Application of **Scalable ML Techniques**

TEAA Lab Report — Degree in **Data Science**
Course 2024/2025
Eusebio Monzó, Adrián Rico, Juan Tomás, Marc Vicedo (Group 9)

Content Index

Table/Figure Index

Introduction

This report focuses on the application of scalable Machine Learning techniques using two datasets: DIGITS and CHBMIT. The first dataset is widely used in image processing tasks, especially for training and testing models for handwritten digit classification. Its versatility makes it a key resource for evaluating and developing algorithms in the field of Machine Learning. On the other hand, the second dataset is designed for the analysis of electroencephalography (EEG) signals and is employed in this study to develop Machine Learning classifiers for the detection and early prediction of epileptic seizures. Two types of models were designed and implemented:

1. A **binary classifier**, aimed at detecting the presence of a seizure immediately after it occurs.
2. A **multi-class classifier**, capable of predicting the onset of a seizure up to 60 minutes in advance.

To achieve the multi-class classification, the inter-ictal and ictal periods in the EEG recordings were labeled. Specifically, six classes were defined to describe different temporal stages relative to the seizure events:

- ➔ **Class 0 (Ictal):** Corresponds to the period during the seizure.
- ➔ **Class 1 (Pre-ictal):** From 10 minutes to 2 seconds before the seizure.
- ➔ **Class 2 (Pre-ictal):** From 20 minutes to 10 minutes before the seizure.
- ➔ **Class 3 (Inter-ictal):** More than 20 minutes before the seizure.
- ➔ **Class 4 (Post-ictal):** Up to 10 seconds after the seizure.
- ➔ **Class 5 (Post-ictal):** From 10 seconds to 20 minutes after the seizure.

This segmentation enables the training of predictive models capable of identifying patterns not only during seizure events but also in the critical minutes leading up to them.

In addition to developing these models, we conduct a comparative analysis of the performance and parameter optimization of the different Machine Learning techniques. To facilitate this, we have used Python libraries such as Matplotlib [8] for data visualization and Pandas [9] for data manipulation and preprocessing. These tools allow for a clear representation and interpretation of results, which we wanted to rely on simple, easy-to-interpret graphics.

Our approach

In this section, we detail the ML models used in this report, as well as the specific configurations and parameters for each. Prior to applying each model, a principal component analysis (**PCA**) was applied with different configurations to enhance model performance by reducing the dimensionality of the input data [1]. To evaluate the models, we used the **F1-macro score metric**, which measures the balance between precision and recall, providing a robust metric in scenarios with imbalanced classes. This allows us to objectively compare the performance of the models under different configurations and parameters.

For each technique, we provide tables indicating all the executions/experiments run as a summary for both datasets. However, for tree-based ensemble techniques, more than fifty experiments were made, so we just included the first rows of the table for simplicity.

These are the four technical approaches detailed in the report:

1. Clustering/Density	2. Tree-based	3. Memory-based	4. Neural Networking
K-means Clustering	*Random Forest*	*Kernel Density Estimator*	*Convolutional NNs*
Gaussian Mixture Model	*Extra Trees*	*K-nearest Neighbours*	*Feed-forward NNs*
*	*Gradient-boosted trees*	*	*

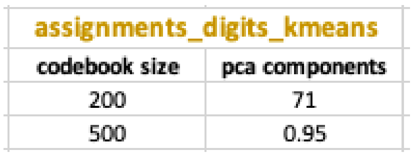
TABLE 1. Technical approaches

1. Clustering/Density

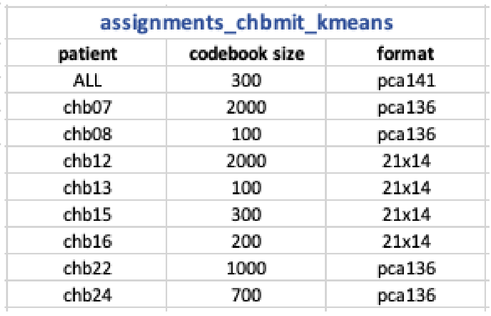
1.1. K-means Clustering

The K-means machine learning model uses a strategy based on grouping samples into a predefined number of clusters, called codebooks, according to their shared characteristics in the data space [6]. This unsupervised learning model organizes the samples into groups that minimize variance within each cluster, leveraging the spatial distances between points to uncover underlying patterns in the data. In the digit images dataset, K-means is used to cluster digit images into groups that represent similar visual patterns. For EEG data, K-means is employed to analyze patterns related to the detection of epileptic seizures.

In the implementation of this technique, we tested different "codebook" sizes, which correspond to the number of clusters into which the data are grouped or the number of centroids in the codebook. Regarding the codebook size range, if too few clusters are used, distinct data points may end up in the same cluster, leading to the loss of critical information. However, using too many clusters could result in overfitting.



assignments_digits_kmeans	
codebook size	pca components
200	71
500	0.95



assignments_chbmit_kmeans		
patient	codebook size	format
ALL	300	pca141
chb07	2000	pca136
chb08	100	pca136
chb12	2000	21x14
chb13	100	21x14
chb15	300	21x14
chb16	200	21x14
chb22	1000	pca136
chb24	700	pca136

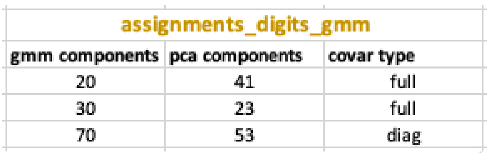
TABLES 2a, 2b. Experiments run on DIGITS (a) and CHBMIT (b) datasets using K-means

1.2. Gaussian Mixture Model (GMM)

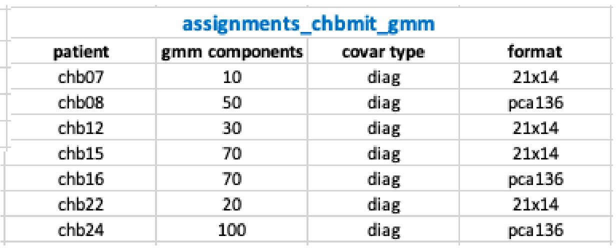
GMM is a probabilistic model that assumes data is distributed as a mixture of Gaussian distributions, using a combination of these to model the probability that data points belong to certain categories or clusters [1].

When applying this model to the DIGITS dataset, each sample is represented as a vector in a vector space, with as many dimensions as components established during the PCA process. The model then fits K Gaussians, each representing a class in the set of digits. To classify each sample, GMM calculates the probabilities of the sample belonging to each component, assigning the digit to the class with the highest probability. Regarding the parameter *gmm_components*, it may seem intuitive to assume that if there are 10 classes (digits from 0 to 9), the optimal number of components, K, should equal the number of classes; this depends on how the data is distributed in the feature space. For example, data from a single class (e.g., the digit "8") could form multiple clusters in the feature space due to variations in how the digits are written. Thus, more than one Gaussian might be needed to fully model this class.

In the context of EEG analysis, GMM is well-suited for identifying patterns in brain activity data. Each sample represents the unique features of EEG signals, allowing GMM to model the temporal and spatial patterns of EEG activity through Gaussian components. Our goal is to classify brain activity into six possible states defined by the time relative to seizures. It might seem reasonable to start with one Gaussian per class, given that each class has a clear clinical significance. However, due to variability between subjects or even within a single subject, classes may overlap. Experimenting with more Gaussians could help capture the full variability within the classes, as explained in the previous paragraph.



assignments_digits_gmm		
gmm components	pca components	covar type
20	41	full
30	23	full
70	53	diag



assignments_chbmit_gmm			
patient	gmm components	covar type	format
chb07	10	diag	21x14
chb08	50	diag	pca136
chb12	30	diag	21x14
chb15	70	diag	21x14
chb16	70	diag	pca136
chb22	20	diag	21x14
chb24	100	diag	pca136

TABLES 3a, 3b. Experiments run on DIGITS (a) and CHBMIT (b) datasets using GMM

2. Tree-based

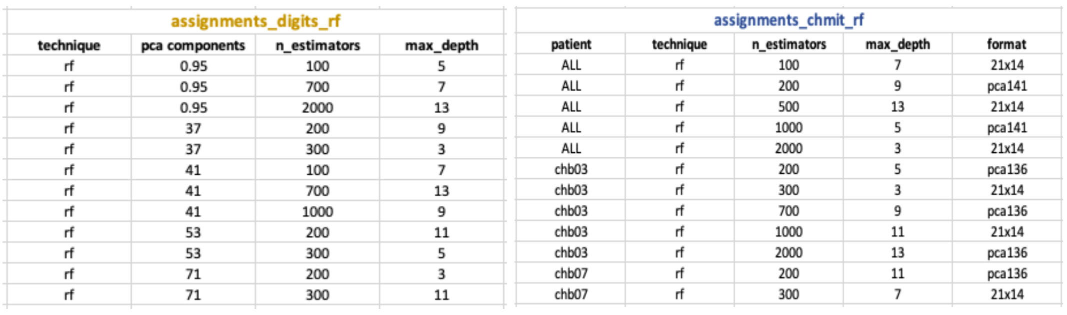
2.1. Random Forest

The Random Forest (RF) model is a machine learning method based on constructing multiple decision trees and combining their results with classification tasks [7].

Focusing on the method's approach when applied to the DIGITS dataset, each tree is trained on a random subset of images, with random subsets of pixels selected as features. Once the model is trained, each tree predicts a digit based on the splits created by the selected pixels. The final class is then decided by majority voting among the trees.

For the CHBMIT dataset, the trees aim to find splits that separate the classes based on the features of the signals. Given the high-dimensional nature of this data, preprocessing the signals and applying PCA enables the model to focus only on significant features. Additionally, the random selection of features at each node helps reduce noise. The final prediction is also based on majority voting, enhancing robustness against variations in the signals.

Focusing on the key parameters, *n_estimators* determines the number of trees participating in the voting process, allowing a trade-off between the desired accuracy and training time. Additionally, *max_depth* limits the depth of the trees, preventing them from memorizing irrelevant details in the data, whether signals or images. Without this limitation, the model could overfit, losing its ability to generalize to unseen data.



