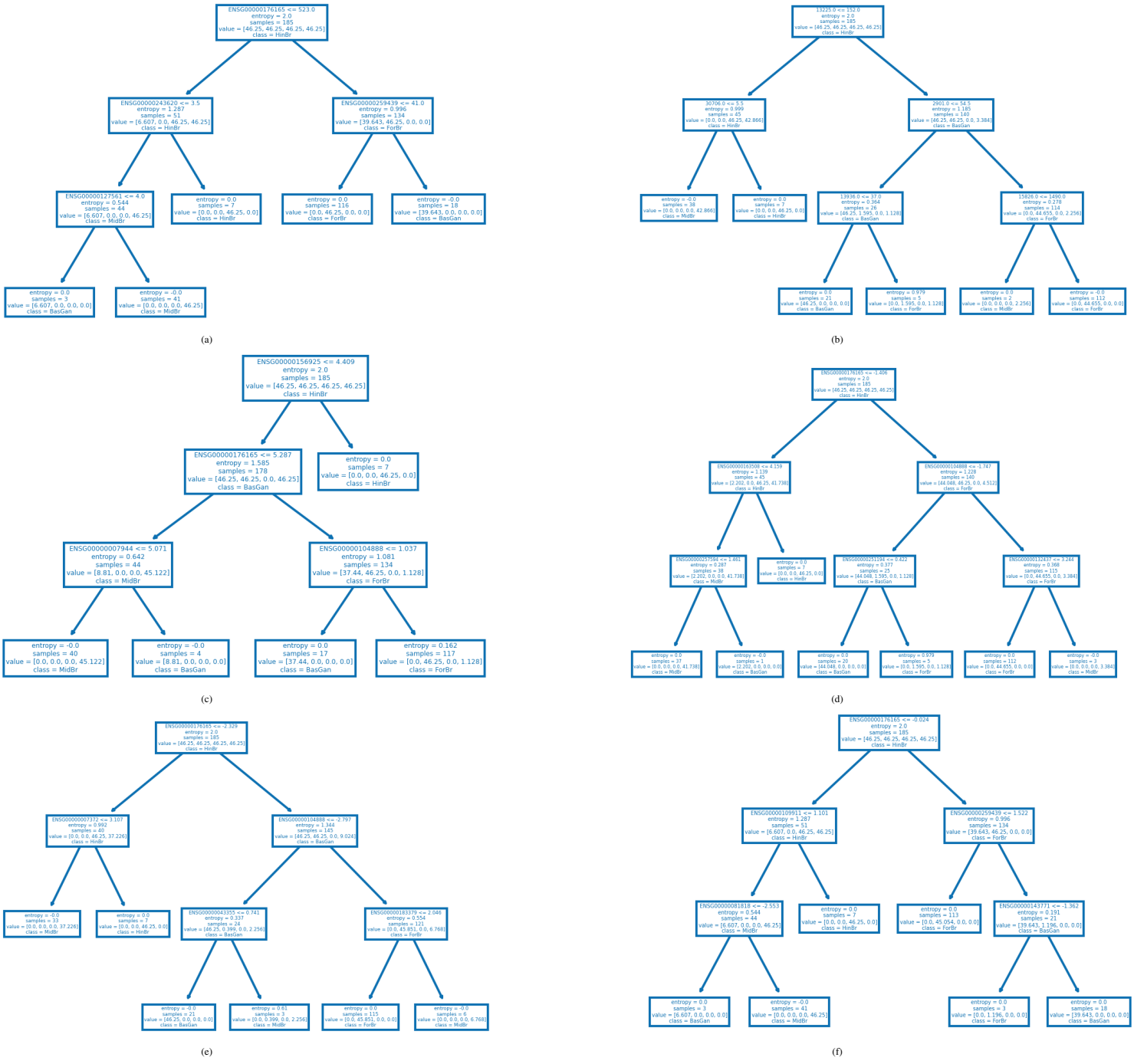
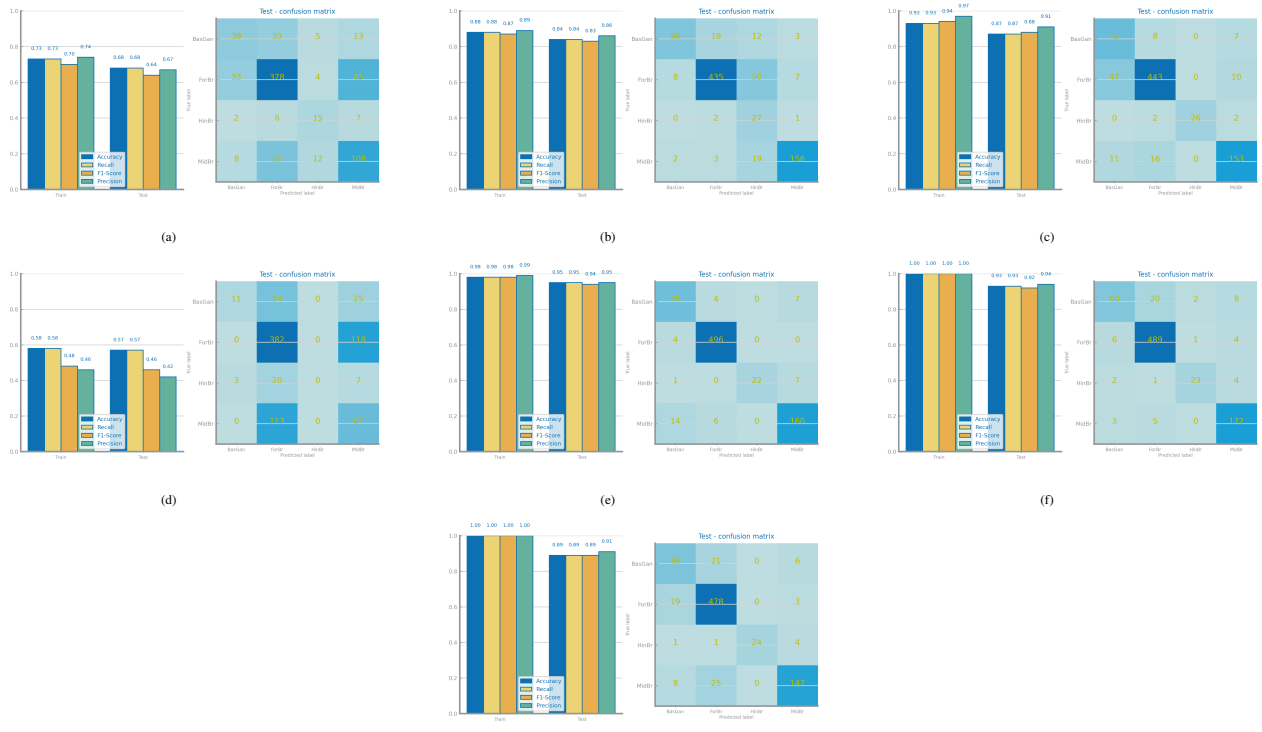