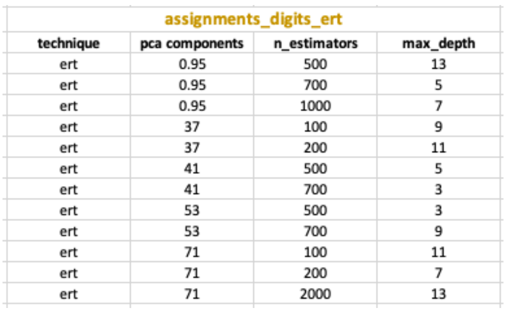
assignments_digits_rf			
technique	pca components	n_estimators	max_depth
rf	0.95	100	5
rf	0.95	700	7
rf	0.95	2000	13
rf	37	200	9
rf	37	300	3
rf	41	100	7
rf	41	700	13
rf	41	1000	9
rf	53	200	11
rf	53	300	5
rf	71	200	3
rf	71	300	11

assignments_chmit_rf				
patient	technique	n_estimators	max_depth	format
ALL	rf	100	7	21x14
ALL	rf	200	9	pca141
ALL	rf	500	13	21x14
ALL	rf	1000	5	pca141
ALL	rf	2000	3	21x14
chb03	rf	200	5	pca136
chb03	rf	300	3	21x14
chb03	rf	700	9	pca136
chb03	rf	1000	11	21x14
chb03	rf	2000	13	pca136
chb07	rf	200	11	pca136
chb07	rf	300	7	21x14

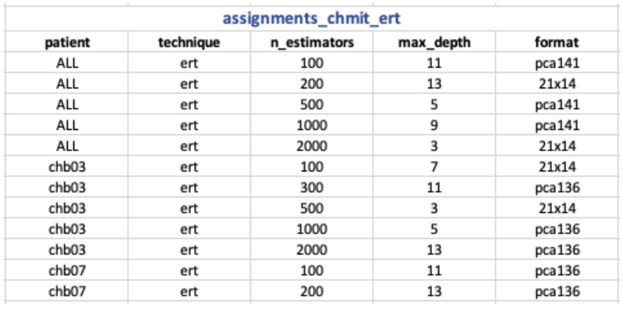
TABLES 4a, 4b. Extract of the experiments run on DIGITS (a) and CHBMIT (b) datasets using RF
Note: The tables in this section do not include all the experiments run, as some entries have been omitted for simplicity

2.2. Extremely Randomized Trees (ERT)

The Extra Trees model (Extremely Randomized Trees) is a variant of the previous model, which introduces greater randomness in the construction of decision trees. With this approach, we can improve generalization and speed up training by simplifying the selection of splits at the nodes [4]. The differences when comparing with the Random Forest model are two: Firstly, when generating the subsets of data to train the trees, bootstrap sampling is not always used; instead, the data can be used directly. Secondly, to establish the splits in the trees, Extra Trees selects the split point completely randomly within the range of the feature's values.



assignments_digits_ert			
technique	pca components	n_estimators	max_depth
ert	0.95	500	13
ert	0.95	700	5
ert	0.95	1000	7
ert	37	100	9
ert	37	200	11
ert	41	500	5
ert	41	700	3
ert	53	500	3
ert	53	700	9
ert	71	100	11
ert	71	200	7
ert	71	2000	13



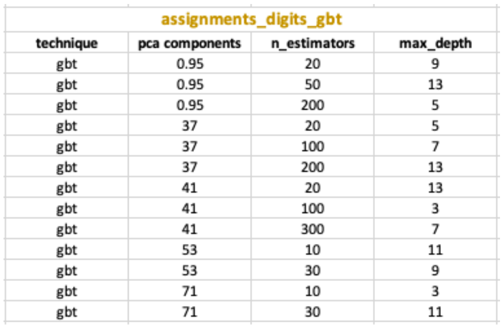
assignments_chmit_ert				
patient	technique	n_estimators	max_depth	format
ALL	ert	100	11	pca141
ALL	ert	200	13	21x14
ALL	ert	500	5	pca141
ALL	ert	1000	9	pca141
ALL	ert	2000	3	21x14
chb03	ert	100	7	21x14
chb03	ert	300	11	pca136
chb03	ert	500	3	21x14
chb03	ert	1000	5	pca136
chb03	ert	2000	13	pca136
chb07	ert	100	11	pca136
chb07	ert	200	13	pca136

TABLES 5a, 5b. Extract of the experiments run on DIGITS (a) and CHBMIT (b) datasets using ERT
Note: The tables in this section do not include all the experiments run, as some entries have been omitted for simplicity

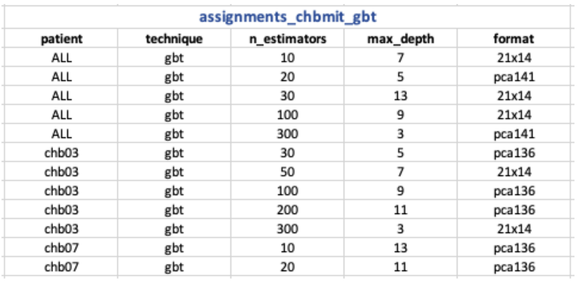
2.3. Gradient-boosted Trees (GBT)

This machine learning model is also based on decision trees, but in a way that trees are built sequentially to correct the errors of the previous ones [3]. To achieve this, at each step the model minimizes a specific loss function using gradient descent, thus adjusting the residuals (errors) of the model accumulated so far as the trees are trained. In the construction of the trees, a learning rate is also used to control the contribution of each tree to the final model, preventing a dominant tree from overly controlling the final prediction.

Then, in the case of multiclass classification, GBT trains a separate set of trees for each class, starting with an initial value for each class, such as the proportion of training samples belonging to that class. Afterwards, each tree produces a numerical value that represents how confident the model is that a sample belongs to that class. As predictions are made, the contributions of all trained trees are summed, and these numerical values are converted to probabilities, allowing the sample to be assigned to the class with the highest probability. Finally, regarding the parameters, we can highlight the *learning_rate*, which allows us to reduce the influence of each individual tree, promoting better generalization.



assignments_digits_gbt			
technique	pca components	n_estimators	max_depth
gbt	0.95	20	9
gbt	0.95	50	13
gbt	0.95	200	5
gbt	37	20	5
gbt	37	100	7
gbt	37	200	13
gbt	41	20	13
gbt	41	100	3
gbt	41	300	7
gbt	53	10	11
gbt	53	30	9
gbt	71	10	3
gbt	71	30	11



assignments_chbmit_gbt				
patient	technique	n_estimators	max_depth	format
ALL	gbt	10	7	21x14
ALL	gbt	20	5	pca141
ALL	gbt	30	13	21x14
ALL	gbt	100	9	21x14
ALL	gbt	300	3	pca141
chb03	gbt	30	5	pca136
chb03	gbt	50	7	21x14
chb03	gbt	100	9	pca136
chb03	gbt	200	11	pca136
chb03	gbt	300	3	21x14
chb07	gbt	10	13	pca136
chb07	gbt	20	11	pca136

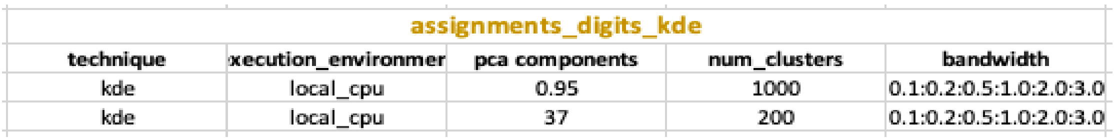
TABLES 6a, 6b. Extract of the experiments run on DIGITS (a) and CHBMIT (b) datasets using GBT
Note: The tables in this section do not include all the experiments run, as some entries have been omitted for simplicity

3. Memory-based

3.1. Kernel Density Estimation (KDE)

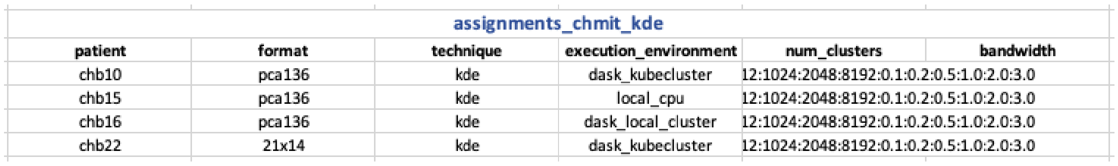
This model is based on estimating probability densities to model the distribution of the data. Instead of assuming a specific form for the distribution (such as a normal distribution), KDE uses a non-parametric approach that fits the data by combining kernels centered on the data points [11]. To make predictions, KDE calculates the probability density for each class independently. These densities reflect how likely it is that a point x belongs to a specific class. This means that, for each class, the model generates a density function that summarizes how the data is distributed in the feature space, which are then combined during prediction to determine which class has the highest probability for the input sample.

Regarding the key parameters, *bandwidth* controls the smoothness of the density estimation. A small value allows the model to fit very closely to the data, capturing fine details, but it may overfit. In contrast, a larger value generalizes better, reducing the risk of overfitting but sacrificing some details. Also, *kmeans_codebook_size* helps simplify the calculation by using a reduced number of centroids, obtained via K-means clustering, to represent each class. This is particularly useful for handling large datasets, where calculating the density for each sample would be computationally expensive.



assignments_digits_kde

technique	xecution_environmer	pca components	num_clusters	bandwidth
kde	local_cpu	0.95	1000	0.1:0.2:0.5:1.0:2.0:3.0
kde	local_cpu	37	200	0.1:0.2:0.5:1.0:2.0:3.0



assignments_chmit_kde

patient	format	technique	execution_environment	num_clusters	bandwidth
chb10	pca136	kde	dask_kubecluster	12:1024:2048:8192:0.1:0.2:0.5:1.0:2.0:3.0	
chb15	pca136	kde	local_cpu	12:1024:2048:8192:0.1:0.2:0.5:1.0:2.0:3.0	
chb16	pca136	kde	dask_local_cluster	12:1024:2048:8192:0.1:0.2:0.5:1.0:2.0:3.0	
chb22	21x14	kde	dask_kubecluster	12:1024:2048:8192:0.1:0.2:0.5:1.0:2.0:3.0	

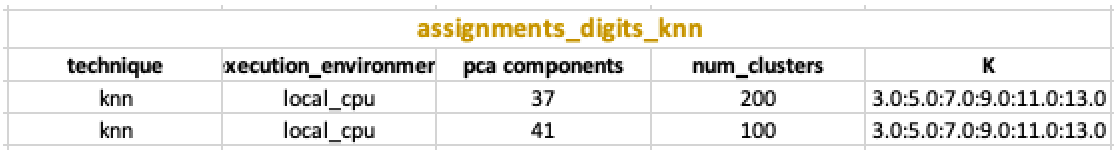
TABLES 7a, 7b. Experiments run on DIGITS (a) and CHBMIT (b) datasets using KDE

3.2. K-nearest Neighbours (KNN)

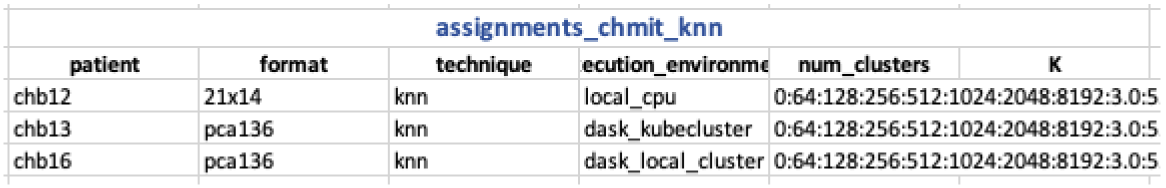
This machine learning model uses a strategy based on classifying a sample according to the classes of its K nearest values in the feature space. This is also a non-parametric model that does not assume a specific form for the data distribution, but instead directly uses the spatial relationships between the samples to make predictions [2].

In the case of KNN, when a new sample needs to be classified, the model looks for the K closest samples to it in the dataset. Then, it assigns the most frequent class among these K neighbors as the final prediction. Instead of looking for global patterns, KNN works locally, making decisions based on the closest samples. For multiclass classification problems, KNN handles multiple classes directly, as it simply considers the proportion of nearby neighbors belonging to each class. The class with the most representation among the neighbors defines the prediction.

Regarding key parameters, we have *K*, which controls the number of neighbors used for classification. A small value makes the model very sensitive to individual points, which can lead to overfitting. On the other hand, a larger *K* smooths the predictions by considering a broader context, but it may dilute the influence of relevant samples.



assignments_digits_knn				
technique	xecution_environmer	pca components	num_clusters	K
knn	local_cpu	37	200	3.0:5.0:7.0:9.0:11.0:13.0
knn	local_cpu	41	100	3.0:5.0:7.0:9.0:11.0:13.0



assignments_chmit_knn					
patient	format	technique	ecution_environme	num_clusters	K
chb12	21x14	knn	local_cpu	0:64:128:256:512:1024:2048:8192:3.0:5	
chb13	pca136	knn	dask_kubecluster	0:64:128:256:512:1024:2048:8192:3.0:5	
chb16	pca136	knn	dask_local_cluster	0:64:128:256:512:1024:2048:8192:3.0:5	

TABLES 8a, 8b. Experiments run on DIGITS (a) and CHBMIT (b) datasets using KNN

---

4. Neural Networking

4.1. Feed-forward Neural Networks

The feed-forward neural network model is a machine learning technique based on architectures composed of fully connected dense layers. In these networks, information flows in a single direction: from the input layer, through one or more hidden layers, to the output layer. Each neuron in one layer is connected to all neurons in the next layer, allowing the network to capture complex relationships between the input features and the target task [5].

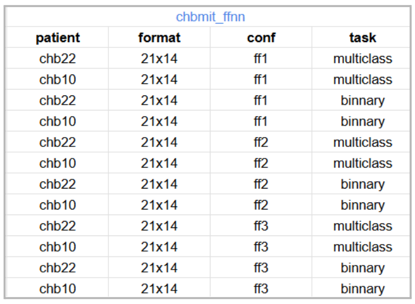
The operation of FFNNs is based on iteratively transforming data representations at each layer. The hidden layers use non-linear activation functions, such as ReLU, which allow the network to learn complex and non-linear patterns in the data.

One of the peculiarities of feed-forward networks is the ability to adjust their capacity and behavior through various mechanisms. For example, we can vary the depth of the network, as the more layers the network has, the greater its ability to model complex patterns, although an excessive number of layers can increase the risk of problems such as vanishing gradients and overfitting. Additionally, the size of the layers determines the amount of information that each layer can process and retain. An insufficient size may limit the learning capacity, while an excessive size can lead to overfitting models. Another mechanism for controlling overfitting and improving generalization are regularization, which can be applied by introducing *Gaussian noise* (i.e. small random variations in the input data during training), or a *dropout* mechanism, which randomly deactivates certain neurons in each training iteration, forcing the network to depend on different combinations of features.

Another key feature of FFNNs is the use of advanced optimizers, such as the Adam algorithm, which dynamically adjusts the learning rate based on the gradient, allowing for faster and more stable convergence. Also, networks can be evaluated using custom metrics, such as categorical accuracy or F1-macro, which provide detailed insight into the model's performance on specific tasks.

That are the three configurations used in our experiments:

- **Feedforward 1 (ff1)**
  - *Overfitting Mitigation: Gaussian noise (standard deviation of 0.2) is added to the input layer to improve generalization and reduce overfitting.*
  - *Hidden Layers: Two fully connected layers with 256 neurons each, using the ReLU activation function.*
  - *Optimization and Metrics: The Adam optimizer is used with default parameters, and the metrics CategoricalAccuracy and F1-macro are monitored.*
  - *Focus: A simpler architecture balancing learning capacity and computational efficiency, suitable for moderately complex tasks.*
- **Feedforward 2 (ff2)**
  - *Network Depth: Six consecutive fully connected layers with 512 neurons each.*
  - *Normalization and Constraints: Each layer applies a MaxNorm (1.0) weight constraint and Batch Normalization (BN) to stabilize training.*
  - *Activation: ReLU activation is applied after batch normalization.*
  - *Optimization: The Adam optimizer is used with a very low learning rate (1e-5) for improved stability in deeper networks.*
  - *Focus: Designed for more complex tasks where Batch Normalization and MaxNorm constraints are essential for stable training and better convergence.*
- **Feedforward 3 (ff3)**
  - *Network Depth: Eight consecutive fully connected layers with 512 neurons each.*
  - *Regularization: Dropout (50%) is applied after each ReLU activation to reduce overfitting by randomly deactivating neurons during training.*
  - *Constraints: Each layer uses a MaxNorm (1.0).*
  - *Optimization: Same as Configuration 2.*
  - *Focus: Best suited for scenarios with a high risk of overfitting, leveraging Dropout for effective regularization in highly parameterized networks.*



chbmit_ffnn

patient	format	conf	task
chb22	21x14	ff1	multiclass
chb10	21x14	ff1	multiclass
chb22	21x14	ff1	binnary
chb10	21x14	ff1	binnary
chb22	21x14	ff2	multiclass
chb10	21x14	ff2	multiclass
chb22	21x14	ff2	binnary
chb10	21x14	ff2	binnary
chb22	21x14	ff3	multiclass
chb10	21x14	ff3	multiclass
chb22	21x14	ff3	binnary
chb10	21x14	ff3	binnary

TABLE 9. Experiments run on CHBMIT dataset using FFNNs

4.2. Convolutional Neural Networks

Convolutional neural networks (CNNs) are a machine learning technique specifically designed to process data with spatial or temporal structures, such as images, signals, or time series. By leveraging the local correlation of the data through the application of convolution operations, CNNs extract meaningful patterns from small regions in the input space. Their operation is based on convolutional layers that use filters to explore local features of the data. These filters move across the input and generate feature maps that highlight specific patterns as they progress through the network [12].

A distinctive aspect of CNNs is their ability to efficiently reduce the dimensionality of the data without losing critical information. This is achieved through techniques such as *pooling*, which selects relevant values from regions of the feature map, or *batch normalization*, which stabilizes and accelerates the learning process by adjusting activations within each mini-batch, ensuring more efficient convergence.

Another important feature of CNNs is their inherent ability to generalize, as the use of shared filters across the input significantly reduces the number of model parameters. However, to avoid issues such as overfitting, additional regularization mechanisms can be incorporated, such as weight normalization (MaxNorm) or the aforementioned dropout technique.

That are the configurations used in the experiments:

➔ **CNN1**

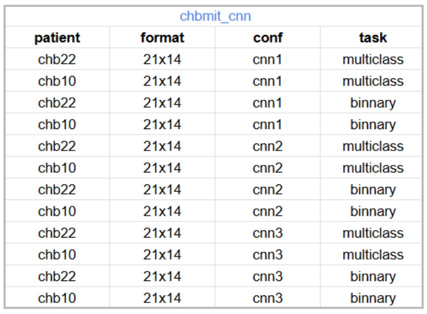
- *Architecture: Two convolutional blocks with 64 and 128 filters, respectively. Each block follows the sequence: Conv2D → Batch Normalization → ReLU → MaxPooling2D.*
- *Feature Extraction: The Conv2D layers extract features from the input, while MaxPooling2D reduces spatial dimensions, improving computational efficiency.*
- *Regularization: A MaxNorm (1.0) kernel constraint is applied to convolutional layers and MaxNorm (3.0) to the dense output layer.*
- *Dense Layer: After flattening, a softmax output layer is used for classification.*
- *Optimization: The Adam optimizer with a learning rate of 1e-2 ensures efficient gradient updates.*
- *Focus: A straightforward CNN designed for standard image classification tasks, balancing performance and computational cost.*

➔ **CNN2**

- *Architecture: Two residual blocks, each containing four Conv2D layers with 64 and 128 filters, respectively. The outputs of these layers are concatenated before applying MaxPooling2D.*
- *Feature Extraction: Residual connections via concatenation allow richer feature representation while mitigating the risk of vanishing gradients.*
- *Normalization: Batch Normalization (BN) stabilizes the training process.*
- *Dense Layer: After flattening, a softmax output layer is used for final classification.*
- *Optimization: The Adam optimizer with default settings ensures stable learning.*
- *Focus: More advanced architecture designed for complex tasks requiring deeper feature extraction and better generalization.*

➔ **Time-delayed (CNN8)**

- *Architecture: A Time-Delayed CNN -TDCNN- is designed for time-series data with an input shape of (T, 21, 14), where T represents time steps.*
- *Convolutional Block: A single Conv2D layer (128 filters, kernel size (3, 21)) extracts temporal patterns across channels and variables.*
- *Fully Connected Layers: After flattening, two dense layers with 2048 and 1024 neurons are applied, each followed by Batch Normalization and ReLU activation.*
- *Output Layer: A softmax layer predicts the final class.*
- *Optimization: The Adam optimizer, using a very low learning rate (1e-6) ensures stability in training.*
- *Focus: Specialized for time-series analysis, effectively capturing temporal dependencies and sequential patterns.*



chbmit_cnn

patient	format	conf	task
chb22	21x14	cnn1	multiclass
chb10	21x14	cnn1	multiclass
chb22	21x14	cnn1	binnary
chb10	21x14	cnn1	binnary
chb22	21x14	cnn2	multiclass
chb10	21x14	cnn2	multiclass
chb22	21x14	cnn2	binnary
chb10	21x14	cnn2	binnary
chb22	21x14	cnn3	multiclass
chb10	21x14	cnn3	multiclass
chb22	21x14	cnn3	binnary
chb10	21x14	cnn3	binnary

TABLE 10. Experiments run on CHBMIT dataset using CNNs

Results and discussion — DIGITS

We can see that all the models give us predictions with practically the same accuracy for both the training and test data. This might indicate that these models do not suffer from overfitting.