Fig. 37: Diagram of the trees built by the DT classifier, for the (a) raw (b) raw_filtered (c) logCPM (d) logCPM_centered (e) logCPM_centered_FS (f) logCPM_zscore.

E. Multi Layer Perceptrons



(a) (b) (c)

(d) (e) (f)

(g)

Fig. 38: Average performance measures (across all 10 folds) of the MLP classifier in the test set and corresponding confusion matrix, for the (a) raw (b) raw_filtered (c) raw_filtered_FS (d) logCPM (e) logCPM_FS (f) logCPM_centered (g) logCPM_zscore.

F. Overall Results

Table 4: Average performance measures values for each dataset and for each classifier, as well as perfomance measures' averages for each classifier, followed by the classifier average performance.

Classifier	Performance Measure	Test Dataset								Measure Average	Classifier Average
		logCPM	logCPM_centered	logCPM_centered_FS	logCPM_FS	logCPM_zscore	raw	raw_filtered	raw_filtered_FS		
DT	Accuracy	0.88	0.89	0.94	0.95	0.90	0.91	0.92	0.75	0.89	0.90
	Recall	0.88	0.89	0.94	0.95	0.90	0.91	0.92	0.75	0.89	
	F1-score	0.88	0.89	0.93	0.95	0.90	0.91	0.92	0.76	0.89	
	Precison	0.9	0.90	0.94	0.96	0.91	0.92	0.92	0.82	0.91	
MLP	Accuracy	0.57	0.93	0.95	0.95	0.89	0.68	0.84	0.87	0.84	0.83
	Recall	0.57	0.93	0.95	0.95	0.89	0.68	0.84	0.87	0.84	
	F1-score	0.46	0.92	0.95	0.94	0.89	0.64	0.83	0.88	0.81	
	Precison	0.42	0.94	0.96	0.95	0.91	0.67	0.86	0.91	0.83	
KNN	Accuracy	0.89	0.91	0.94	0.93	0.83	0.76	0.72	0.72	0.84	0.84
	Recall	0.89	0.91	0.94	0.93	0.83	0.76	0.72	0.72	0.84	
	F1-score	0.89	0.90	0.94	0.93	0.82	0.76	0.71	0.71	0.83	
	Precison	0.90	0.91	0.94	0.94	0.85	0.76	0.72	0.71	0.84	
SVM	Accuracy	0.99	0.99	0.99	0.99	0.96	0.97	0.95	0.94	0.98	0.98
	Recall	0.99	0.99	0.99	0.99	0.96	0.97	0.95	0.94	0.98	
	F1-score	0.99	0.99	0.99	0.99	0.95	0.97	0.95	0.94	0.98	
	Precison	0.99	0.99	0.99	0.99	0.96	0.97	0.95	0.94	0.98	