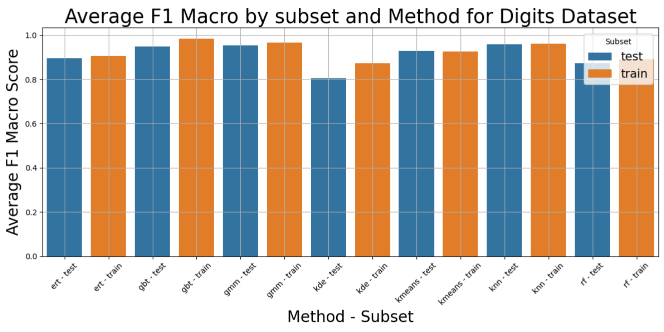


FIGURE 1. Average F1-macro by subset and technical approach — DIGITS dataset.

To evaluate the models' performance, the following graphs only use the test data.

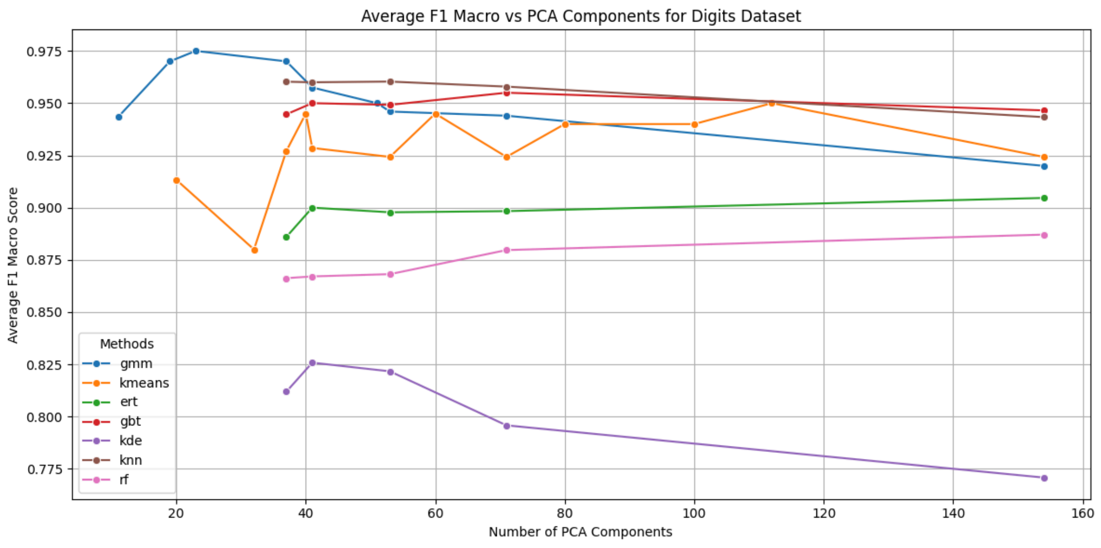


FIGURE 2. Average F1-macro vs number of PCA components for each technical approach — DIGITS dataset.

In graph 2, we can observe how the performance of each model varies depending on the number of PCA components used. We can also see that the model with the best performance for this dataset is GMM when using between 23 PCA components, as we obtain the highest f1-macro score. Also, we should recall that the Kernel density estimation technique (KDE) offers a significantly lower f1-macro than other techniques, even showing a decay on this score when analyzing more principal components.

For both datasets, we prepared a specific graphical analysis for each of the four aforementioned "technical approaches": Clustering/Density, Tree-based, Memory-based and Neural Networking. However, due to the course schedule being modified, this last approach is only used for the CHBMIT dataset.

Clustering/Density techniques

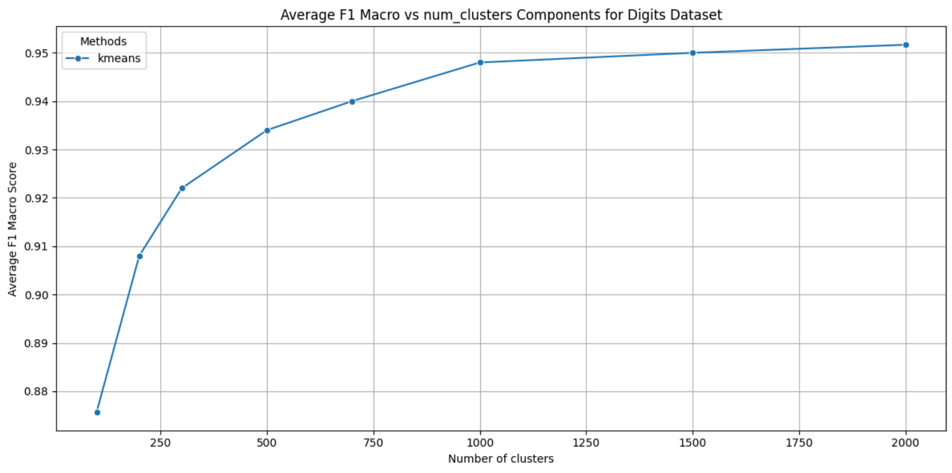


FIGURE 3. Average F1-macro vs number of clusters — DIGITS dataset.

In graph 3, we can see the performance of the k-means model depending on the number of clusters, and we observe that this performance follows a logarithmic progression. The f1-macro metric improves significantly up to 1000 clusters, after which further improvements become practically negligible.

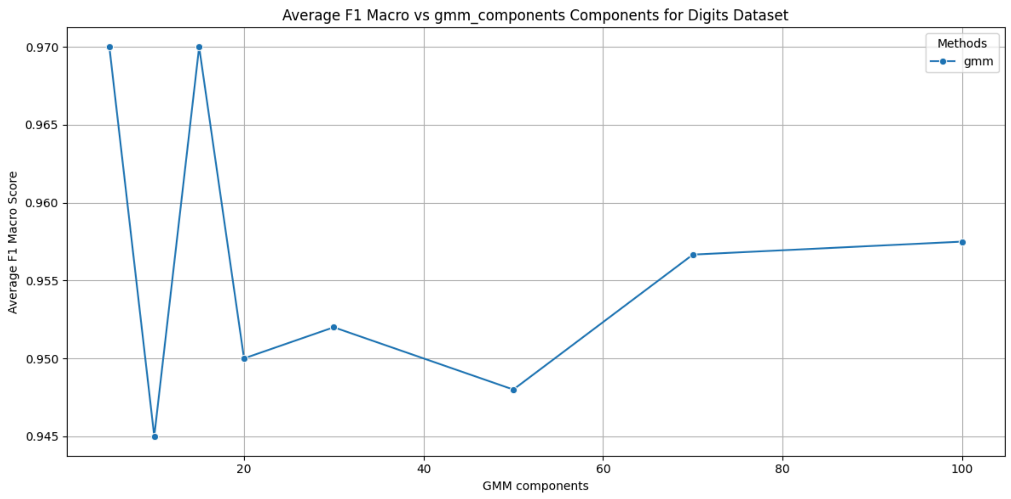


FIGURE 4. Average F1-macro vs GMM components — DIGITS dataset.

This graph shows how the performance of the GMM model varies with the number of components applied. Not many conclusions can be drawn, as the performance evolution appears to behave almost randomly as components increase. Moreover, if we see the Y axis, we can appreciate that the variation is only of 0.025 points of the F1-macro.

Tree-based techniques

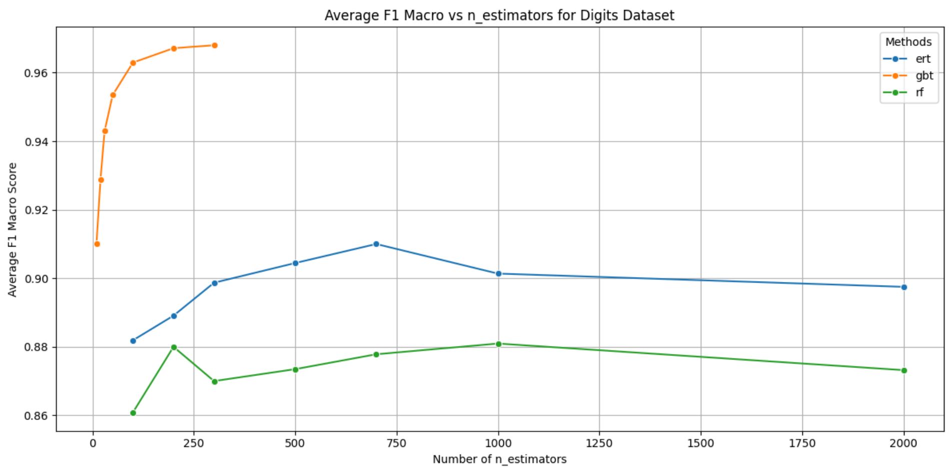


FIGURE 5. Average F1-macro vs number of estimators — DIGITS dataset.

In graph 5, attached above, we can see the performance of the three tree-based methods —RF, ERT, and GBT— depending on the number of estimators used. We observe that GBT achieves the best performance and requires fewer estimators to reach its maximum performance. This happens because the GBT technique needs more simple trees to work better, as it is a boosting method that is based on weak learners.

In Random Forest, that parameter seems not to have an important influence in the performance, at least with the values proved, and in ERT we can see a slight improvement until 700 estimators.

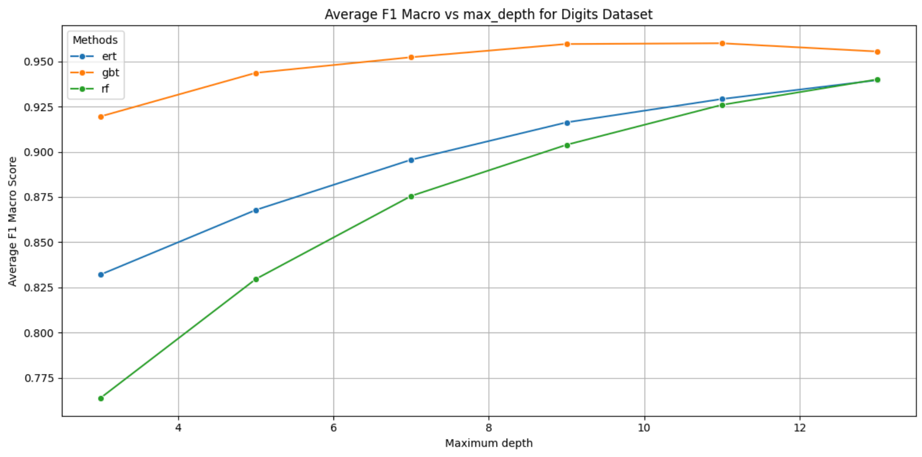


FIGURE 6. Average F1-macro vs maximum tree depth — DIGITS dataset.

The following graph illustrates how the models' performance change when varying the maximum tree depth. Once again, we see that GBT is the method with the best performance and requires less depth to reach its peak performance, explained for the same reason that in the number of estimators. The best maximum depth for GBT is 9, and for ERT and RF is 13 (being RF the method that has a bigger dependence on the depth to work better).

Memory-based techniques

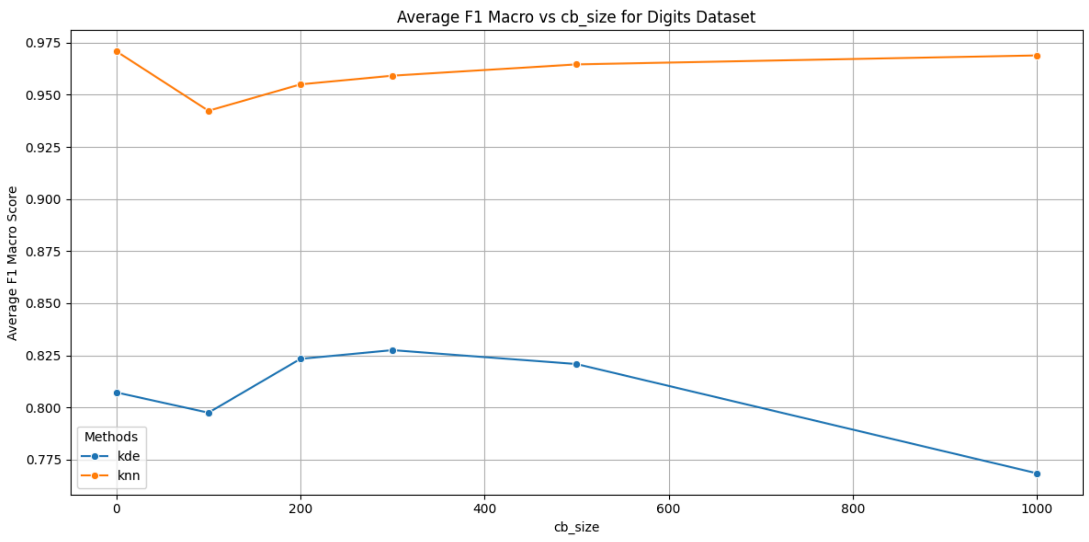


FIGURE 7. Average F1-macro vs codebook size — DIGITS dataset.

Here, we corroborate what we saw on graph 2: the Kernel Density Estimation shows a worse performance than the K Nearest Neighbours.

If we observe the KNN method, the variation of the codebook size seems not to have a significant influence on the results of the model. Although, with the KDE technique we obtain the best results with a medium codebook size, being 200 the best value in terms of the relation quality resources used.

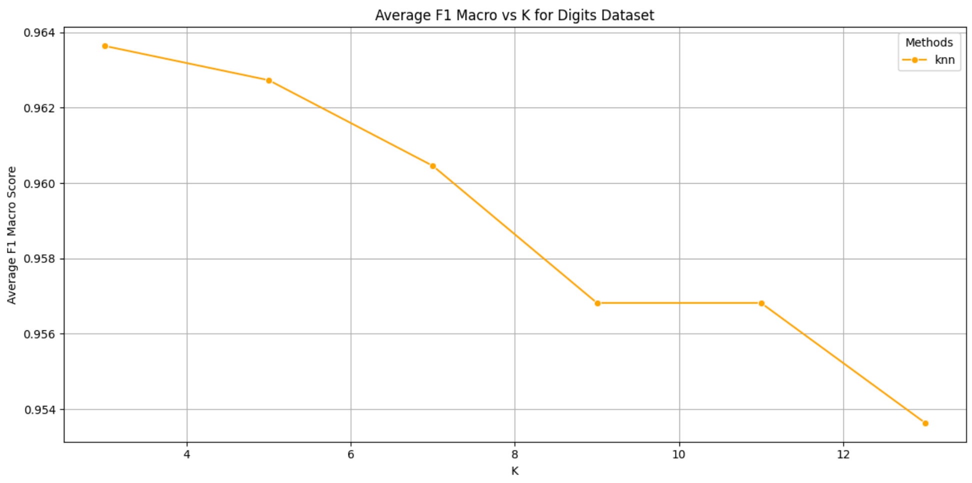


FIGURE 8. Average F1-macro vs number of neighbours (K) — DIGITS dataset.

Now, we analyse in graph 8 the number of neighbours that we should use in the KNN method for the aforementioned dataset. The variation of the f1-macro is very small (0.01) but we see a tendency in the results; the fewer number of neighbors we use the better results we obtain. So, the best number of neighbours according to our experiments is 3.

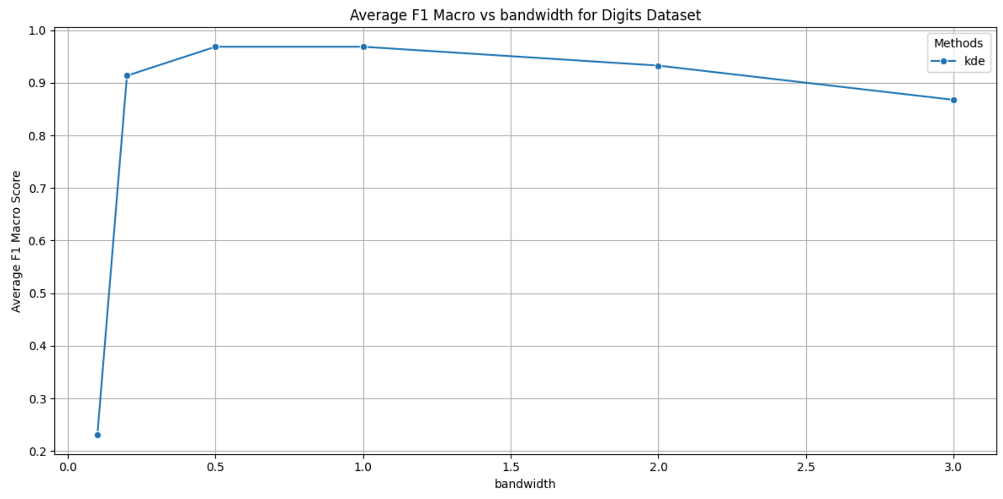


FIGURE 9. Average F1-macro vs bandwidth— DIGITS dataset.

In graph 9, we see the importance of the bandwidth in the KDE technique, being 0.5 and 1 the best parameter values to work with this dataset. In any case, we should remark the significantly low f1-macro obtained when iterating the lowest value for bandwidth in our experiments.

Finally, now that we know the best hyperparameters for the two memory-based methods, we have remade the graph on figure 7 but with fixing the values of bandwidth and K to 1 and 3 respectively. Now, we see completely different results, as both methods have very similar performance. Moreover, we see that KNN reaches stability with a codebook size of 500, and KDE with 200, but in the two cases the variation of the F1 macro is small (0.01 in KDE and 0.02 in KNN).

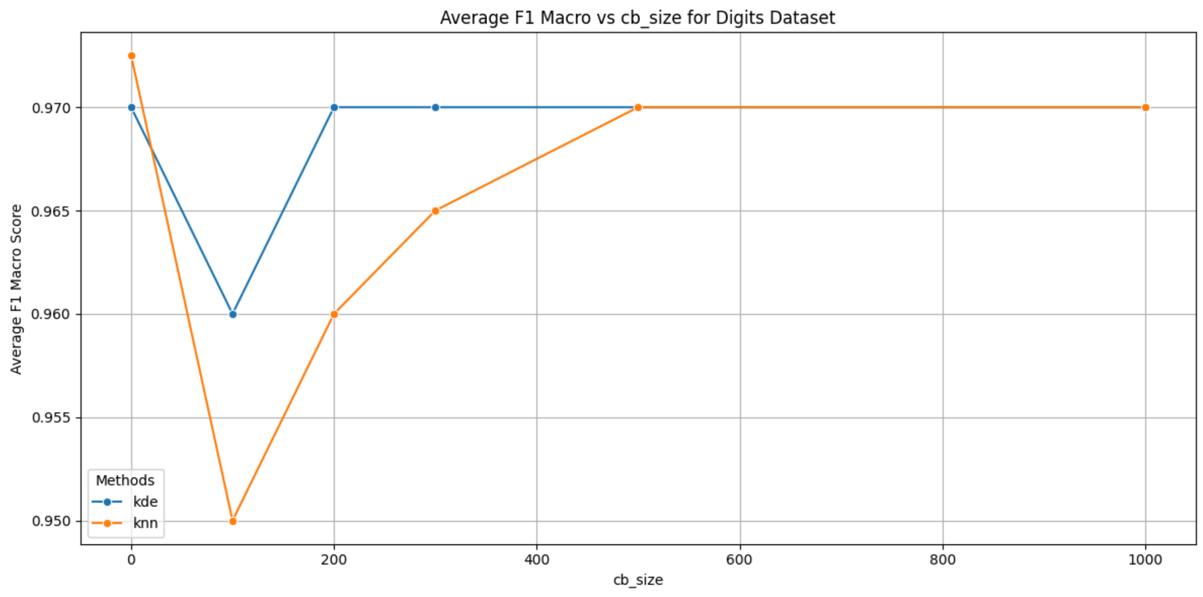


FIGURE 10. Average F1-macro vs codebook size, with optimal hyperparameters— DIGITS dataset.

Results and discussion — CHBMIT

For this dataset, we can observe that all methods show a significant variation between the train and test results, indicating that overfitting occurred during the training of the models.

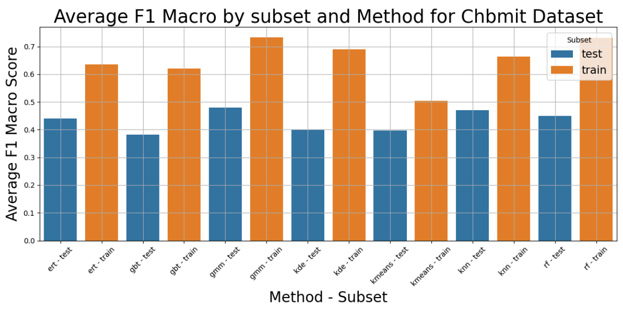
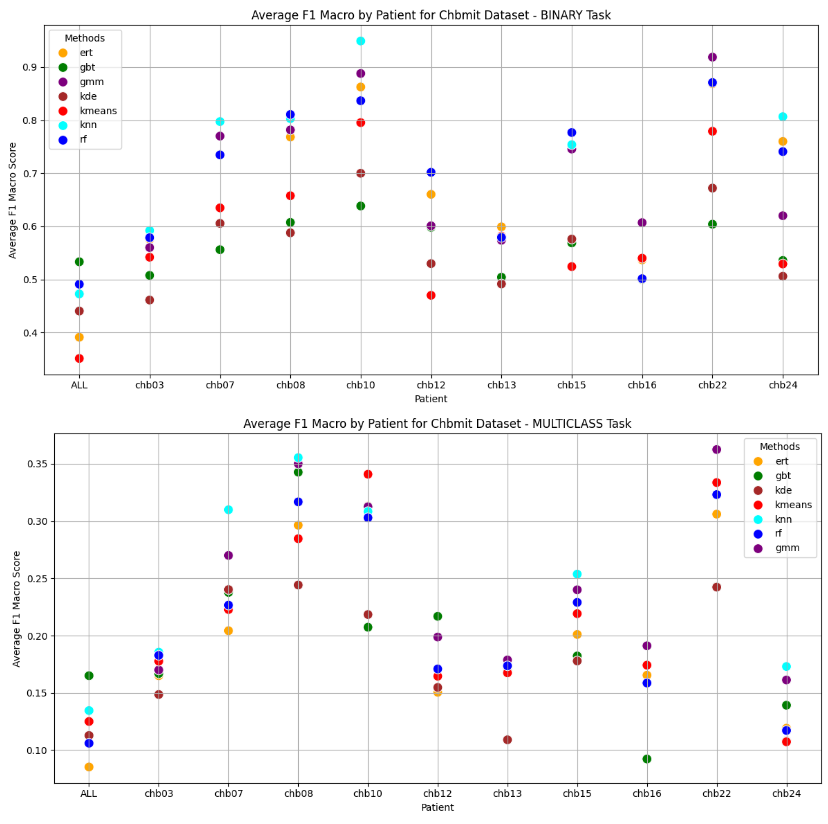


FIGURE 11. Average F1-macro by subset and technical approach — CHBMIT dataset.

We can also see evidence that the performance of the models with this dataset is much lower compared to their performance with the DIGITS dataset; the highest f1-macro scores for test subsets here are near 0.5, while in the previous dataset are higher than 0.9.

Again, a more detailed analysis will now be conducted using only the test data.



FIGURES 12a, 12b. Average F1-macro by technical approach and patient analysed, for binary (a) and multiclass (b) classification tasks — CHBMIT dataset.

In these graphs, we can see how each model adapts to different patients as well as to all patients together. In the top graph -12a- we analyse the performance for binary classification, while in the bottom graph -12b- we focus on 6-class classification. We notice that, in most cases, the best-performing model is, again, Gaussian Mixture Modelling (GMM), and the patients who are easiest to predict are patients 10 and 22. We also obtained remarkable results from the K-Nearest-Neighbours (KNN) technique.

On the other hand, when analyzing ALL patients, we see that GBT gets far better results than the other models, but this performance is much lower than the average of the training made for each patient individually.

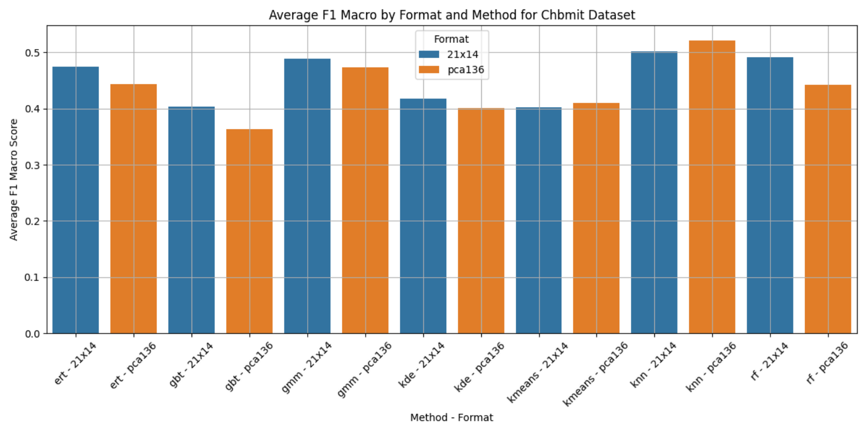


FIGURE 13. Average F1-macro by format and technical approach — CHBMIT dataset.

Here, on the histogram on figure 13, if we compare the f1-macro results for each ML technique and each format, we can see that 21x14 gives us the best scores for five out of the seven techniques. PCA-141 format is not compared because it is only used to train with all the patients at the same time.

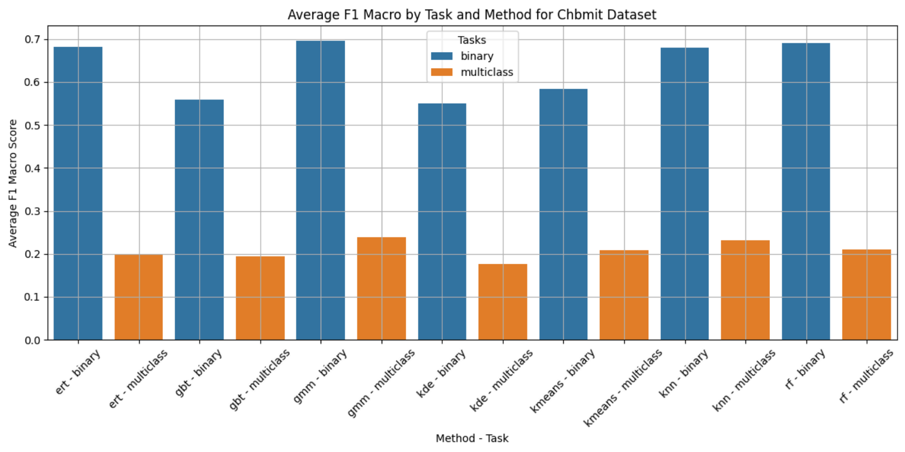


FIGURE 14. Average F1-macro by classification task and technical approach — CHBMIT dataset.

Moreover, if we compare the predictive power of binary and multiclass classification tasks, there is a very significant difference between the performance of both, with values over 0.55 in the f1-macro for all binary tasks and values under 0.25 for all multiclass tasks. This could be happening due to the overfitting commented before, as well as the unbalance in the classes.

In the following pages we will analyse the results focusing on each technical approach, just as explained before:

Clustering/Density techniques

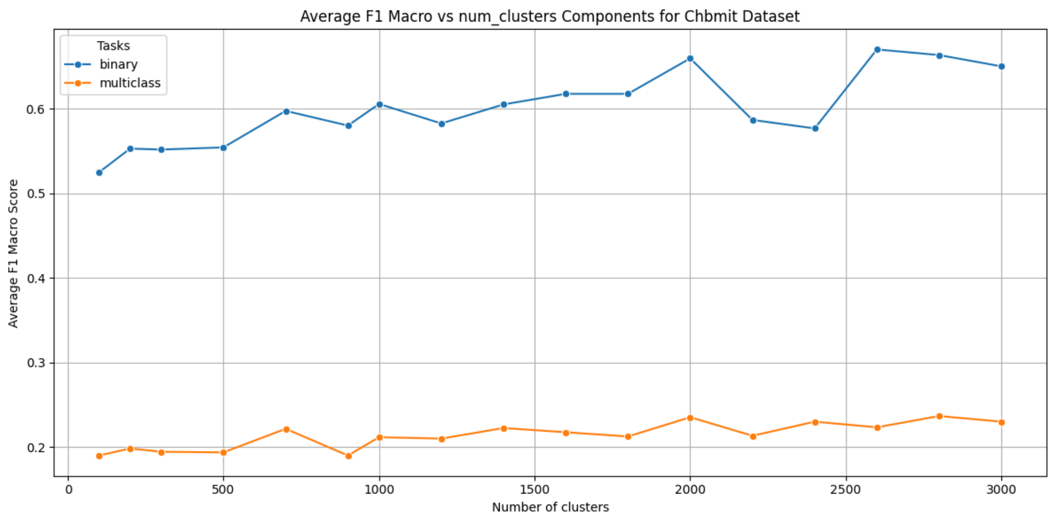


FIGURE 15. Average F1-macro vs number of clusters, by classification task — CHBMIT dataset.

In graph 15, we can see how the performance of the k-means model varies depending on the number of clusters, separated by the type of classification. We observe that for binary classification there is a significant improvement as the number of clusters increases with 2000 being the best value in the resources performance, but for multiclass classification the improvement is minimal and none of the iterations can give us acceptable results.

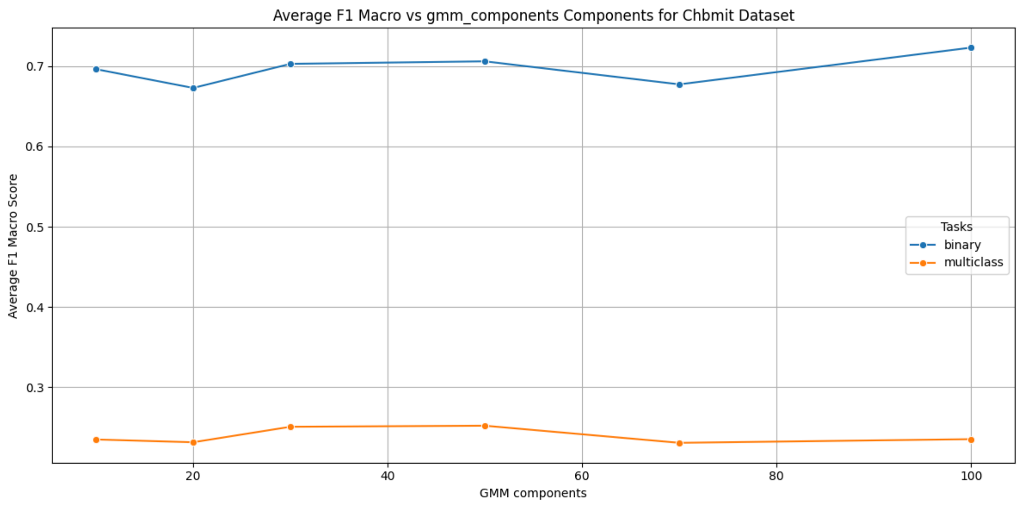


FIGURE 16. Average F1-macro vs GMM components, by classification task — CHBMIT dataset.

In contrast, we can check this graph, where we can see how the performance changes depending on the GMM components, separated by classification type. We observe that there is little to no variation in the results with regards to changes on this variable.

Tree-based techniques

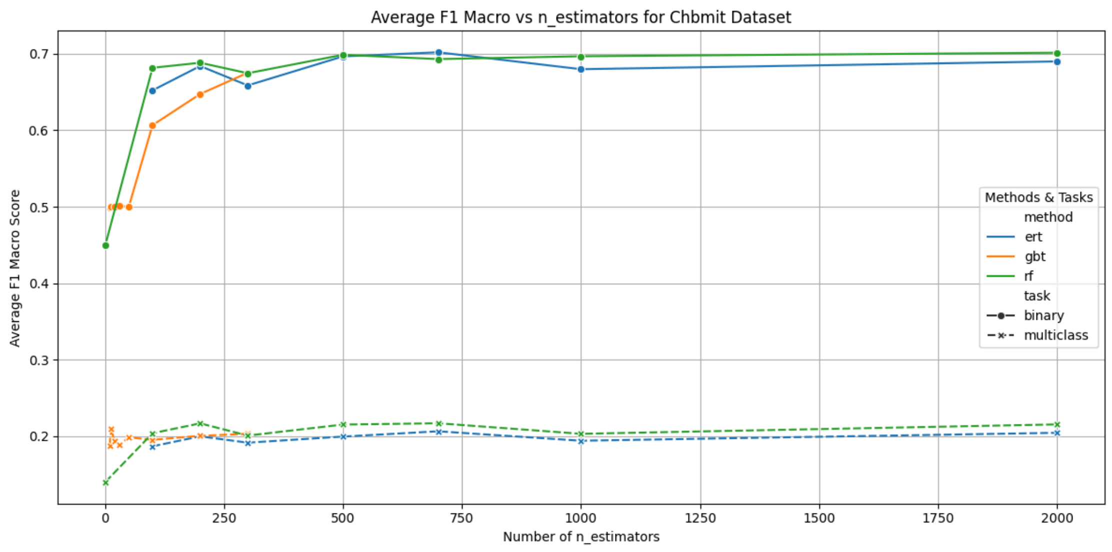


FIGURE 17. Average F1-macro vs number of estimators, by classification task — CHBMIT dataset.

Now, we focus on how the performance changes in tree-based techniques when we change the number of estimators, separated by classification type. We see that for binary classification, both RF and ERT reach a point close to their best performance with 200 estimators, while GBT achieves a similar value with 300 estimators. In contrast, for multiclass classification, there is almost no variation in performance when changing the number of estimators, with f1-macro scores being near 0.2.

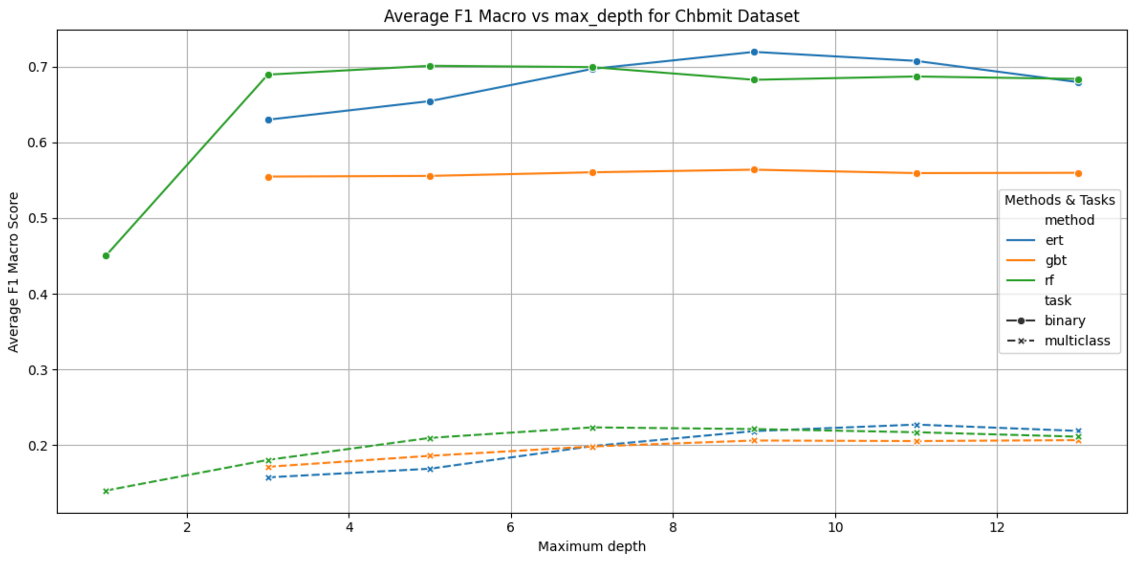
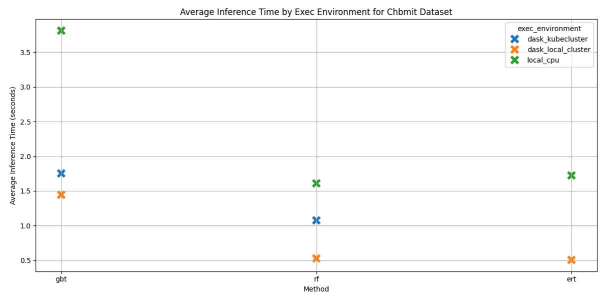


FIGURE 18. Average F1-macro vs maximum tree depth, by classification task — CHBMIT dataset.

If we focus now on the depth of our tree analysis, we see the same difference regarding the type of classification task assigned, with binary classification tasks performing far better. In this case, we observe a better prediction power from RF and ERT techniques rather than gradient boosting. Anyways, we should take into account the computational cost of each model in order to determine which technique is better.



In the graph attached on the left, we get a comparison of the three different execution environments used —local CPU, Dask local clusters and Kubernetes clusters— regarding their average inference time. This refers to the duration of time that it takes for a trained model to make predictions on new given data [10]. Thus, we can observe that the executions using Dask local clusters are quicker than the other tree-based techniques; Also, we must take into account that executions of gradient boosting trees, although delivering best performances, consume more time than the other techniques.

FIGURE 19. Average inference time for tree-based techniques, by execution environment — CHBMIT dataset.

Memory-based techniques

Now we are going to analyze our results when changing hyperparameters for the memory-based techniques:

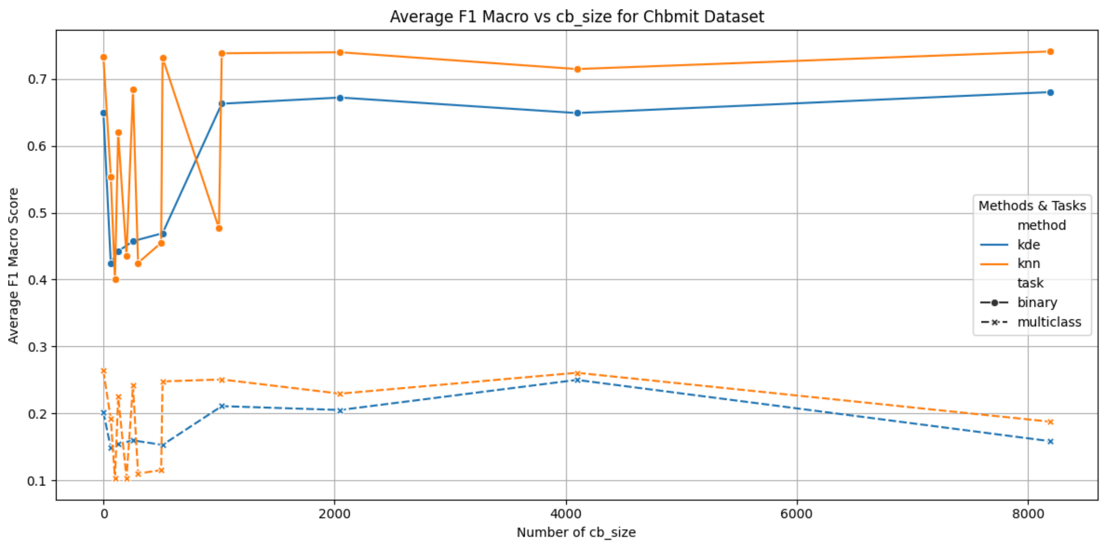
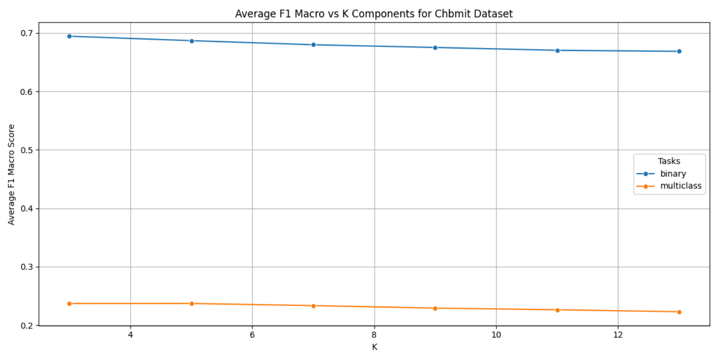


FIGURE 20. Average F1-macro vs codebook size, by classification task — CHBMIT dataset.

The codebook size seems to be very important in the two methods giving very irregular results until a size of 1024, and then the performance remains stable: thus, we consider this is the best value for those two methods.



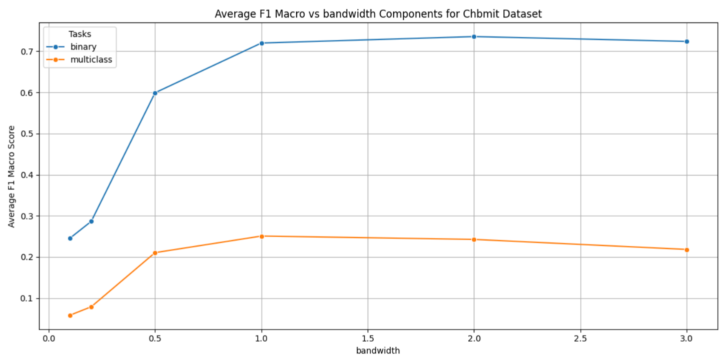


FIGURE 21 (left). Average F1-macro vs number of neighbours (K), by classification task — CHBMIT dataset.
FIGURE 22 (right). Average F1-macro vs bandwidth, by classification task — CHBMIT dataset.

In KNN, the number of neighbours (as depicted in figure 21) seems not to have importance in the performance of the model. On the other hand, the bandwidth (reflected in figure 22) is very relevant in the performance of the KDE method; the value that optimizes the performance is 1 in both binary and multiclass classification.

We attach here the graph 20 reworked, using these aforementioned optimal hyperparameters; anyways, we do not appreciate any relevant differences.

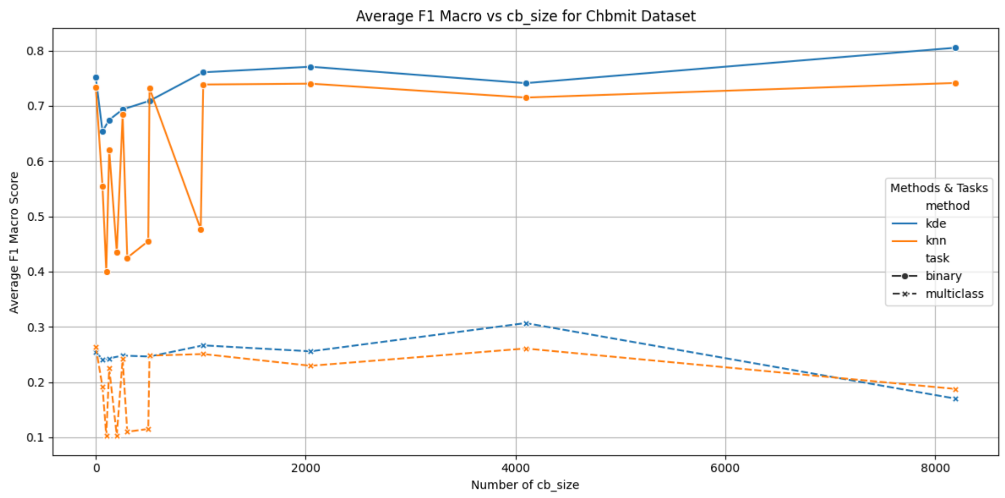
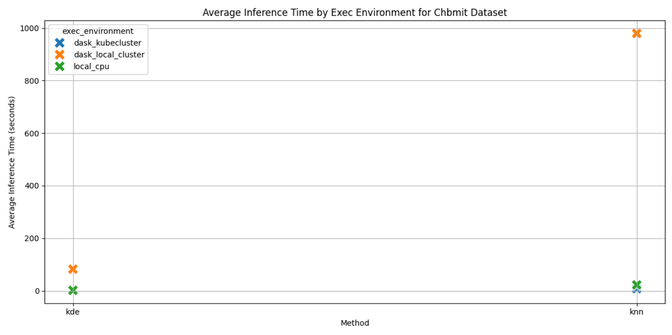


FIGURE 23. Average F1-macro vs CB size with optimal hyperparameters, by classification task — CHBMIT dataset.



The time used in the memory based techniques is exponentially higher if we do not use kubernetes clusters, especially with the KNN technique. Thus, we conclude that it is really important to have a good environment to execute those methods in such complex databases.

FIGURE 24. Average inference time for memory-based techniques, by execution environment — CHBMIT dataset.

Neural Networking

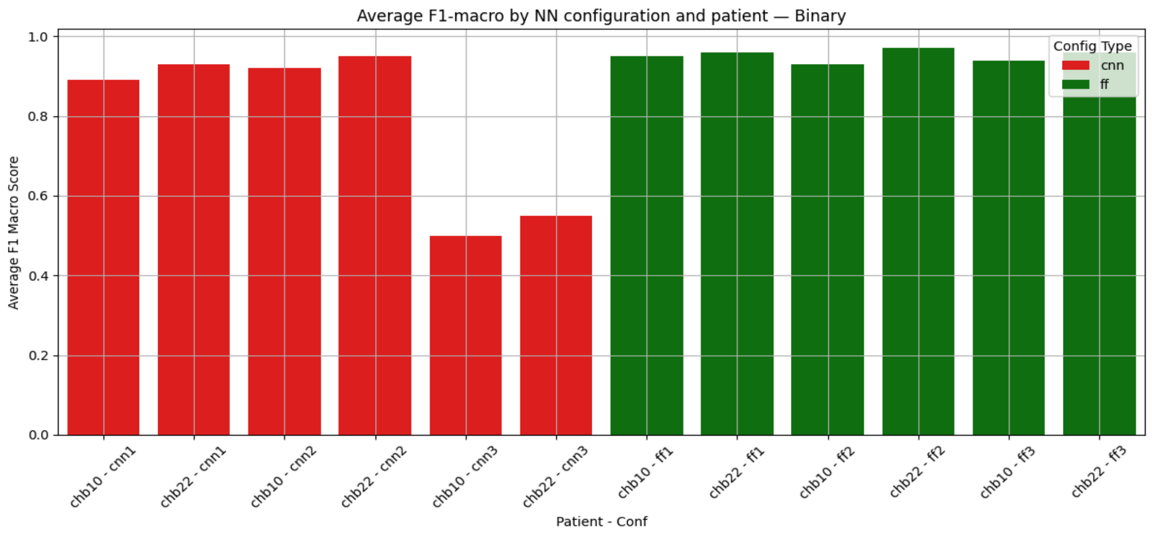


FIGURE 25. Average F1-macro by NN configuration for binary task — CHBMIT dataset, patients 10 and 22.

In this section, we have analyzed two patients (chb10 and chb22, the best-predicted) using three different configurations for feed forward and three configurations for convolutional neural networks. The performance of the feed forward models is slightly better than the CNN ones, but all configurations in general obtain better results than the classical methods analysed before. The only neural network method whose performance has been worse is the time-delayed convolutional neural network, {cnn8}.

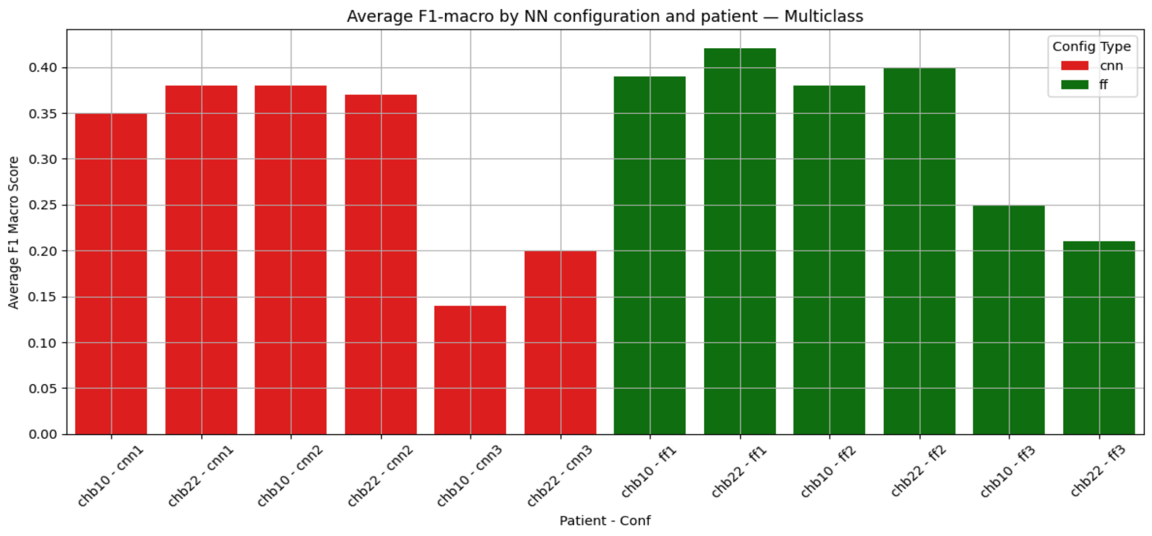


FIGURE 26. Average F1-macro by NN configuration for multiclass task — CHBMIT dataset, patients 10 and 22.

On the other hand, we see that for multiclass classification the results, despite being better than using classical methods, are far away to be good results, reaching around 0.40 of f1-macro in the best of the cases. The feed forward methods have, in general terms, a better performance than convolutional ones; yet again configurations {cnn8} and {ff3} work significantly worse than the others.

Conclusions

The models applied to the **DIGITS** dataset demonstrate consistent performance, with no overfitting observed between the training and test subsets. The Gaussian Mixture Model (GMM) stands out as the best-performing model, particularly when using around 23 principal components. This highlights the effectiveness of GMM on simple, well-structured data with low variability, which is the case of the referred dataset. Focusing again on the performance, most experiments deliver a f1-macro score of more than 0.9, which is not only consistent but also considerably high.

In contrast, the **CHBMIT** dataset, which contains complex medical data (EEG), poses greater challenges. The models show significant overfitting, with f1-macro scores lower than those for DIGITS, particularly for multiclass tasks with values around 0.3. This could be happening due to noise and class imbalance. The GMM remains one of the most effective models, especially for binary classification tasks (between seizures and non-seizures), with patients 10 and 22 being the easiest to predict. However, when analyzing all patients together, Gradient-boosted Trees deliver a better performance, although this is an average of performance across individual patients.

In a nutshell, the DIGITS dataset is relatively straightforward, with stable and predictable results from simpler models. In contrast, CHBMIT presents a more complex scenario due to its noisy and imbalanced medical data, requiring more careful model tuning to achieve good performance. While the first dataset allows for easy classification, the second one emphasizes the need for specialized approaches to handle real-world medical data challenges.

References

[1] Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

[2] Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*(1), 21–27.

[3] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5), 1189–1232.

[4] Geurts, P., Ernst, D. & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1), 2–42.

[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.

[6] Han, J., Kamber, M., & Pei, J. (2011). *Data mining: Concepts and techniques (3rd ed.)*. Morgan Kaufmann.

[7] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

[8] Matplotlib Development Team. (2024). *Matplotlib: Visualization with Python*. Consulted November 6, 2024 on https://matplotlib.org/stable/contents.html

[9] Pandas Development Team. (2024). *Pandas: Python Data Analysis Library*. Consulted November 6, 2024 on https://pandas.pydata.org/docs/

[10] Quora Anonymous Author. (2024). *What is inference time in deep learning?* Quora. Consulted November 15, 2024 on https://www.quora.com/What-is-inference-time-in-deep-learning

[11] Silverman, B.W. (1986) Density Estimation for Statistics and Data Analysis. Chapman & Hall, London.

[12] Zhang, Z., Cui, P., & Zhu, W. (2021). *Deep Learning for Data Mining Applications.* Springer.